D7.1
On Programmable Corpora
Report and Prototype (DraCor)

Authors of the Report: Ingo Börner, Peer Trilcke

Concept & Development of the DraCor Prototype: Frank Fischer, Carsten Milling – Ingo Börner, Mathias Göbel, Mark Schwindt, Daniil Skorinkin, Henny Sluyter-Gäthje, Peer Trilcke

Date: February 28, 2023

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101004984

Project Acronym: CLS INFRA
Project Full Title: Computational Literary Studies Infrastructure
Grant Agreement No.: 101004984

Deliverable/Document Information

Deliverable No.: D7.1
Deliverable Title: On Programmable Corpora. Report and Prototype (DraCor)
Authors: Ingo Börner, Peer Trilcke
Concept & Development of the DraCor Prototype: Frank Fischer, Carsten Milling – Ingo Börner, Mathias Göbel, Mark Schwindt, Daniil Skorinkin, Henny Sluyter-Gäthje, Peer Trilcke
Review: Maciej Eder, Michał Mrugalski, Álvaro Pérez Pozo, Salvador Ros, Daniil Skorinkin, Henny Sluyter-Gäthje
Dissemination Level: PUBLIC

DOI: 10.5281/zenodo.7664964

Document History

Version/Date	Changes/Approval	Author/Approved by
2022-02-17	Version for Review	Ingo Börner, Peer Trilcke
2022-02-27	Version for Submission	Ingo Börner, Peer Trilcke

Index

List of Figures

List of Tables

About this Deliverable

In addition to this report, this Deliverable D7.1 includes a technical prototype for Programmable Corpora: the Drama Corpora Platform "DraCor", which has been under construction since 2018 and was most recently further developed within CLS INFRA.

The main access points to the different components of the DraCor system are:

- Code and Data on GitHub: https://github.com/dracor-org
- DraCor Front-end: https://dracor.org/
- DraCor API: https://dracor.org/doc/api

Numerous scholars participate in the development of DraCor and in the curation of the corpora (the latter is not part of this Deliverable). For an overview, see https://dracor.org/doc/credits.

For the DraCor platform as a whole, these persons are responsible:

- Editor-in-chief: Frank Fischer (Freie Universität Berlin)
- Co-editors: Peer Trilcke (University of Potsdam), Julia Jennifer Beine (Ruhr University Bochum), Daniil Skorinkin (University of Potsdam)
- Technical lead: Carsten Milling (University of Potsdam)
- Technical co-leads: Ingo Börner (University of Potsdam), Mathias Göbel (University of Göttingen)
- Art director: Mark Schwindt (Ruhr University Bochum)

1. Publishable Summary

While the discipline of Computational Literary Studies (CLS) consolidates, infrastructural challenges are arising that have to be addressed to ensure that good, sustainable and open scholarship can be carried out in this dynamic field of Digital Humanities research. In this situation, Work Package 7 of the CLS project, entitled "Building the Ecosystem of and for Programmable Corpora", is developing a small-scale, but highly functional prototype for an infrastructural ecosystem for CLS research, following the concept of a network-based software architecture. The prototype, implemented as the multi-component system "DraCor" (Drama Corpora Platform), realizes the concept of "Programmable Corpora", which is defined as *corpora that expose an open, transparently documented and (at least partly) research-driven API to make texts machine-actionable*. This report gives a detailed description of the DraCor system as a prototype for "Programmable Corpora". It also shares two experiments in adapting and transferring the approach of an API-based CLS research infrastructure to other systems and resources.

2. Introduction and Methodology

2.1 Towards an Infrastructural Ecosystem for CLS

With the ongoing formation of the discipline of Computational Literary Studies (CLS)—apparent for instance in the founding of a dedicated journal (JCLS)[1] and an annual conference (CCLS)[2]—a whole set of pragmatic and infrastructural challenges arise that have to be addressed to ensure that good, sustainable and open scholarship can be carried out in this emerging field. The CLS INFRA project, where the present report was prepared, has set itself the task of conceptualizing and prototyping both community-driven and infrastructural solutions to these disciplinary challenges, to support and promote digitally progressive as well as socially relevant research on the European (and the world's) literary cultural heritage.

Some of the major challenges of data driven CLS research arise from what we will address as the problems of *dispersion,* of *heterogeneity,* and of *instability* of the field's resources, especially of its (datafied) epistemic objects and its (algorithmized) methods. While we will explain in more detail later in this report what we mean by these problems, we will give some rather general examples here: The epistemic *objects* of CLS research (i.e. corpora of literary texts) are, for example, often prepared in the context of specific research projects, taking into account different needs and following different standards (or different interpretations of the same standards), and they are stored in varying locations with differing ways of access; digital *methods*

[1] https://jcls.io.
[2] https://jcls.io/site/conference.

are also often developed in specific projects (and the corresponding software respectively) and the code is published at again other locations, so that it can easily become a considerable challenge to just read the data from one project into the software of another project. The resources are thus dispersed and it is often difficult to connect or integrate them for further research. In turn, the results obtained for a specific study may depend heavily on the states and contexts of the data and software involved, so that it is not just difficult to reproduce CLS research but also to conduct comparative or follow-up research that builds on existing studies.

On the one hand, these challenges are a result of the current lack of standards, routines, and conventions in a field that, as a starting community, in many respects is still in an experimental phase and in which routines for the review and publication of research software[3] and data[4] have only recently begun to consolidate. On the other hand, the current challenges might be an effect of missing feasible disciplinary infrastructures. But before such infrastructures can be developed, there needs to be discussions on the architectural styles of research environments in CLS with which the challenges can be met. Thus, at a more abstract level, efforts towards standardization and normalization are needed to develop a machine-readable domain model of CLS (and presumably literary studies in general) and to formulate taxonomies as well as controlled vocabularies. At the same time, at a more concrete level, infrastructural prototypes are needed in which we experiment with ideas of a disciplinary ecosystem for CLS. The latter is what this report (and the accompanying digital prototype) is about.

The long-term perspective of the ideas and prototypes presented here is what, following Roy Fielding, a pioneer of the World Wide Web and creator of the Representational State Transfer (REST) architectural style, could be called a “network-based software architecture” (Fielding 2000) for CLS. Therefore, in the course of CLS INFRA Work Package 7, we want to develop ideas on how the above-mentioned problems of *dispersion, heterogeneity*, and *instability* of the CLS field’s resources can be addressed with appropriate infrastructural concepts. In that sense, we will approach the problem of dispersion with a *distributed architecture*, the problem of heterogeneity with *workflows of homogenization*, and the problem of instability with *versioning techniques*.

We will develop our ideas and our prototype(s) starting from a conceptual vision inspired by the unfinished work “A Programmable Web” of programmer and open data activist Aaron Swartz, who died in 2013. His “A Programmable Web” outlines a situation in which applications are no longer “not just another tool [...] to use, but part of the ecology—a section of the programmable web” (Swartz 2013: 7). While the CLS INFRA project as a whole is developing ideas for a digital ecosystem of CLS and testing these ideas through studies, community activities, and technical prototypes, CLS INFRA’s Work Package 7—whose activities are subject of this

[3] Cf. e.g. the Guidelines https://jcls.io/site/code-data-review.
[4] Cf. https://openhumanitiesdata.metajnl.com.

report—–focuses on a central building block of this ecosystem: on the concept of “Programmable Corpora” (Fischer et al. 2019) that emerges from Swartz’s considerations. One of Swartz’s visions was that APIs (Application Programming Interfaces) are not only added to web resources and web applications as “an afterthought or a completely separate piece", but that, someday, APIs “naturally grow out of” web resources and web applications (Swartz 2013: 7), so that they are always essentially open to the web. So once this somewhat “natural growth” of APIs really sets in, the web becomes fully “programmable”, according to Swartz. APIs in this sense are the core building blocks of Swartz's concept – and they are correspondingly central to our concept of “Programmable Corpora”, which in a first step can be understood as *corpora that expose an API* to make texts machine-actionable.

In our prototype-based work on an idea of a CLS research infrastructure, we are deliberately starting from *corpora*. Indeed, there are positions that state that, “across many research domains in the humanities and social sciences, the *corpus* has emerged as a major genre of cultural and scientific knowledge” (Gavin 2023: 4). If you focus on the CLS, it can even be said, that—unlike in traditional literary studies, which usually starts from a single literary work or the oeuvre of an author—the corpus has become the central epistemic object,[5] while it is important to underline that the notion of corpora might differ in linguistics and CLS (Mrugalski et al. 2022). In our approach, therefore, it is the corpora that must be made programmable as a first step, thus “growing” out of them the future ecology of CLS—driven by the spread of “naturally growing” APIs, as Swartz would have said.

2.2 A Prototyping Approach

In CLS INFRA’s work package 7, we develop a technical prototype of Programmable Corpora. Prototyping involves, following Budde et al., “producing early working versions (‘prototypes’) of the future application system and experimenting with them” (Budde et al. 1992: 89). The prototype to be developed—“prototype” roughly understood with Naumann and Jenkins as “an individual that exhibits the essential features of a later type” (1982: 29)—can serve as a reduced model[6] for what one day might be the fully functional infrastructure of a network-based ecosystem for CLS. On the way there, the prototyping approach, discussed in software development since the 1980s (Naumann and Jenkins 1982; Budde et al. 1992), sets our methodological guidelines.

First of all, the prototype approach is particularly well suited to the specific starting conditions of our work, where the user needs cannot clearly be defined at the beginning of the

[5] Cf. Deliverable 5.1 “Review of the Data Landscape”, https://doi.org/10.5281/zenodo.6861022 (Mrugalski et al. 2022), and 6.1 “Inventory of existing data sources and formats”, https://doi.org/10.5281/zenodo.7520287 (Ďurčo et al. 2022).
[6] “Model”, here understood with Tavolato and Vincena as “an operational software system which—though somehow limited—gives an impression of how the final system will work” (1984: 438).

project.[7] Thus, the agile method of prototyping allows us to take an “approach to software construction based on experiment and experience” (Budde et al. 1992: 90), where the target system is not fixed, but is conceptually developed and optimized in a series of iterations. As Floyd already noted in 1984, “a prototype should always be considered a learning vehicle providing more precise ideas about what the target system should be like” (Floyd 1984: 3). In this sense, the prototyping approach keeps the development process open not least for visionary interventions, an aspect that seems particularly appropriate given the current dynamic development of the disciplinary field of CLS. As a “learning vehicle”, the prototype of Programmable Corpora developed by us thus eventually also serves as a method for the CLS to become self-aware of its infrastructural needs as well as of its own vision of an infrastructural ecology.

In their early work on prototyping in software development, Tavolata and Vincena (1984: 437) differentiated three methodological approaches that usually are termed prototyping: the “plug-in strategy or incremental delivery approach”, the “evolutionary development method”, and the “throw-it-away approach”. In our adaptation of the prototyping approach, we combine the first and the second method. On one side, our prototype of Programmable Corpora is open for extensions following a plug-in strategy, so that, for example, additional microservices[8] can be (and already have been) connected to our prototype during the course of the project. On the other side, we generally follow the approach of evolutionary development, where a working prototype system is “built and delivered to the user for experimentation”, then “modified […] in a step-by-step fashion to incorporate the experiences of the experimentation”, whereby this “is done in such a way that the prototype evolves gradually into the final product” (Tavolato and Vincena 1984: 437), which in the end might evolve into a “pilot system” (Budde et al. 1992: 91).[9]

In addition, the prototyping process in our work package is designed along four methodological specifications.

- With regard to the overall project CLS INFRA, we understand our approach as *Reflective Prototyping*: The exchange with the other CLS INFRA work packages as well as the input from the reports of the other work packages leads to a reflection of the prototype development, for example in view of new knowledge about user

[7] Rather, in the CLS INFRA project, user needs are identified in a series of tasks and described in corresponding reports. With Schöch, Fileva Dudar 2022, A first report has already been published as Deliverable 3.1, cf. “Baseline Methodological User Needs Analysis”, https://doi.org/10.5281/zenodo.6389333.
[8] For the concept of microservice cf. Nadareishvili et al. 2016: 3, who define: “A microservice is an independently deployable component of bounded scope that supports interoperability through message-based communication.”
[9] This corresponds basically to the "production-driven prototypes" as described by Ruecker for the Digital Humanities (Ruecker 2015: 2).

needs, sharpened approaches to theorizing literary corpora or a more comprehensive idea of the CLS data landscape.
- Second, our development activities explicitly follow a *Research-Driven Prototyping* approach: In our work package, we are not only developing a prototype for Programmable Corpora and thus a model for a research infrastructure of the CLS; rather, at the same time, we are already using this prototype for actual research studies, so that the prototype must prove itself under the real-life conditions of the CLS research.
- Third, our methodological approach follows the idea of *Community-Oriented Prototyping*: In this sense, we accompany the development of the prototype with activities that are intended to encourage its use by the CLS research community in order to stimulate usability testing of the prototype.
- Finally, we follow a *Transfer Prototyping* approach: Selected components of our prototype are adapted (and thus "transferred") to other software settings to test their generalizability and thus their suitability as common building blocks of an infrastructural ecosystem for CLS.

The prototype for Programmable Corpora developed in our work package is the *Drama Corpora Platform* (*DraCor*), which is freely usable via the web and which we will present in more detail in section 5. DraCor is a multicomponent prototype that includes a number of homogenized *corpora* and several *APIs*, some of which are document-based and some of which are research-driven; in addition, the DraCor prototype includes exemplary *microservices*: Corpora, APIs, and microservices will accordingly be the components that we explain and reflect on in more detail throughout this report.

3. CLS Research in Digital Ecosystems between Embeddedness and Instability: Some Key Considerations

Our development of a prototype of Programmable Corpora as a small-scaled vision of an ecosystem of CLS starts from some key considerations about research in digital ecosystems, which we will first briefly introduce before moving on to a detailed description of the prototype.

As we have outlined above, we start by noting the *dispersion* of resources and applications for CLS research. However, our infrastructural approach is not aimed at solving dispersion through centralization, i.e., developing a monolithic infrastructure which would not only be susceptible to both changes and failures but also problematic given the structural diversity of the CLS field. Instead, we want to tackle dispersion by conceptually orienting our prototype to ideas of *distributedness*, and thus to approaches of distributed software architecture. Thereby, we understand distributedness in a positive sense as a mode of (networked) *embeddedness*.

Embeddedness implies the potential for (mutual) interaction. In this sense, thinking about a future infrastructure for CLS could mean looking more closely at modes, ways, and styles of interaction between resources and applications, which in our operationalization means looking in a particular way at interfaces or, more precisely, at application programming interfaces (APIs). How can research software "interact" with corpora? How can websites, apps, or microservices "interact" with corpora? And considering that we in the end are not only speaking of an infrastructure, but of a digital ecosystem, i.e., a *socio*-technical environment: How can *people*, researchers and citizens alike, interact with corpora (and thus with their literary cultural heritage)?

Rethinking dispersion as distributedness means paying attention both to the individual digital components which are used in CLS *and* to the components' way to connect with each other. With respect to our prototyping of a CLS ecosystem, this means, for example, that we are not only working on community-based approaches to homogenizing and federating corpora, but also on how these corpora can be connected to research software or to websites, to name just these two. And it is precisely this ongoing reflection on the possibilities of connection that eventually leads—as Aaron Swartz has put it––to APIs "naturally growing out of" the resources and applications.

So, we change the perspective on dispersion and understand it in terms of distributedness, which is conceptualized as embeddedness and realized through APIs that—as connectors—function as enablers of an interaction-based digital ecosystem of CLS. While such a digital ecosystem, in which the individual components can communicate and connect with each other in an as standardized manner as possible, could be a way of addressing the *dispersion* of applications and resources, it at the same time entails challenges for the stability of research settings. An example may illustrate this challenge of *instability*: CLS research projects regularly integrate generic tools into their analysis pipelines, such as the open-source library spaCy, which features a wide range of methods from Natural Language Processing, e.g., Named Entity Recognition (NER), Part of Speech tagging (POS) or dependency parsing. Like many programs, *spaCy* (Montani et al. 2023) is by no means "finished", but is constantly being developed further; for example, the language models used by *spaCy* are regularly optimized. To include such a dynamic component from the digital ecosystem in an analysis pipeline can thus lead to the situation that—even if the architecture of the pipeline has not changed—the results produced by the pipeline are different. From a pragmatic point of view, such a situation can be addressed (and thus more or less brought under control) by comprehensive documentation and precise versioning.

At the same time, this embeddedness and the associated dynamics of the digital ecosystem must be considered conceptually from an infrastructural perspective and taken into account as a constraint for development activities. In this sense, this challenge is also always reflected in the development work in work package 7. Accordingly, a separate task is devoted precisely to this problem, on which the “Report on versioning requirements of APIs and corpora within CLS” will be published in spring 2024 as deliverable D7.3. The report will present approaches to counteracting instability: on the one hand by corpus versioning using the version control system Git; on the other hand, by stabilizing actual digital research settings and their parameterizations using Docker-based containerization technologies, especially for APIs and methods as microservices.

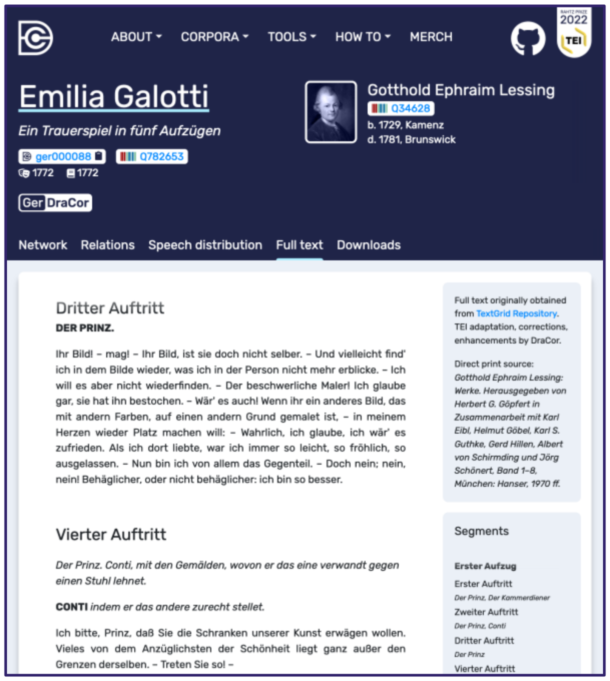
4. The DraCor Prototype in Action. Four Showcases

Before we get into a systematic and technically oriented description of the prototype for Programmable Corpora implemented as DraCor, we will first give an idea of possible uses of the developed prototype by showcasing three examples. Each of the four examples illustrates a different way of using the DraCor prototype in a CLS research environment: first, an approach to geo-based visualization of corpus metadata using Linked Open Data; second, an API-based approach to standardized extraction of specific textual data across different corpora; and third, a method-based approach based on Social Network Analysis metrics.

4.1 Showcase 1: One-Click Download of Modeled Text Data

When considering an ecosystem for *Computational* Literary Studies, one usually thinks of applications that operate at a relatively sophisticated technical resp. computational level. But actually, it is central to the development of the discipline and its infrastructures that it remains accessible even to novices and beginners. It is also for this reason that an essential component of the DraCor prototype is a user-friendly front-end that, on the one hand, provides a set of services with an easily accessible graphical user interface (GUI) and, on the other hand, allows access to the corpora data with as little technical expertise as possible. The frontend with its graphical user interface thus also assumes didactic functions: Texts can be easily navigated through various tabs and viewed in different shapes and modes of modeling.

For example, any play from the corpora contained in DraCor can be displayed in a text view that does not differ significantly from classic ways of displaying texts in e-readers or web browsers (see Fig. 01). While texts appear in such a full-text view as conventional epistemic objects of literary studies (ready for close reading), after a tab change (see Fig. 02), one-click downloads of differently modeled derivations from these full texts can be downloaded for “distant reading” (Moretti 2013).



Dritter Auftritt

DER PRINZ.

Ihr Bild! – mag! – Ihr Bild, ist sie doch nicht selber. – Und vielleicht find' ich in dem Bilde wieder, was ich in der Person nicht mehr erblicke. – Ich will es aber nicht wiederfinden. – Der beschwerliche Maler! Ich glaube gar, sie hat ihn bestochen. – Wär' es auch! Wenn ihr ein anderes Bild, das mit andern Farben, auf einen andern Grund gemalet ist, – in meinem Herzen wieder Platz machen will: – Wahrlich, ich glaube, ich wär' es zufrieden. Als ich dort liebte, war ich immer so leicht, so fröhlich, so ausgelassen. – Nun bin ich von allem das Gegenteil. – Doch nein; nein, nein! Behäglicher, oder nicht behäglicher: ich bin so besser.

Vierter Auftritt

Der Prinz. Conti, mit den Gemälden, wovon er das eine verwandt gegen einen Stuhl lehnet.

CONTI *indem er das andere zurecht stellet.*

Ich bitte, Prinz, daß Sie die Schranken unserer Kunst erwägen wollen. Vieles von dem Anzüglichsten der Schönheit liegt ganz außer den Grenzen derselben. – Treten Sie so! –

Full text originally obtained from TextGrid Repository. TEI adaptation, corrections, enhancements by DraCor.

Direct print source: *Gotthold Ephraim Lessing: Werke. Herausgegeben von Herbert G. Göpfert in Zusammenarbeit mit Karl Eibl, Helmut Göbel, Karl S. Guthke, Gerd Hillen, Albert von Schirmding und Jörg Schönert, Band 1–8, München: Hanser, 1970 ff.*

Segments

Erster Aufzug

Erster Auftritt

Der Prinz, Der Kammerdiener

Zweiter Auftritt

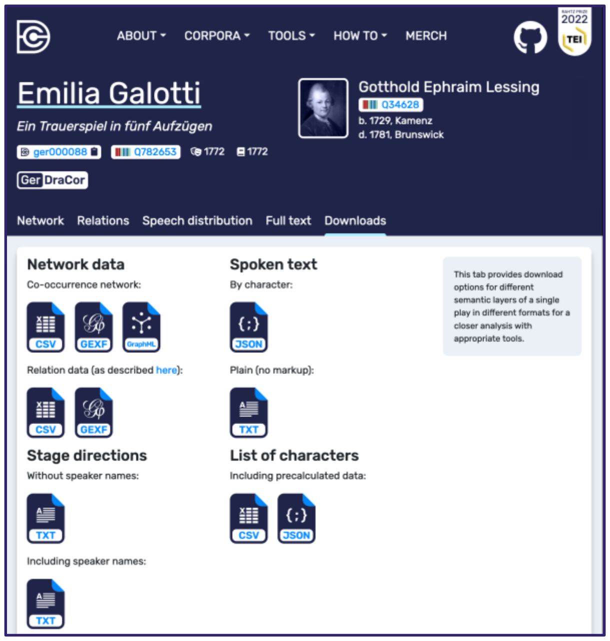
Der Prinz, Conti

Dritter Auftritt

Der Prinz

Vierter Auftritt

Fig. 01: Full text view of a DraCor play in the front-end



This tab provides download options for different semantic layers of a single play in different formats for a closer analysis with appropriate tools.

Fig. 02: Download options for a DraCor play in the front-end

This allows DraCor to introduce the different epistemic and technical manifestations of text in the CLS in an easily accessible way. Furthermore, it is also possible to work with these modeled text data immediately, which allows a quick introduction to methods and tools of the CLS.

For this showcase, we choose to use the data of a co-occurrence network (i.e. a network of characters connected via their co-presence on the stage), downloading the XML-based GEXF format that can be opened with open source programming libraries such as *networkx* (Hagberg et al. 2008) or the widely used open source desktop software *Gephi* (Bastian et al. 2009).[10] With a few clicks after the download, it is thus possible to create a network graph (see Fig. 03) that now allows the literary text to be viewed in an entirely different modeling mode, predestined for distant reading (Moretti 2011).

[10] Gephi 0.10.1, https://gephi.org.

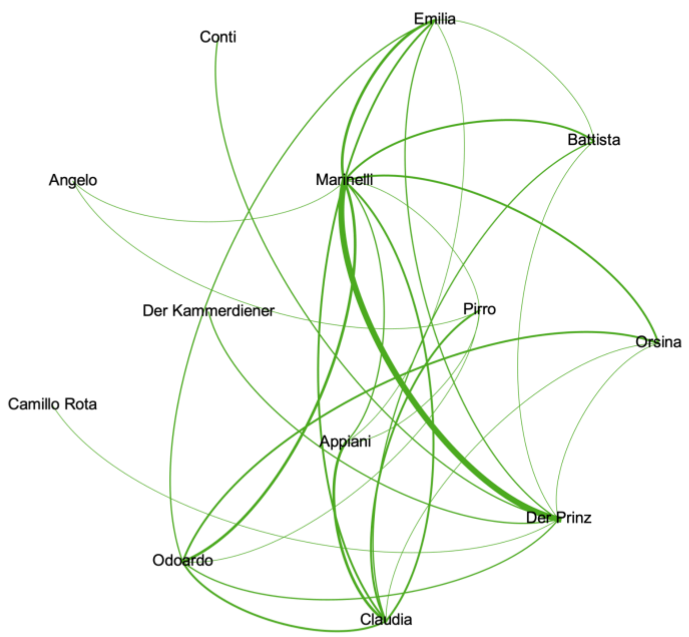


Fig. 03: Network visualization with Gephi, based on a DraCor one-click download

4.2 Showcase 2: Geo-Mapping Locations of First Performances

However, DraCor's beginner-friendly front-end is just one way to access and use the data in this Programmable Corpus. Instead, the computational literary scholar will usually access the data either directly in the form of the TEI-XML[11] or via the various APIs and API endpoints. In the following showcase, the focus is on DraCor's SPARQL[12] endpoint.

During the homogenization of metadata that theater plays undergo as part of the integration into the DraCor environment, Wikidata identifiers (entity IDs)[13] for both authors and individual works are typically included in the metadata of each play encoded in the `<teiHeader>`. For example, for the German-language bourgeois tragedy "Emilia Galloti" by Gotthold Ephraim Lessing this data is available in DraCor (Wikidata ID is highlighted). First for the author:[14]

```
<author>
    <persName>
```

[11] TEI-XML is a type of the XML format that complies the standards defined by the "Text Encoding Initiative (TEI)", cf. https://tei-c.org/
[12] Specification https://www.w3.org/TR/sparql11-query. A tutorial on how to use SPARQL to query Wikidata can be accessed at: https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial
[13] Cf. https://www.wikidata.org/wiki/Wikidata:Identifiers
[14] Cf. Code reference <1>.

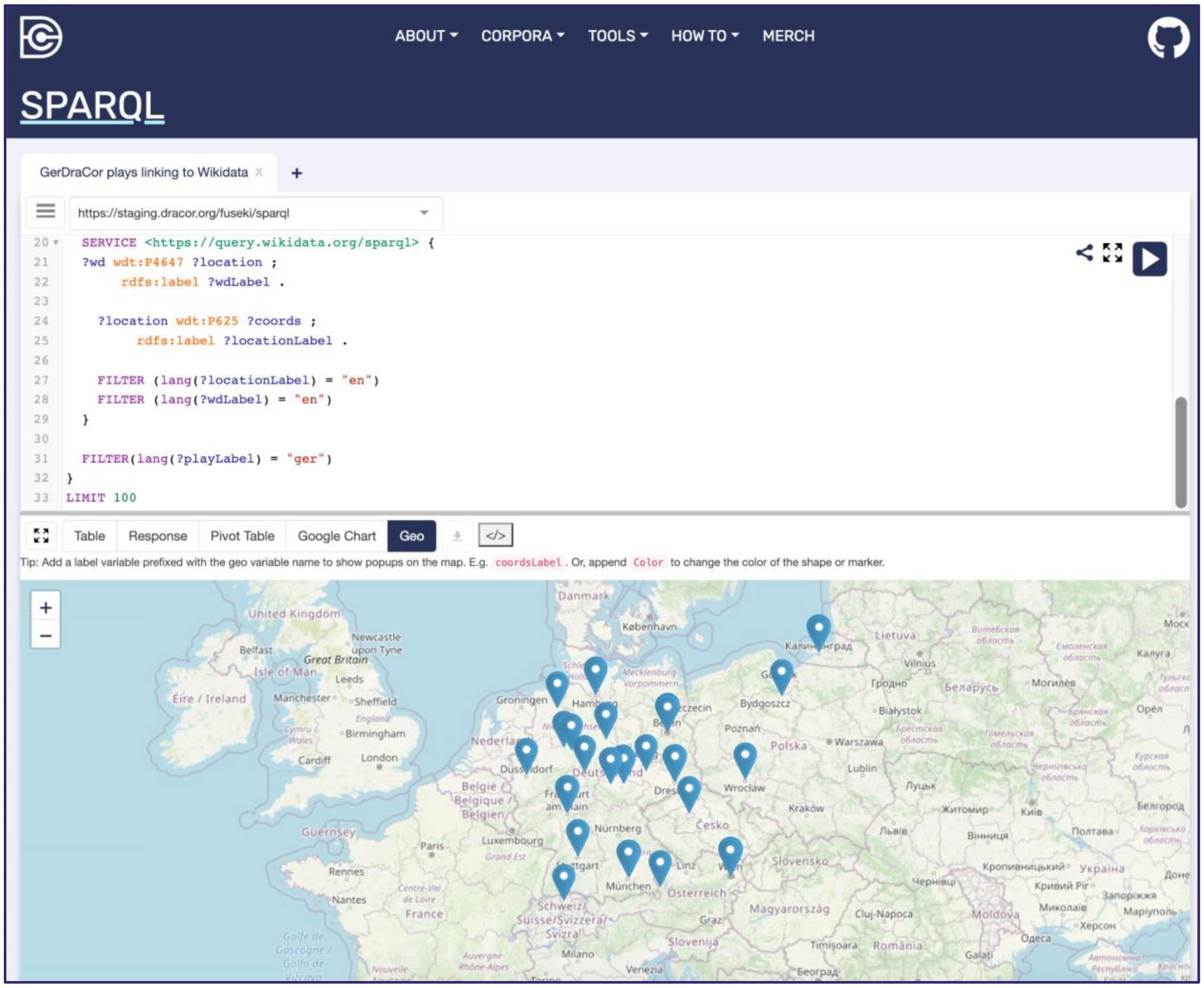
```
                <forename>Gotthold</forename>.
                <forename>Ephraim</forename>
                <surname>Lessing</surname>
            </persName>
            <idno type="wikidata">Q34628</idno>
            <idno type="pnd">118572121</idno>
        </author>
```

Then for the individual work:[15]

```
        <listRelation>
               <relation name="wikidata"
               active="https://dracor.org/entity/ger000088"
               passive="http://www.wikidata.org/entity/Q782653"/>
        </listRelation>
```

Thanks to this metadata, the plays in DraCor can be linked to further information from, among others, Wikidata, thus be embedded in the wide ecosystem of Linked Open Data and thereby benefit from the often crowd-based data enrichment projects in the World Wide Web. For example, numerous Wikidata entries on plays contain information about the "location of first performances".[16] In the case of Lessing's "Emilia Galotti", this location is the "Hagenmarkt-Theater", which also has a Wikidata entry.[17] The entry for "location of the first performance" in Wikidata has information about its "coordinate location",[18] which provides the corresponding geodata (52°16'1.9" N, 10°31'28.9" E). This embedding of DraCor plays in the Linked Open Data Cloud now makes it possible to run SPARQL queries for the entire corpora, for example. In a Jupyter Notebook,[19] we showcased a corresponding query that displays on a map all the information available in Wikidata about locations of first performances of plays in the German-language drama corpus GerDraCor. Fig. 04 shows (parts of) the query and the map.

[15] Cf. Code reference <2>.
[16] Wikidata Property "P4647", see https://www.wikidata.org/wiki/Property:P4647 for more information.
[17] Cf. https://www.wikidata.org/wiki/Q1270860.
[18] Wikidata Property "P625", see https://www.wikidata.org/wiki/Property:P625 for more information.
[19] See the section "A federated query: Connecting DraCor and Wikidata" in https://github.com/dracor-org/dracor-notebooks/blob/lod-intro/lod-intro/lod-intro.ipynb.


ABOUT
CORPORA
TOOLS
HOW TO
MERCH
SPARQL
GerDraCor plays linking to Wikidata
https://staging.dracor.org/fuseki/sparql
Table
Response
Pivot Table
Google Chart
Geo

```
  SERVICE <https://query.wikidata.org/sparql> {
  ?wd wdt:P4647 ?location ;
      rdfs:label ?wdLabel .

    ?location wdt:P625 ?coords ;
        rdfs:label ?locationLabel .

    FILTER (lang(?locationLabel) = "en")
    FILTER (lang(?wdLabel) = "en")
  }

  FILTER(lang(?playLabel) = "ger")
}
LIMIT 100
```

Tip: Add a label variable prefixed with the geo variable name to show popups on the map. E.g. `coordsLabel`. Or, append `Color` to change the color of the shape or marker.

Fig. 04: Connecting DraCor and Wikidata: Mapping Locations of First Performances of GerDraCor Plays

4.3 Showcase 3: Extracting Stage Directions for NLP

While the previous showcase uses the SPARQL endpoint of our Programmable Corpora prototype DraCor, the following showcase uses the custom developed DraCor API. Alongside the TEI encoded DraCor plays are, among others, various XQuery-based extractor functions, which make it possible, via the DraCor API, to retrieve specific and standardized text segments and use them as input for, for example, Natural Language Processing (NLP) pipelines. Thus, for instance, the TEI-based structure of the data in DraCor can be used to address specific research questions, as in our next showcase.

Again, the homogenization of the drama corpora in DraCor serves as a starting point for our showcase. During this homogenization, all plays are systematically structured in such a way that the speaker's text can be consistently differentiated from the stage directions. For this purpose, the corresponding TEI elements are used, where `<stage>` distinctly tags the text of the stage directions.[20] The following TEI snippet from Lessing's "Emilia Galotti" exemplifies the data structure.

```
<sp who="#appiani">
      <speaker>APPIANI</speaker>
      <stage>tritt tiefsinnig, mit vor sich hingeschlagnen Augen herein, und
      kömmt ihnen  näher, ohne sie zu erblicken; bis Emilia ihm entgegen
      springt.</stage>
      <p>Ah, meine Teuerste! – Ich war mir Sie in dem Vorzimmer nicht
      vermutend.</p>
</sp>
```

Via the DraCor API it is now possible to get all stage directions of a play with the corresponding request URL: https://dracor.org/api/corpora/ger/play/lessing-emilia-galotti/stage-directions. At the same time, it is possible to retrieve all the spoken texts of the plays via another endpoint. For Lessing's "Emilia Galotti", the corresponding request URL would be: https://dracor.org/api/corpora/ger/play/lessing-emilia-galotti/spoken-text.

The obtained text can now be further processed in various ways. In an early showcase, we used the NLP tool *spaCy* to perform a sentence splitting on the text data for all plays in the German-language drama corpus GerDraCor and then compared the average sentence lengths for the stage directions with those of the speaker text.

The result showed that the sentence lengths in the speaker texts were longer on average overall, but that at the same time a development can be observed leading to a successive convergence of sentence lengths (see Fig. 05) – a development that, as we have suggested (Trilcke et al. 2020), can be explained in the context of research debates about the epification of drama in the 19th century.

[20] Cf. https://www.tei-c.org/release/doc/tei-p5-doc/en/html/ref-stage.html.

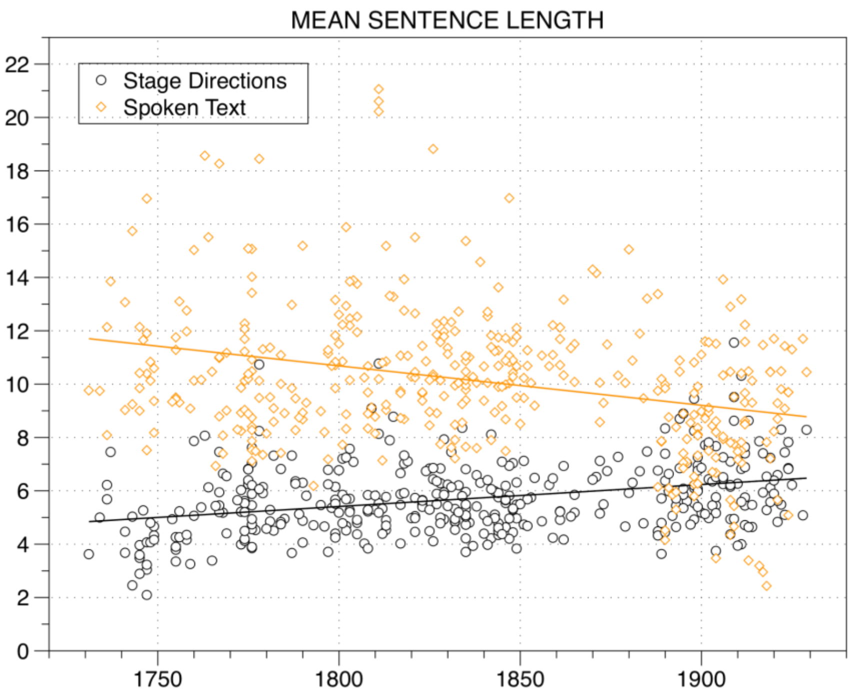


Fig. 05: Mean Sentence Length in Stage Directions and Spoken Text in GerDraCor plays visualized with Datagraph[21] *(cf. Trilcke et al. 2020)*

4.4 Showcase 4: Plotting Network Measures for Thousands of Plays

Connected to the DraCor corpora are various microservices that—following the principle of "method as a microservice"—apply specific methods of CLS to the text data in the drama corpora. The outputs from these microservices are, in the form of metrics, made available via the DraCor API. Part of these microservice-based and thus research-driven API functions rely on methods from Social Network Analysis. Again, based on the homogenized TEI structure of the plays in DraCor, in particular the semi-automated speaker identification, a dedicated microservice first automatically constructs network graphs to which then a number of algorithms from Network Analysis are applied.

The technical capability to retrieve metrics for the texts from several different drama corpora via one single API enables the use of standardized analyses for comparative literary studies, as Trilcke et al. (in press) have shown applying the concept of "Small World" to almost 3,000 dramas of European literature.

[21] https://www.visualdatatools.com/DataGraph.

Analyzing plays with reference to the "Small World" concept requires the calculation of the network metric of "Average Path Length".[22] This metric can, as outlined, be retrieved via the DraCor API. For our final showcase, we pull from the DraCor API a file of aggregate metrics, including "Average Path Length". For example, the request URL for the German-language corpus GerDraCor in this case is: https://dracor.org/api/corpora/ger/metadata/csv

After collecting the corresponding data for all DraCor corpora via the API, we filter the data based on the metadata for plays published between 1500 and 1900. In a final step, we plot the "Average Path Length" for the now remaining 2,622 plays as a chart (Fig. 06)––thus with just a few clicks taking a decisive step towards a fully-fledged "distant reading" study, whereby at the same time it becomes clear what is still missing in these data (and what would have to be provided in an elaborated study): the interpretation of the data, which has to elaborate the meaning of such plots.

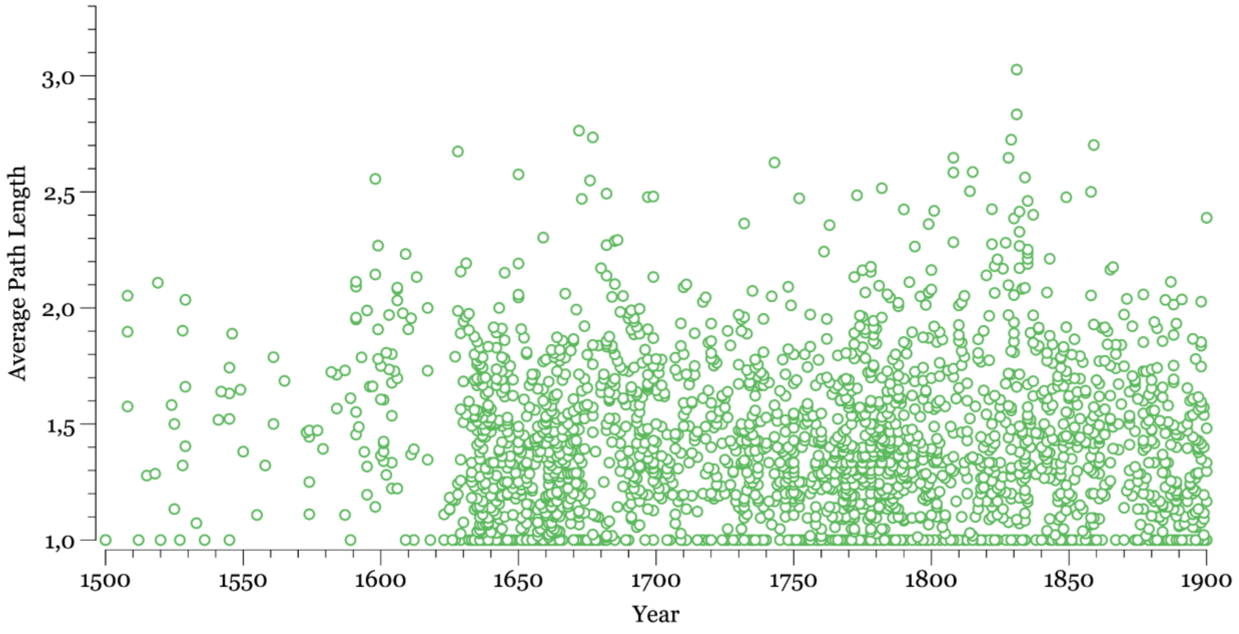


Fig. 06: Average Path Length for 2,622 plays in DraCor visualized with Datagraph

[22] See the documentation on https://networkx.org/documentation/networkx-1.3/reference/generated/networkx.average_shortest_path_length.html

5. Description of the DraCor Prototype

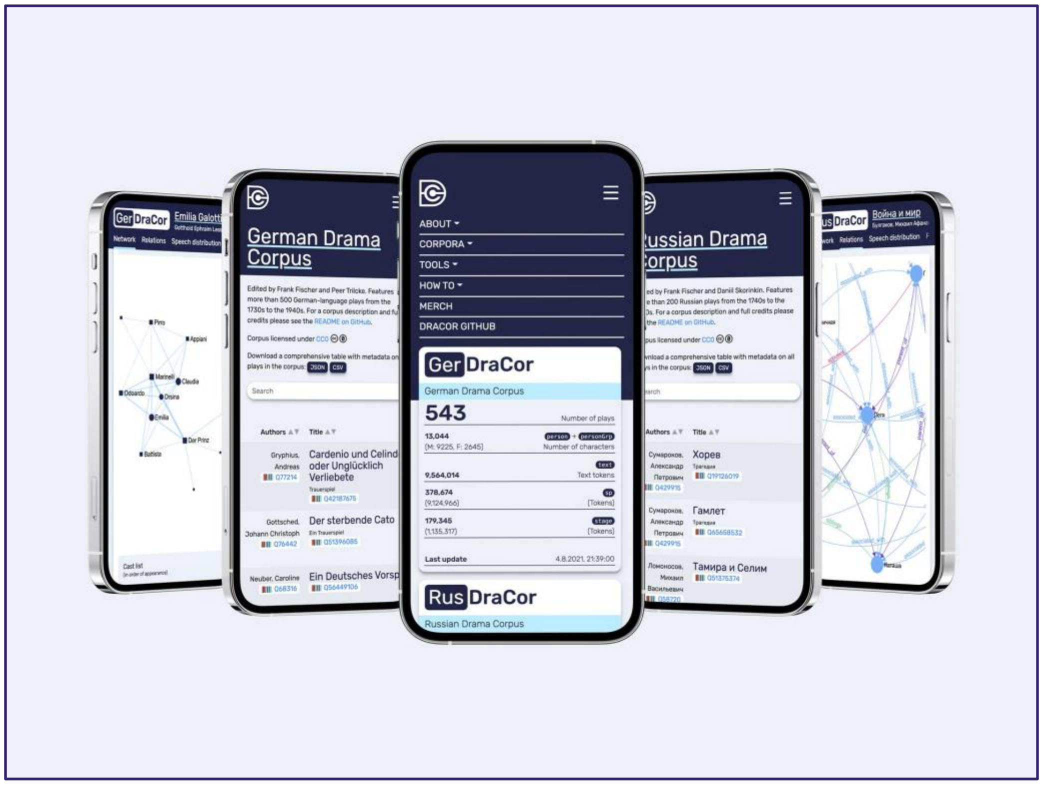


Fig. 07: Different DraCor pages as displayed on a mobile phone

In CLS INFRA’s work package 7, we exemplify the concept of programmable corpora with a prototype, which is DraCor (Drama Corpora Platform, https://dracor.org). DraCor itself follows the idea of a network-based software architecture and thus constitutes a system consisting of multiple interconnected components: At its core there are homogenized **corpora** in several (European) languages. These corpora are curated in GitHub repositories and stored in an **XML database** as a central data store which provides a RESTful **API** that powers a **front end** and can be used individually to retrieve the raw TEI-XML data, metadata, and derived data in several formats. Attached **microservices** offer additional functionalities, e.g., a **Metrics Service** is used to calculate network metrics based on the play data and a **Triple Store**, which holds representations of the plays as Linked Data and provides a **SPARQL endpoint**.

In the following, we describe in detail the individual components of the DraCor system.

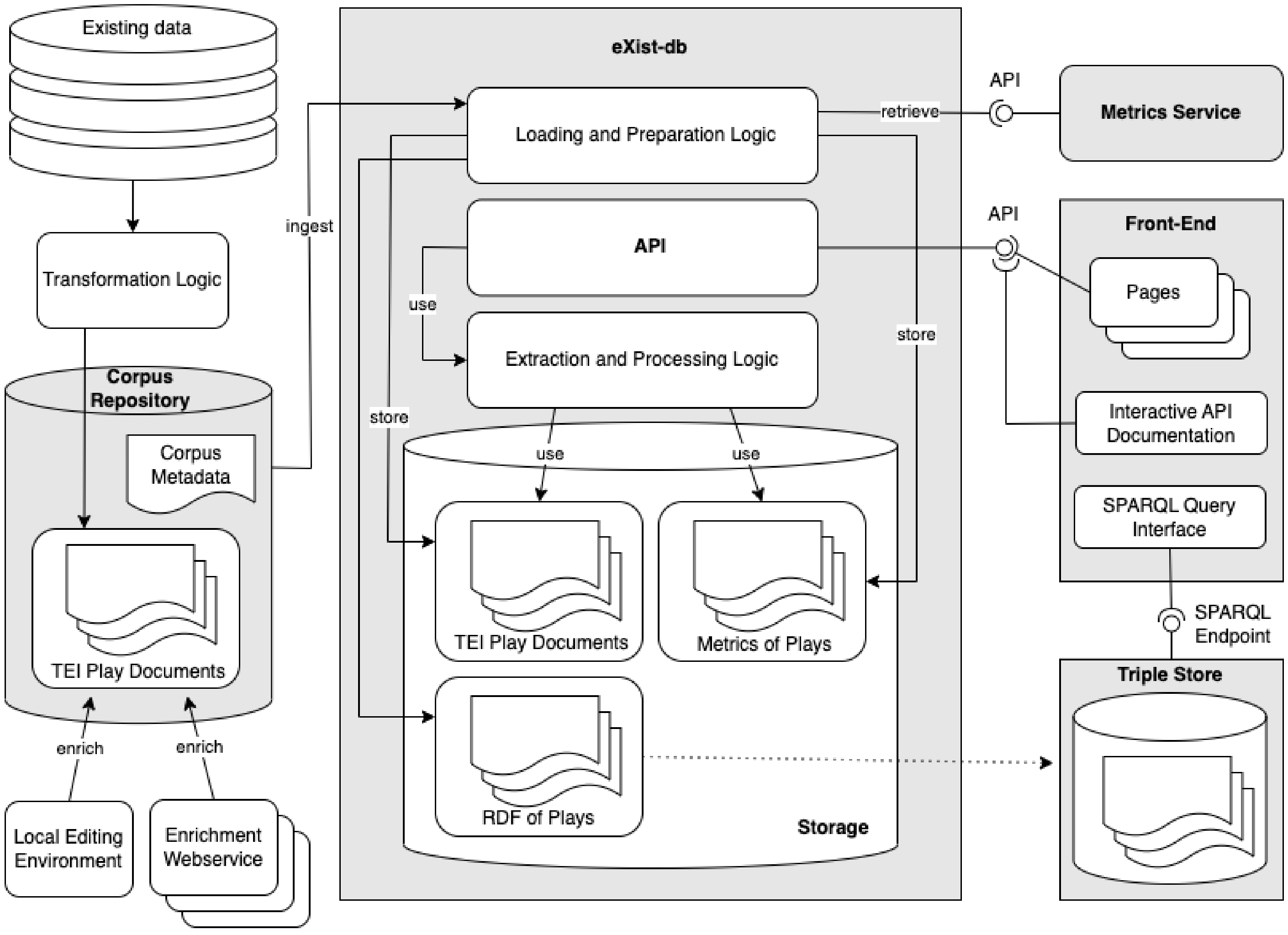


Fig. 08: Overview of the DraCor System

5.1 Corpora

As stated above, corpora as the central epistemic objects of CLS serve as the starting point of our development work. DraCor corpora are encoded following the guidelines of the Text Encoding Initiative (TEI P5)[23]. DraCor uses a TEI customization that contains only selected TEI elements supported by the wider DraCor system and restricts the use of XML attributes and its values. The

[23] https://www.tei-c.org/release/doc/tei-p5-doc/en/html/index.html Special elements for the encoding of dramatic texts are described in the module “Performance Texts” https://www.tei-c.org/release/doc/tei-p5-doc/en/html/DR.html

customization is documented by an ODD[24] from which a RelaxNG schema[25] is generated, that is used to validate the encoding of the plays. Schematron[26] rules are implemented to check for certain formal features of the XML files, e.g., the structuring into segments (acts, scenes) and the identification of individual speakers of speech acts which allow for the extraction of character networks.

Although the corpora in DraCor have different sources, they are largely homogeneous both structurally and in terms of metadata. This homogeneity makes it possible, for example, to perform comparative research on the corpora or to perform processing operations (such as extracting information or counting) in a comparable way on the corpora. However, homogeneity is something that has to be created first, because even if corpora are available in the target format TEI, they often differ in the way TEI is applied, e.g., there are several ways, that can be used to encode the structural division "scene": On could, for example use the TEI element `<div>`[27] and, optionally, classify this segment with the attribute `@type`[28]. The running number of the scene in the play could be attached to the division by using the attribute `@n` Another way to use the `@n` attribute could be to indicate the level or depth inside the text segment structure, e.g. divisions on the highest level have an `@n` value of `"1"`, divisions one level below have `"2"`, and so on.[29] Another encoding approach would be to use the numbered divisions elements[30] `<div1>` and `<div2>` for "acts" and "scenes".[31] Another way of classifying segments would be to use the attribute `@ana`[32]. These examples demonstrate that it is necessary to define a single strategy to encode the phenomena and eventually transform other means of encoding them into a single structure.

---

[24] https://github.com/dracor-org/dracor-schema/blob/main/odd/dracor.odd

[25] The RelaxNG Specification is available here: https://relaxng.org/spec-20011203.html. The schema used for DraCor is available on GitHub: https://github.com/dracor-org/dracor-schema/blob/main/odd/out/dracor.rng.

[26] See https://www.schematron.com for more information on the validation language. The schematron file that allows for checking of a play is available on GitHub: https://github.com/dracor-org/dracor-schema/blob/main/schematron/dracor.sch.

[27] https://tei-c.org/release/doc/tei-p5-doc/en/html/ref-div.html.

[28] https://tei-c.org/release/doc/tei-p5-doc/en/html/ref-att.typed.html.

[29] In the "DTA Basisformat" of the German Text Archive (DTA) the attribute @n is used this way, cf. https://www.deutschestextarchiv.de/doku/basisformat/div.html. In the "German Drama Corpus" (GerDraCor) there are two files (https://dracor.org/id/ger000474, https://dracor.org/id/ger000485) coming from the DTA that have been transformed to match the encoding of DraCor corpora.

[30] https://tei-c.org/release/doc/tei-p5-doc/en/html/DS.html#DSDIV2.

[31] The TEI files of the "Perseus Digital Library" (http://www.perseus.tufts.edu/hopper; GitHub: https://github.com/PerseusDL), that were transformed and included into the "Roman Drama Corpus" (RomDraCor) used these elements, cf. <3>.

[32] https://tei-c.org/release/doc/tei-p5-doc/en/html/ref-att.global.analytic.html.

In general, three different strategies for the homogenization of corpora can be distinguished.[33]

- In the **“from scratch” approach**, the entire datafication of literary texts is done in a multi-step workflow, which often starts at the very beginning with the digitization and raw text creation of texts. In this process, all properties of the target format can be controlled by the creators or the corpus curators. In this case, homogenization is basically not even necessary, because the uniform workflow—if it is guideline-based—guarantees homogeneity.
- The “**transformation” approach** starts from an existing corpus whose text encoding as a whole has to be adapted to the requirements of the homogeneous target format. In many cases, this can be done automatically by mapping scripts that have to be developed individually in each case. In addition, data enrichment may need to be done.
- In the “**aggregation” approach**, digital files of literary texts are taken from different resources and combined to form a new corpus. Here, the heterogeneity of the source materials is usually at a maximum. This makes it necessary, as a rule, to implement several transformation scenarios; in addition, extensive manual homogenization interventions are usually required.

Larger, community-driven corpus building initiatives usually combine these different approaches. In CLS context, the most important initiative for literary corpora is certainly the ELTeC project,[34] which is building a "European Literary Text Collection". By relying on shared Sampling Criteria,[35] elaborated Encoding Guidelines,[36] and a sophisticated Workflow Schema,[37] ELTeC has established a de facto standard in “progressive collection building” (Schöch et al. 2021).

While the “from scratch” approach hardly plays a role in building DraCor, the “transformation” and the “aggregation” approach have been extensively experimented with. From a technical point of view, these approaches comprise a series of automated and manual transformations of the source data, which depend crucially on the format and markup of the files. Texts from a single, homogeneous collection with pre-existing markup and metadata (“transformation” approach) will require different workflows and pipelines than those coming, for example, from a variety of raw text sources (“aggregation” approach).

Consequently, we have been prototyping a modular workflow made up of a set of demand-dependent components. In addition to guideline-based manual revisions (e.g., pre-structuring texts with Markdown), we use XSLT scripts for automated transformations. Edits specific to

[33] See also Börner et al. [submitted] and the concept of “Distributed Corpus Building”, discussed by Giovannini et al. [submitted].
[34] Cf. Burnard, Schöch and Odebrecht 2021.
[35] https://distantreading.github.io/sampling_proposal.html
[36] https://distantreading.github.io/encoding_proposal.html
[37] https://distantreading.github.io/workflow_proposal.html

theater plays, such as the task of speaker identification, are supported by an *Oxygen* framework;[38] we are furthermore experimenting with task-specific GUI applications based on the Javascript framework *React*.[39] The correction and enrichment of metadata, such as the addition of Wikidata IDs, is organized semi-automatically via *OpenRefine*[40].

A particular challenge is posed by living corpora (i.e., corpora that are still being worked on, hence still changing in their composition, structure, data quality, annotation depth, etc.). Here, the manual transformations performed during “onboarding” (i.e., the ingest of a corpus into the DraCor system) should be re-applicable in case of edits to the source data. Accordingly, we implemented routines for a ‘backward compatibility’ of the markup: the changes made by us during the integration in the DraCor ecosystem can later be applied again to a newer version of the source files.[41]

The literary texts in the DraCor corpora have a basic structural markup and are ready for further analysis by various methods of CLS. Furthermore, they have been additionally enriched with a research-driven markup for the application of a specific method, the network analysis of literary texts (Trilcke 2013). Therefore, the encoding is tailored to allow for the extraction of co-presence networks relying on structural segmentation of a given play into acts and scenes and having uniquely identifiable speaking and acting characters. The cast lists or dramatis personae that are contained in most dramatic texts are an insufficient source in this regard, because they tend to be incomplete. Speaker labels contained in the proper text are also often misleading, because they are often not stable enough to serve as an identifier. Therefore, the plays encoded for the DraCor plattform have an additional section in their metadata in the `<teiHeader>` that lists all characters as `<person>` elements in a `<listPerson>` and assigns them a unique identifier (`@xml:id`), that is then used in the attribute `@who` to link the individual speech acts `<sp>` with their respective speakers. The co-presence network can then be extracted by a designated algorithm.[42] While each distinct speaker represents a node in a network, a relation (edge) is established if the speeches `<sp>` of two or more speakers appear in the same segment `<div>` (normally a “scene” classified by the attribute `@type`). It is important to note that this method of

[38] https://github.com/dracor-org/dracor-oxygen-framework.
[39] Cf. e.g. our prototype of a “Who-Is-Identification”-Tool https://github.com/dracor-org/epdracor-whois resp. https://dracor-org.github.io/epdracor-whois for the interface.
[40] https://openrefine.org .
[41] For this, cf. our prototype script in the EPDraCor repository: https://github.com/dracor-org/epdracor.
[42] The algorithms used for extracting our network data are implemented in XQuery: The function of the DraCor API `metrics:get-network-metrics` <4> extracts the segments of a given TEI file using the function `dutil:get-segments` <5> and for each of these segments gets the distinct speakers with the function `dutil:distinct-speakers` <6>. The network metrics are calculated based on these extracted features with the “DraCor Metrics Service” (see section 5.4) using the Python package “networkx”.

extracting networks from dramatic texts is only one possibility among many others and that the scope for interpretation is determined by this mechanism of extraction.

While the TEI data of the theater plays contained in the corpora have some research driven features, in general, they are suitable for the field of digital drama analysis, which manifests itself in a growing list of research done based on DraCor corpora[43] and is especially facilitated by also hosting the corpora on GitHub[44] with an open license that allows for extensive re-use.

5.2 DraCor Data Storage

5.2.1 GitHub Repositories

GitHub is a key infrastructural component, because it serves not only for storing the corpora, but also for the development of the other components of the system, because it offers a rich set of tools, e.g., versioning[45] and issue tracking, as well as a space for discussions and collaboration.

Both application code and corpora are hosted on GitHub in designated repositories within the DraCor "organization" (https://github.com/dracor-org). Each corpus repository includes at least a TEI-XML file "corpus.xml" that contains the metadata of the corpus and a folder "tei" which comprises the TEI-XML files of the individual plays.

Optionally, but highly recommended, a corpus repository should contain a "README.md" file in Markdown that describes the corpus.

5.2.2 eXist-db

A key component of the DraCor system is the XML-Database eXist-db[46]. It is used not only for storing the XML data of the plays, but also providing the DraCor API[47], which is implemented in

[43] See https://dracor.org/doc/research.
[44] https://github.com/dracor-org
[45] Each revision campaign of a single play is uniquely identifiable by its commit ID, which is an asset in a system that is working with "living corpora". By relying on GitHub and using the commit IDs as identifiers, it is possible to assemble (and re-assemble) corpora representing a given state in the development. See the example in the "Docker" notebook on how to create a stable corpus in a local DraCor instance: https://github.com/dracor-org/dracor-notebooks/blob/docker/docker/local-dracor-with-docker.ipynb
[46] Cf. http://exist-db.org. The version running on the production server at the time of writing of this report is 6.0.1.
[47] GitHub repository of the DraCor API: https://github.com/dracor-org/dracor-api

XQuery[48] with RESTXQ[49] and runs as an application within the eXist-db instance. The application features a modular architecture[50].

Corpora are stored in the database as collections in `/db/data/dracor` which contains five subcollections (“metrics”, “rdf”, “sitelinks”, “tei” and “webhook”) representing different aspects of a play or providing additional functionality. For each corpus added to the database in each of these collections a sub-collection for the corpus is created. For example, when adding the German Drama Corpus (“GerDraCor”) with the corpusname “ger” to the database the source TEI-XML data is stored in `/db/data/dracor/tei/ger`, the derived network metrics are stored in `/db/data/dracor/metrics/ger`, etc.

[48] Cf. https://www.w3.org/TR/xquery

[49] Specification of RESTXQ: http://exquery.github.io/exquery/exquery-restxq-specification/restxq-1.0-specification.html.

[50] Code of the modules are available in the latest version here https://github.com/dracor-org/dracor-api/tree/main/modules. Code reference <7> represents the state of the folder in version 0.88.0, which overall is closest to the versions reviewed in this report. : The module “api.xqm” contains the xQuery code of the core API functionality and the definition of the endpoints, “config.xqm” contains settings, e.g. the paths of data collections and others (some of these variables can be overwritten with environment variables in a Docker setting). The functionality to load corpora as ZIP files from GitHub is contained in “load.xqm”. The module “metrics.xqm” contains the code that allows for the calculation of play metrics (e.g., number of speeches and stage directions). It also has a function that extracts the structural information of a play and its speakers and, based on this, retrieves calculated network metrics by sending a POST request to the metrics service. The code to create a RDF representation of the play is contained in the module “rdf.xqm”. This module also handles the connection to the Triple Store. The module “trigger.xqm” defines functions which are executed when a document is added to, deleted from, or updated in the database. These functions update calculated metrics and RDF representations. The module “util.xqm” contains utility functions, which actually offer the core functionality of this service. The module’s functions are called in the API module, whereas the API module in general handles the creation of the HTTP response, the functions that retrieve the information from the TEI files (extractor functions), are contained in the “util.xqm” module. The module “webhook.xqm” implements a webhook (https://docs.github.com/en/developers/webhooks-and-events/webhooks/about-webhooks) that can be used to automatically update the database when changes are made to the data repositories of individual corpora. The module “wikidata.xqm” provides functionality to retrieve information from Wikidata, e.g., retrieves additional information about an author by sending a SPARQL query to Wikidata’s SPARQL endpoint. Additional endpoints to implement a document-driven API following the specification of the “Distributed Text Services” (DTS) API are available in the module “dts.xqm” (on DTS cf. section 6).

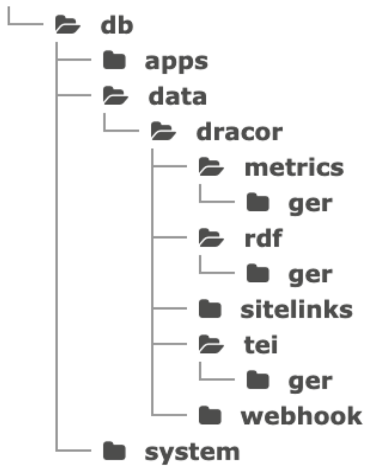


Fig. 09: View of collections in the eXist-db IDE “eXide”[51] after loading GerDracor into a local DraCor instance

A webhook that triggers an update of corpora when changes are pushed to the main branch is configured for the staging server https://staging.dracor.org. For the production server https://dracor.org the ingest of corpora is triggered manually by the administrators, to make sure, only reviewed changes to the corpora end up in the database.

The eXist-db provides a generic REST API[52] which—in a local setup, i.e., by using Docker—can be used to directly access and manipulate resources in the database by appending the collection path of a resource to the base-url `{url of the eXist-db}/rest/`. For example, with a running local eXist-db at localhost, the standard port 8080 and GerDraCor loaded, the source TEI files of the plays in the corpus can be listed with a GET request to http://localhost:8080/exist/rest/db/data/dracor/tei/ger. The stored network metrics of a single play, e.g. “Emilia Galotti”, are available at http://localhost:8080/exist/rest/db/data/dracor/metrics/ger/lessing-emilia-galotti.xml.

5.3 DraCor API[53]

Although the eXist-db already offers a generic REST API to retrieve (and manipulate) stored documents, more specialized functionality that is motivated by uses in research and/or by affordances of the frontend is implemented as an API – hence called “DraCor API”. While the generic eXist-db REST API is designed to somewhat ‘mirror’ the organization of resources as

[51] GitHub Repository: https://github.com/eXist-db/eXide.
[52] See documentation: https://exist-db.org/exist/apps/doc/devguide_rest.
[53] The version of the DraCor API running on the production server at the time of writing of this report is 0.87.1, cf. the corresponding release https://github.com/dracor-org/dracor-api/releases/tag/v0.87.1. The code references partly also refer to the following version https://github.com/dracor-org/dracor-api/releases/tag/v0.88.0.

collections in the database, the DraCor API was designed with a domain model of digital drama analysis in mind and thus organizes the API functionality around the two core entities "corpus" and theater "play", thus following the assumption already mentioned above that corpora are the central epistemic objects of CLS (cf. Gavin 2023: 4). Information can be requested or methods can be invoked on the level of the whole corpus, as well as for a single play, but a play is always considered part of a corpus and thus in general needs to be identified by the name of the corpus ("corpusname") and the name (slug) of a play ("playname") resulting in a typical repeating naming pattern for an endpoint URL, like `{base-url}/api/corpora/{corpusname}/play/{playname}`.

5.3.1 Implementation

The module "api.xqm"[54] contains the code of the DraCor API and defines its endpoints making use of RESTXQ, whereas the URL of the endpoint is defined in the XQuery function annotation

`%rest:path("/corpora/{$corpusname}/play/{$playname}/tei")`

The XQuery code below implements the endpoint `/corpora/{corpusname}/play/{playname}/tei` and can serve as an example of how in general the API endpoints of the DraCor API are written:

```
(:~
: Get TEI representation of a single play
:
: @param $corpusname Corpus name
: @param $playname Play name
: @result TEI document
:)
declare
  %rest:GET
  %rest:path("/corpora/{$corpusname}/play/{$playname}/tei")
  %rest:produces("application/xml", "text/xml")
  %output:media-type("application/xml")
function api:play-tei($corpusname, $playname) {
  let $doc := dutil:get-doc($corpusname, $playname)
  return
    if (not($doc)) then
      <rest:response>
      <http:response status="404"/>
    </rest:response>
  else
    let $tei := $doc//tei:TEI
    let $model-pi := $doc/processing-instruction(xml-model)
```

[54] Cf. code reference <8>.

```
            return if ($model-pi) then
              document {
                processing-instruction {'xml-model'} {$model-pi/string()},
                $tei
              }
            else $tei
        };
```

The first part between `(:~` and `:)` of the code is a multiline comment that documents the function. The actual function starts with the keyword `declare` followed by the function annotations that define the endpoint:

- `%rest` specifies the HTTP method (GET),
- the endpoint URL is set with `%rest:path`.
- The response format is specified in the annotation `%rest:produces`.

The function will return XML data. The actual name of the function "api:play-tei" follows the `function` keyword. The function annotation `%rest:path` controls that the path parameters of `$corpusname` and `$playname` are extracted from the request URL of the API call and passed to the function. The code line starting with the keyword `let` defines a variable `$doc` to which the contents of an XML document are assigned with the help of the utility function `util:get-doc`. A control structure starting with the keyword `if` checks if the document is available and, in case it is not, returns the HTTP status code `404`, which commonly is used, if a resource is not available.[55]

If the XML is available and successfully retrieved by the `get-doc` function, the code in the else block is executed. If the original document contains an xml-model processing instruction, it is issued, and, eventually, the TEI document is returned. Other endpoints are implemented accordingly. As a rule of thumb there is a function that defines the endpoint (path, accepted/returned formats ...). This function retrieves data from one or more utility functions contained in the util module and returns the data as the response of the API endpoint.

The formats returned by the API are JSON, CSV, RDF+XML, TEI+XML, XML (Gephi, GraphML) and PLAINTEXT. In some cases, e.g., in the case the endpoints are to retrieve network data, the format is part of the endpoint's URL, but some of the endpoints rely on the HTML "Accept" header field to trigger different behavior, e.g., the endpoints `/id/{id}` and `/corpora/{corpusname}/play/{playname}/spoken-text-by-character` evaluate the "Accept" header to decide which format to return. In the XQuery code this is implemented by providing functions with different function annotations in the RESTXQ namespace, e.g.,

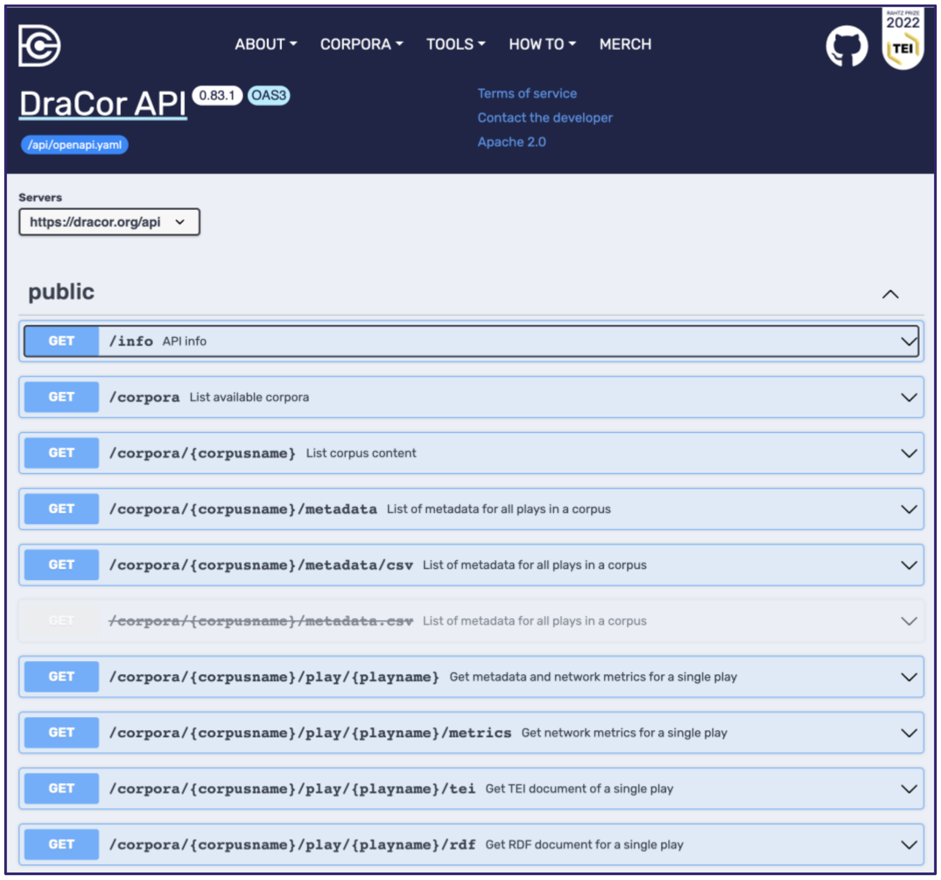
[55] Cf. https://www.rfc-editor.org/rfc/rfc9110.html#name-404-not-found.

`%rest:produces("text/csv")` vs. `%rest:produces("application/json")`. The values of the annotations are Internet Media Types (MIME types). Not for all formats returned by the API there are MIME types available: Whereas there is a designated MIME type for TEI already declared, which is `application/tei+xml`[56], there are no designated MIME types registered for the network data formats GraphML and GEXF. Therefore, relying solely on content negotiation via MIME types in the Accept Header field of the HTTP request for all endpoints did not seem feasible[57], which resulted in multiple endpoints returning the same information, but in different serializations (see for example section 5.3.3.4 on network data).

5.3.2 OpenAPI Documentation

The DraCor API is documented in the "OpenAPI Specification Format"[58]. It is a standardized format for describing an API, including the available endpoints and the supported HTTP methods (GET, POST, …), the expected input and output for each endpoint, which can be described with schemas, and many other details. The OpenAPI specification is language agnostic, meaning it can be used to describe APIs written in any programming language. APIs described with OpenAPI are interoperable, because the documentation is machine-readable and can be easily understood and consumed by a wide range of tools, such as API development frameworks or documentation generators. It is also possible to automatically generate client libraries for many programming languages from an OpenAPI specification.[59] The OpenAPI specification serves as a contract between the API provider and consumers, clearly defining what the API does and does not provide (for another use-case of OpenAPI see section 6.1.2).

[56] See https://www.rfc-editor.org/rfc/rfc6129.html; still, the API uses "application/xml".
[57] An option would be to rely on an additional parameter "format", e.g. "Accept application/xml;format=graphml" vs. "Accept application/xml;format=gexf", see https://www.rfc-editor.org/rfc/rfc9110.html#name-content-negotiation-fields. The use of MIME types is generally encouraged by guidelines on API design, cf. e.g., the REST API Design Rulebook: "Rule: Media type negotiation should be supported when multiple representations are available" (Massé 2012: 43).
[58]On OpenAPI see https://swagger.io/specification and the section 5.3.2 in this report. The API specification of the DraCor API can be found in the "api.yml" file: https://github.com/dracor-org/dracor-api/blob/main/api.yaml
[59] https://swagger.io/tools/swagger-codegen.



DraCor API 0.83.1 OAS3

/api/openapi.yaml

Terms of service
Contact the developer
Apache 2.0

Servers
https://dracor.org/api

public

GET `/info` API info

GET `/corpora` List available corpora

GET `/corpora/{corpusname}` List corpus content

GET `/corpora/{corpusname}/metadata` List of metadata for all plays in a corpus

GET `/corpora/{corpusname}/metadata/csv` List of metadata for all plays in a corpus

GET ~~`/corpora/{corpusname}/metadata.csv`~~ List of metadata for all plays in a corpus

GET `/corpora/{corpusname}/play/{playname}` Get metadata and network metrics for a single play

GET `/corpora/{corpusname}/play/{playname}/metrics` Get network metrics for a single play

GET `/corpora/{corpusname}/play/{playname}/tei` Get TEI document of a single play

GET `/corpora/{corpusname}/play/{playname}/rdf` Get RDF document for a single play

Fig. 10: DraCor API OpenAPI specification visualized with SwaggerUI

The specification of the DraCor API is rendered using Swagger UI[60] as an interactive documentation[61]. The API can be also used right away from within the browser, e.g., opening the URL https://dracor.org/api/corpora will return a list of the available corpora.

5.3.3 Functionality and Endpoints

On a general level, the endpoints of the DraCor API can be differentiated into two groups: *admin endpoints* and *content endpoints*.

There are a few endpoints that have a central function in the administration of the system. They allow for adding a corpus to the database, triggering the process to load data from a repository (on GitHub) or delete a corpus from the database. By following the REST principles, these endpoints use the standard HTTP methods POST and DELETE on the corpora endpoint `/corpora/{corpusname}`, and, in case of deleting a single play, the HTTP method DELETE

60 https://swagger.io/tools/swagger-ui
61 https://dracor.org/doc/api. An introduction on how to use the interactive documentation is available here: https://github.com/dracor-org/dracor-notebooks/blob/textplus/api-tutorial/textplus-api-tutorial.ipynb.

on the `/corpora/{corpusname}/play/{playname}` endpoint. Adding a single play to a corpus is considered an update operation on the `/corpora/{corpusname}/play/{playname}` endpoint and therefore uses the PUT method.[62] All "admin" operations require that a user has authenticated herself or himself and has the necessary write permissions on the database. Corpora are loaded and updated on the production system with these endpoints. While these functionalities are not relevant for the common user working with the ready-to-use infrastructure provided at https://dracor.org, they gain importance in scenarios in which a local installation of the system is set up, e.g., by using Docker. The admin endpoints are the easiest way to create and populate custom corpora.[63]

The larger number of public ("content") endpoints are motivated by the needs of research and the front-end (see section 5.6). They are primarily used by issuing HTTP GET requests and as a response return data in various formats (JSON, CSV, XML, TEI+XML, RDF+XML, PLAINTEXT). They do not require a user to be logged in, therefore the endpoints are grouped under the label "public" in the OpenAPI specification.[64]

Of these "public" endpoints two endpoints—i.e., `/info` and `/id/{id}`—provide utility functionality. The `/info` endpoint supports some self-description capabilities of the API by returning information about the underlying versions of the eXist-db and the API.[65] In the returned JSON object, the field with the key "version" contains the version number of the application, which refers to a release that can be tracked down on GitHub[66], thus allowing a user to get information about the specific development state of the system. When using the API in research, it is advisable to note the used version of the API to allow for a reproduction of the results based on the actual infrastructural components used.[67]

The "resolver endpoint" `/id/{id}`[68] which can resolve DraCor IDs, e.g., "ger000088" to corpus name "ger" and the play name "lessing-emilia-galotti" issues a redirect depending on the

[62] The interactive documentation (using Swagger UI, see section below) can be accessed at https://dracor.org/doc/api#/admin.
[63] For a tutorial see https://github.com/dracor-org/dracor-notebooks/blob/docker/docker/local-dracor-with-docker.ipynb.
[64] The interactive documentation of the "public" endpoints can be accessed at https://dracor.org/doc/api#/public.
[65] The interactive documentation can be accessed at https://dracor.org/doc/api#/public/api-info. The endpoint is implemented by the function `api:info` <10> in "api.xqm".
[66] see https://github.com/dracor-org/dracor-api/releases for all releases of the API.
[67] In deliverable D 7.3, we will showcase a more advanced approach towards repeating research by using Docker technology.
[68] The interactive documentation of the endpoint can be accessed at https://dracor.org/doc/api#/public/resolve-id. The endpoint is implemented by the function `api:id-to-url` <11> which issues a redirect to an url generated by the local function `local:id-to-url` <12> based on the provided "Accept" header field.

provided "Accept" header field[69] in the request. For example, if a client requests the resource "https://dracor.org/api/id/ger000088" without any specific "Accept" header, the server sends a 303 "See other"[70] status code and provides "https://dracor.org/ger/lessing-emilia-galotti" as the new location in the "Location" response header[71]. If the client is a web browser, it will normally display the webpage of the single play. If the client requests RDF data by including `application/rdf+xml` in the "Accept" field of the request header, the server will also send the status code "303" but redirect the client to the `/corpora/{corpusname}/play/{playname}/rdf` endpoint, which returns metadata about the play in RDF.[72] Apart from RDF the other MIME type that is supported by this content negotiation mechanism is `application/json` which forwards to the `/corpora/{corpusname}/play/{playname}` endpoint. Most prominently, the resolver endpoint is used to properly redirect to a single play page if the (shorter) URL includes only the DraCor ID, e.g., "https://dracor.org/id/ger000088". This allows for the shortest possible citation of a play on the DraCor platform by just providing the ID.[73]

The other endpoints grouped under "public" in the documentation are providing information on

(1) a single-language corpus (see section 5.3.3.1),
(2) a single play in a corpus (see section 5.3.3.2),
(3) data on the key constituents of a play, i.e. (see section 5.3.3.3),
    (3a) segments,
    (3b) characters,
    (3c) the spoken text and
    (3d) stage directions, as well as
(4) derived network data on a single play, its characters, and their relations (see section 5.3.3.4).

These aspects are seldom fully covered by a single endpoint because the information on a single entity is available in different serializations or structured with a different focus or probable use case in mind.

[69] Cf. https://www.rfc-editor.org/rfc/rfc9110.html#section-12.5.1.
[70] Cf. https://www.rfc-editor.org/rfc/rfc7231#section-6.4.4.
[71] Cf. https://www.rfc-editor.org/rfc/rfc7231#section-7.1.2.
[72] This behavior conforms to what has been described as "Cool URIs for the Semantic Web": https://www.w3.org/TR/cooluris.
[73] Cf. for example the way Trilcke et al. (in press) are referencing DraCor plays.

5.3.3.1 Information on a Corpus

Tab. 01 contains an overview of the available features for a single corpus. The column "Feature" contains a unique name for the feature. In the endpoint-columns the keys of the respective JSON objects are given. An asterisk "*" indicates that the object is part of an array, a dot "." is used to notate the paths of nested objects. When referring to a corpus feature in the text the siglum "C" followed by the row number of the table in square brackets is used, e.g. [C1] for the feature "corpus_name".

		endpoint	
No.	**Feature**	**/corpora**	**/corpora/{corpusname}**
C1	corpus_name	*.name	name
C2	corpus_uri	*.uri	N.A.
C3	corpus_title	*.title	title
C4	corpus_acronym	*.acronym	acronym
C5	corpus_description	*.description	description
C6	corpus_repository	*.repository	repository
C7	corpus_licence	*.licence	licence
C8	corpus_licence_url	*.licenceUrl	licenceUrl
C9	corpus_num_of_plays	*.metrics.plays	N.A.
C10	corpus_num_of_characters	*.metrics.characters	N.A.
C11	corpus_num_of_characters_male	*.metrics.male	N.A.
C12	corpus_num_of_characters_female	*.metrics.female	N.A.
C13	corpus_num_of_tei_text_elements	*.metrics.text	N.A.
C14	corpus_num_of_sp	*.metrics.sp	N.A.
C15	corpus_num_of_stage	*.metrics.stage	N.A.
C16	corpus_num_of_word_tokens_in_text_elements	*metrics.wordcount.text	N.A.
C17	corpus_num_of_word_tokens_in_sp	*.metrics.wordcount.sp	N.A.
C18	corpus_num_of_word_tokens_in_stage	*.metrics.wordcount.stage	N.A.
C19	corpus_metrics_date_updated	*.metrics.updated	N.A.
C20	corpus_play_objects	N.A.	dramas

Tab. 01: Corpus Features

Information on available corpora (and, consequently, on a single corpus[74]) can be retrieved from the `/corpora` endpoint[75] which returns an array of objects representing the corpora with some metadata: The field with the key "name" [C1] of the corpus contains an identifier ("corpusname") that is needed when requesting information about a single corpus. The other fields contain the title of the corpus (key "title", [C3]), a "description" [C5], URL of the repository on GitHub [C6], information on the license (key "licence" [!] [C7]) and a URL at which the license can be found [C8]. The field with the key "uri" [C2] contains an identifier that corresponds to the request URL of the `/corpora/{corpusname}` endpoint and can be used to request more information about the corpus, e.g., included plays. The data returned is extracted from the "corpus.xml" file that describes each corpus.[76]

If the optional query parameter "include" is provided and set to "metrics" in the request, additional count-based metrics on the corpora can be included in the response, i.e. number of plays [C9], texts [C13][77], speech acts [C14], stage directions [C15], characters [C10] (male [C11] and female [C12]) and number of word tokens in speeches [C17], stage directions [C18], and texts [C16].[78]

The contents, meaning the included plays, of a single corpus can be retrieved from the endpoint `/corpora/{corpusname}`[79]. This endpoint also offers basic metadata about the

[74] The most comprehensive data on a single corpus is contained by the list of all corpora as returned by the `/corpora` endpoint. Although the responses of the endpoints `/coropora/{corpusname}` contains some metadata on the corpus that is extracted from the "corpus.xml", the endpoint `/corpora/{corpusname}` (and, respectively, `/corpora/{corpusname}/metadata` and `/corpora/{corpusname}/metadata/csv` provide data on the included plays). The above-mentioned metrics about a single corpus are only contained in the list of corpora returned by the `/corpora` endpoint. It is fair to say that the naming of the endpoints is somewhat misleading.

[75] The interactive documentation can be accessed at https://dracor.org/doc/api#/public/list-corpora.

[76] The function `dutil:get-corpus` <13> retrieves the data from the "corpus.xml" file stored in the "tei" collection of the given corpus in the database. The information is extracted by the function `dutil:get-corpus-info` <14>.

[77] Whereas the field "plays" contains the number of `<tei:TEI>` elements <15>, which represents the number of documents in a corpus, "texts" is the count of all elements `<tei:text>` inside the documents <16>.

[78] The metrics are retrieved by the function `local:get-corpus-metrics` <17> in "api.xqm". The function is called from the function `api:corpora` if the parameter "include" is set to "metrics" <18>. The function counts elements in the collection of TEI documents, except for the word counts <19>, which are retrieved from the pre-calculated files in the "metrics" collection and summed up. This is done to avoid having to tokenize all texts at this point to count the tokens . When requesting the metadata on a single corpus by calling the endpoint `/corpora/{corpusname}/metadata` the metrics are provided by the function `dutil:get-corpus-meta-data` (cf. <20>) <21> which does not include the number of "plays" (elements `<tei:TEI>`) and "texts" (elements `<tei:text>`), but an array of the actual plays with metadata.

[79] The interactive documentation can be accessed at https://dracor.org/doc/api#/public/list-corpus-content.

corpus that should allow for unambiguously identifying it, e.g., its identifier (field “name”). The metadata on the corpus is, again, fetched from “corpus.xml”, as in the case of the `/corpora` endpoint. In addition, there is a field with the key “dramas” [C20] which includes an array of objects representing the individual plays.

5.3.3.2 Information on a Play

The information on a single play is distributed over several endpoints, which can be seen in the Tab 02 that gives an overview over the 64 features available.[80] The column “Feature” contains a unique name for the feature, in the endpoint-columns the key of the respective JSON objects (or in the case of a CSV file, the column name) is given. An asterisk “*” indicates that the object is part of an array, a dot “.” is used to notate the paths of nested objects. When referring to a play feature in the text the siglum “P” followed by the row number of the table in square brackets is used, e.g. [P2] for the feature “play_id”.

The features can be grouped as follows:

- Identifiers of the play (and the corpus it is contained in), external IDs (Wikidata) [P1–4]
- Titles, such as main title and subtitle, if available, and English translation thereof [P5–8]
- Information on the author(s), names in several variants (fullname, shortname), external identifier(s) of the author (primarily Wikidata) [P9–21]
- Genre [P22, 23]
- Dates [P24–27]
- Sources digital and print [P28–34]
- Wikipedia links [P35]
- Segments, such as acts, scenes, paragraphs, lines; but also structures of dramatic texts, speeches, stage directions; number of word tokens therein [P36–43]
- Characters, including social relations; information on culmination of characters in a segment [P44-52]
- Network data, metrics based on a co-presence network constructed from appearances of speaking characters in a segment [P53–P64]

		endpoint				
No.	**Feature**	**/corpora/{corpusname}**	**/corpora/{corpusname}/metadata**	**/corpora/{corpusname}/metadata/csv**	**/corpora/{corpusname}/play/{playname}**	**/corpora/{corpusname}/play/{playname}/**

[80] The endpoint `/corpora/{corpusname}/metadata` includes the name of the corpus in two fields, “playName” is considered a duplicate of the field with the key “name” and is not included in the table.

						metrics
P1	play_corpus_name	N.A.	N.A.	N.A.	corpus	corpus
P2	play_id	dramas.*.id	*.id	id	id	id
P3	play_name	dramas.*.name	*.name	name	name	name
P4	play_wikidata_id	dramas.*.wikidataId	N.A.	N.A.	wikidataId	N.A.
P5	play_title	dramas.*.title	*.title	title	title	N.A.
P6	play_subtitle	dramas.*.subtitle	*.subtitle	subtitle	subtitle	N.A.
P7	play_title_en	dramas.*.titleEn	N.A.	N.A.	titleEn	N.A.
P8	play_subtitle_en	dramas.*.subtitleEn	N.A.	N.A.	subtitleEn	N.A.
P9	play_author_name	dramas.*.authors.*.name	N.A.	N.A.	authors.*.name	N.A.
P10	play_author_name_en	N.A.	N.A.	N.A.	authors.*.nameEn	N.A.
P11	play_author_fullname	dramas.*.authors.*.fullname	N.A.	N.A.	authors.*.fullname	N.A.
P12	play_author_fullname_en	N.A.	N.A.	N.A.	authors.*.fullname En	N.A.
P13	play_author_shortname	dramas.*.authors.*.shortnam e	N.A.	N.A.	authors.*.shortnam e	N.A.
P14	play_author_shortname_en	N.A.	N.A.	N.A.	authors.*.shortnam eEn	N.A.
P15	play_first_author_name	dramas.*.author.name	N.A.	N.A.	author.name	N.A.
P16	play_first_author_shortname	N.A.	*.firstAuthor	firstAuthor	N.A.	N.A.
P17	play_first_author_deprecatio n_warning	N.A.	N.A.	N.A.	author.warning	N.A.
P18	play_author_also_known_as	N.A.	N.A.	N.A.	authors.*.alsoKnow nAs	N.A.
P19	play_author_ref_external_id	dramas.*.authors.*.refs.*.ref	N.A.	N.A.	authors.*.refs.*.ref	N.A.
P20	play_author_ref_type	dramas.*.authors.*.refs.*.typ e	N.A.	N.A.	authors.*.refs.*.typ e	N.A.
P21	play_num_of_co_authors	N.A.	*.numOfCoAut hors	numOfCoAuthor s	N.A.	N.A.
P22	play_genre_normalized	N.A.	*.normalizedG enre	normalizedGenre	genre	N.A.
P23	play_is_libretto	N.A.	*.libretto	libretto	libretto	N.A.
P24	play_year_written	dramas.*.writtenYear	*.yearWritten	yearWritten	yearWritten	N.A.
P25	play_year_printed	dramas.*.printYear	*.yearPrinted	yearPrinted	yearPrinted	N.A.
P26	play_year_premiered	dramas.*.premiereYear	*.yearPremiere d	yearPremiered	yearPremiered	N.A.
P27	play_year_normalized	dramas.*.yearNormalized	*.yearNormaliz	yearNormalized	yearNormalized	N.A.

			ed			
P28	play_digital_source_name	dramas.*.source	N.A.	N.A.	source.name	N.A.
P29	play_digital_source_url	dramas.*.sourceUrl	*.digitalSource	digitalSource	source.url	N.A.
P30	play_original_source_full_citation	N.A.	N.A.	N.A.	originalSource	N.A.
P31	play_original_source_publisher	N.A.	*.originalSourcePublisher	originalSourcePublisher	N.A.	N.A.
P32	play_original_source_publication_place	N.A.	*.originalSourcePubPlace	originalSourcePubPlace	N.A.	N.A.
P33	play_original_source_publication_year	N.A.	*.originalSourceYear	originalSourceYear	N.A.	N.A.
P34	play_original_source_num_of_pages	N.A.	*.originalSourceNumberOfPages	originalSourceNumberOfPages	N.A.	N.A.
P35	play_num_of_wikipedia_links	N.A.	*.wikipediaLinkCount	wikipediaLinkCount	N.A.	wikipediaLinkCount
P36	play_segments	N.A.	N.A.	N.A.	segments	N.A.
P37	play_num_of_segments	N.A.	*.numOfSegments	numOfSegments	N.A.	N.A.
P38	play_num_of_acts	N.A.	*.numOfActs	numOfActs	N.A.	N.A.
P39	play_num_of_paragraphs	N.A.	*.numOfP	numOfP	N.A.	N.A.
P40	play_num_of_verse_lines	N.A.	*.numOfL	numOfL	N.A.	N.A.
P41	play_num_of_word_tokens_in_text_elements	N.A.	*.wordCountText	wordCountText	N.A.	N.A.
P42	play_num_of_word_tokens_in_sp	N.A.	*.wordCountSp	wordCountSp	N.A.	N.A.
P43	play_num_of_word_tokens_in_stage	N.A.	*.wordCountStage	wordCountStage	N.A.	N.A.
P44	play_characters	N.A.	N.A.	N.A.	cast	N.A.
P45	play_num_of_speakers	N.A.	*.numOfSpeakers	numOfSpeakers	N.A.	N.A.
P46	play_num_of_speakers_gender_female	N.A.	*.numOfSpeakersFemale	numOfSpeakersFemale	N.A.	N.A.
P47	play_num_of_speakers_gender_male	N.A.	*.numOfSpeakersMale	numOfSpeakersMale	N.A.	N.A.
P48	play_num_of_speakers_gender_unknown	N.A.	*.numOfSpeakersUnknown	numOfSpeakersUnknown	N.A.	N.A.
P49	play_num_of_person_groups	N.A.	*.numOfPersonGroups	numPersonGroups	N.A.	N.A.
P50	play_all_in_segment	N.A.	N.A.	N.A.	allInSegment	N.A.
P51	play_all_in_index	N.A.	N.A.	N.A.	allInIndex	N.A.

P52	play_character_relations	N.A.	N.A.	N.A.	relations	N.A.
P53	play_network_data_csv_url	dramas.*.networkdataCsvUrl	N.A.	N.A.	N.A.	N.A.
P54	play_network_nodes	N.A.	N.A.	N.A.	N.A.	nodes
P55	play_network_size	dramas.*.networkSize	*.size	size	N.A.	size
P56	play_network_num_edges	N.A.	*.numEdges	numEdges	N.A.	numEdges
P57	play_network_average_degree	N.A.	*.averageDegree	averageDegree	N.A.	averageDegree
P58	play_network_density	N.A.	*.density	density	N.A.	density
P59	play_network_diameter	N.A.	*.diameter	diameter	N.A.	diameter
P60	play_network_average_path_length	N.A.	*.averagePathLength	averagePathLength	N.A.	averagePathLength
P61	play_network_average_clustering	N.A.	*.averageClustering	averageClustering	N.A.	averageClustering
P62	play_network_num_connected_components	N.A.	*.numConnectedComponents	numConnectedComponents	N.A.	numConnectedComponents
P63	play_network_max_degree	N.A.	*.maxDegree	maxDegree	N.A.	maxDegree
P64	play_network_max_degree_character_ids	N.A.	*.maxDegreeIds	maxDegreeIds	N.A.	maxDegreeIds

Tab. 02: Play Features

The play objects contained in the array in the response of the `/corpora/{corpusname}` endpoint [C20] contain data about a single play including identifiers (fields “id” [P2], “name” [P3], “wikidataId” [P4]), titles (“title” [P5], “subtitle” [P6]), author(s) (“authors” [P9, 11, 13, 19, 20], and, deprecated, “author” [P15]), dates (“writtenYear” [P24], “printYear” [P25], “premiereYear” [P26] and “yearNormalized” [P27]) information about the source (“source” [P28], “sourceUrl” [P29]), the number of nodes of the extracted network, which is equal to the number of characters (“networkSize” [P55]), and a link to the data of the network as CSV (“networkdataCsvUrl” [P53]). Even more extensive data for all plays in a corpus including count-based [P37–43, 45–49] and network-based metrics [P55–64] is returned by the two endpoints `/corpora/{corpusname}/metadata`[81], returning a JSON file, and

[81]The interactive documentation is available at https://dracor.org/doc/api#/public/corpus-metadata. The endpoint is implemented with the function `api:corpus-meta-data` <22>, which generates its data with the function `dutil:get-corpus-meta-data` <21>. Because the data generated is also used by the endpoint returning a CSV, the JSON object is flat and resembles the columns in the CSV file. This has an effect on the serialization of fields, that have multiple values, e.g. “maxDegreeIds”, which is serialized as a string value, not as an array (cf. <23>).

`/corpora/{corpusname}/metadata/csv`[82] returning the information as a table in the CSV format. Characters and segments (i.e. acts and scenes)[83] of a single play can be fetched from the `/corpora/{corpusname}/play/{playname}` endpoint.[84] The endpoint `/corpora/{corpusname}/play/{playname}/metrics`[85] predominantly contains the network metrics, that are calculated by the DraCor Metrics Service (see section 5.4) upon loading the data into the database, i.e. “size” [P55], “numEdges” [P56], “averageDegree” [P57], “density” [P58], “diameter” [P59], “averagePathLength” [P60], “averageClustering” [P61], “numConnectedComponents” [P62], “maxDegree” [P63] and “maxDegreeIds” [P64].

The data on plays and corpora is also available in a RDF serialization that is generated upon loading a corpus into the database. The endpoint is `/corpora/{corpusname}/play/{playname}/rdf`.[86] The whole TEI source document of a play can be retrieved from the `/corpora/{corpusname}/play/{playname}/tei` endpoint.[87]

[82] The endpoint `/corpora/{corpusname}/metadata.csv`, which provided the metadata table, is deprecated and should not be used. Guides on API design discourage including file extensions in API URL patterns, cf. “Rule: File extensions should not be included in URIs” (Massé 2012:13). The interactive documentation of the endpoint `/corpora/{corpusname}/metadata/csv` is available at: https://dracor.org/doc/api#/public/corpus-metadata-csv-endpoint. The functionality is implemented by two functions: `api:corpus-meta-data-csv` <24> which wraps the function `api:get-corpus-meta-data-csv` <25>. As in the case of its JSON returning twin endpoint, it generates the data with the util function `dutil:get-corpus-meta-data` <21>, but serializes it as CSV. The columns are defined in the “api.xqm” module as a variable “metadata-columns” <26> and, probably, contain the most compact overview of available fields of a single corpus available in DraCor.

[83] The designated endpoint that returns a list of segments in a custom XML `/corpora/{corpusname}/play/{playname}/segmentation` is deprecated, see https://dracor.org/doc/api#/public/play-segments-xml. The segments can be found in the response of the `/corpora/{corpusname}/play/{playname}` endpoint, albeit in JSON.

[84] The endpoint is implemented with the function `api:play-info` <27> which retrieves the data from the function `dutil:get-play-info` <28>.

[85] The endpoint is implemented with the function `api:play-metrics` <29> which retrieves the data from the function `dutil:get-play-metrics` <30>.

[86] The endpoint is implemented by the function `api:play-rdf` <31>. It returns the file contained in the collection `/db/data/dracor/rdf/{corpusname}`. The module that generates the RDF is still under development. In the version deployed on the DraCor server the module “rdf.xqm” (https://github.com/dracor-org/dracor-api/blob/main/modules/rdf.xqm) generates the triples based on the DraCor ontology https://github.com/dracor-org/dracor-schema/blob/main/ontology/dracor-ontology.xml; an updated version of the module “rdf.xqm” will additionally serialize the information according to the standard ontologies CIDOC-CRM (https://www.cidoc-crm.org) and the CIDOC harmonized ontology FRBRoo (https://www.cidoc-crm.org/frbroo). The current state of the module can be accessed here: https://github.com/dracor-org/dracor-api/blob/43-refine-rdf-generation/modules/rdf.xqm.

[87] The endpoint is implemented with the function `api:play-tei` <32> that gets the data from the function `dutil:get-doc` <33>. The data stored in the collection `/db/data/dracor/tei/{corpusname}` is returned.

5.3.3.3 Information on Key Constituents of a Play

Segments

Information on segments, i.e., acts, scenes and other types of structuring a play, e.g., “appearances” [de: “Auftritte”] is not available as a single designated endpoint. Some information on the segmentation of a play is represented by the features “play_num_of_acts” [P38] and “play_num_of_segments” [P37]. The structure of a play in segments can be inferred from the field with the key “segments” [P36] in the response object of the `/corpora/{corpusname}/play/{playname}` endpoint.[88] To some extent the feature “play_all_in_segment” [P50] is also a feature of a single segment, because it marks the segment in which all characters have appeared on the stage at least once. One of the reasons, why the class segment is somewhat underrepresented might be due to the fact that contrary to the other entity classes corpus, play and character, there are no explicit IDs on segments,[89] i.e., an attribute `@xml:id` on the element `<div>` is not used, which makes it difficult to address a single segment and implement a designated endpoint.

Tab. 03 gives an overview of the features of a segment that are included in the response of `/corpora/{corpusname}/play/{playname}` endpoint. The deprecated endpoint `/corpora/{corpusname}/play/{playname}/segmentation` is not taken into account.

		endpoints
No.	**Feature**	**/corpora/{corpusname}/play/{playname}**
S1	segment_type	segments.*.type
S2	segment_number	segments.*.number
S3	segment_title	segments.*.title
S4	segment_speaking_characters	segments.*.speakers

Tab. 03: Segment Features

Characters

Data on the characters of a play are, on the one hand, included in the response of the endpoint `/corpora/{corpusname}/play/{playname}` in the field with the key “cast” [P44], on the

[88] The function `dutil:get-play-info` that is generating the response data of `/corpora/{corpusname}/play/{playname}` retrieves the information on the segments with the function `dutil:get-segments` <34>.

[89] We are still discussing at which level(s) and in which depth the assignment of IDs makes sense. In fact, such an assignment presupposes a more extensive (and thus debatable) modeling of the dramatic text. In addition, there is a risk that assigning IDs at too fine a granularity (e.g., at the word level) will significantly overload the XML.

other hand, there are two designated endpoints returning a list of characters: The endpoint `/corpora/{corpusname}/play/{playname}/cast`[90] returns this data in JSON, the endpoint `/corpora/{corpusname}/play/{playname}/cast/csv`[91] returns it as a table in CSV.

Tab. 04 lists the features available for a single character. The table includes the above discussed character-specific endpoints, but also the endpoints `/corpora/{corpusname}/play/{playname}/metrics` which contains a field "nodes" representing the characters [P54] and `/corpora/{corpusname}/play/{playname}/spoken-text-by-character`, because the spoken text of a character can be considered a feature of a character. In total 15 character features are available from five endpoints.

		endpoint				
No.	**Feature**	**/corpora/{corpusname}/play/{playname}**	**/corpora/{corpusname}/play/{playname}/metrics**	**/corpora/{corpusname}/play/{playname}/cast**	**/corpora/{corpusname}/play/{playname}/cast/csv**	**/corpora/{corpusname}/play/{playname}/spoken-text-by-character**
Ch1	character_id	cast.*.id	nodes.*.id	*.id	id	*.id
Ch2	character_name	cast.*.name	N.A.	*.name	name	*.label
Ch3	character_is_group	cast.*.isGroup	N.A.	*.isGroup	isGroup	*.isGroup
Ch4	character_gender	cast.*.sex	N.A.	*.gender	gender	*.gender
Ch5	character_wikidata_id	cast.*.wikidata_id	N.A.	*.wikidataId	wikidataId	N.A.
Ch6	character_node_betweenness	N.A.	nodes.*.betweenness	*.betweenness	betweenness	N.A.
Ch7	character_node_degree	N.A.	nodes.*.degree	*.degree	degree	N.A.

[90] The interactive documentation of the endpoint is available at: https://dracor.org/doc/api#/public/get-cast. The endpoint is implemented with the function `api:cast-info` <35>, that retrieves the data from the function `dutil:cast-info` <36>.

[91] The interactive documentation of the endpoint is available at: https://dracor.org/doc/api#/public/get-cast-csv. The endpoint is implemented with the function `api:cast-info-csv-ext` <37>. The same result can be achieved when sending the appropriate "Accept" header with the value "text/csv" in the request to the `/corpora/{corpusname}/play/{playname}/cast` endpoint, that evaluates the headers and returns data according to the requested MIME-type. This is implemented by specifying the "Accept" header in the function annotation of the function `api:cast-info-csv` <38> here: <39>. The data is generated by using the same function as `/corpora/{corpusname}/play/{playname}/cast`, i.e., `dutil:cast-info`. The columns are set in `api:cast-info-csv` and the data returned by `dutil:cast-info` ist filtered accordingly in the api function.

Ch8	character_node_closeness	N.A.	nodes.*.closeness	*.closeness	closeness	N.A.
Ch9	character_node_eigenvector	N.A.	nodes.*.eigenvector	*.eigenvector	eigenvector	N.A.
Ch10	character_node_weighted_degree	N.A.	nodes.*.weighted Degree	*.weightedDegree	weightedDegree	N.A.
Ch11	character_num_of_segments	N.A.	N.A.	*.numOfScenes	numOfScenes	N.A.
Ch12	character_num_of_sp	N.A.	N.A.	*.numOfSpeech Acts	numOfSpeechActs	N.A.
Ch13	character_num_of_word_tokens	N.A.	N.A.	*.numOfWords	numOfWords	N.A.
Ch14	character_roles	N.A.	N.A.	N.A.	N.A.	*.roles
Ch15	character_spoken_text	N.A.	N.A.	N.A.	N.A.	*.text

Tab. 04: Character Features

Spoken Text

The spoken text by characters, i.e., the text without stage directions, is available from the endpoint `/corpora/{corpusname}/play/{playname}/spoken-text`. The text can be filtered by setting the query parameters “gender”, “relation” and “role”. The data is returned in PLAINTEXT format.[92] The endpoint `/corpora/{corpusname}/play/{playname}/spoken-text-by-character` returns the spoken text grouped by character. Depending on the “Accept” header in the request, the endpoint can return the data in JSON format or in a table as CSV.[93]

An example taken from the play “Emilia Galotti” (https://dracor.org/id/ger000088) can demonstrate how the stage directions encoded with the TEI element `<stage>` are excluded from the spoken text in the API outputs. The following snippet is taken from the TEI[94], spoken text is in bold script.

```
<div type="act">
    <head>Erster Aufzug</head>
    <stage>Die Szene, ein Kabinett des Prinzen.</stage>
    <div type="scene">
```

[92] The interactive documentation is available at: https://dracor.org/doc/api#/public/play-spoken-text. The endpoint is implemented with the function `api:spoken-text` <40>. The filtering is implemented in the function `dutil:get-speech-filtered` <41>. If no parameters are set, the function `dutil:get-speech` <42> is used.

[93] The interactive documentation is available at: https://dracor.org/doc/api#/public/play-spoken-text-by-character. The endpoint is implemented with the function `api:spoken-text-by-character` which calls `api:get-spoken-text-by-character` <43>. The data is generated with the local function `local:get-text-by-character` <44>. The CSV is generated with `api:spoken-text-by-character-csv` <45>, the JSON with `api:spoken-text-by-character-json` <46>.

[94] Cf. <47>.

```
            <head>Erster Auftritt</head>
            <sp who="#der_prinz">
                <speaker>DER PRINZ</speaker>
                 <stage>an einem Arbeitstische, voller Briefschaften und Papiere,
                 deren einige er durchläuft.</stage>
                 <p>Klagen, nichts als Klagen! Bittschriften, nichts als
                 Bittschriften! – Die traurigen Geschäfte; und man beneidet uns
                 noch! – Das glaub' ich; wenn wir allen helfen könnten:
                 dann wären wir zu beneiden. – Emilia? <stage>Indem er noch eine
                 von den Bittschriften aufschlägt, und nach dem unterschriebnen
                 Namen sieht.</stage> Eine Emilia? – Aber eine Emilia Bruneschi –
                 nicht Galotti. Nicht Emilia Galotti! – Was will sie, diese Emilia
                 Bruneschi? <stage>Er lieset.</stage> <!-- … --></p>
            </sp>
            <!-- ... -->
        </div>
    </div>
```

The fragment above retrieved as PLAINTEXT response from the endpoint `/corpora/{corpusname}/play/{playname}/spoken-text`[95] looks as such:

> Klagen, nichts als Klagen! Bittschriften, nichts als Bittschriften! – Die traurigen Geschäfte; und man beneidet uns noch! – Das glaub' ich; wenn wir allen helfen könnten: dann wären wir zu beneiden. – Emilia? Eine Emilia? – Aber eine Emilia Bruneschi – nicht Galotti. Nicht Emilia Galotti! – Was will sie, diese Emilia Bruneschi?

The JSON array contains the text grouped by character as an object with some metadata on the speaker and the text included in the field with the key “text”, which contains an array with the individual speeches as items. The following example below is the same snipped as in the previous example in a different serialization:[96]

```
[ {
  "id" : "der_prinz",
  "label" : "Der Prinz",
  "isGroup" : false,
  "gender" : "MALE",
  "roles" : [ ],
  "text" : [ "Klagen, nichts als Klagen! Bittschriften, nichts als
Bittschriften! – Die traurigen Geschäfte; und man beneidet uns noch! – Das
glaub' ich; wenn wir allen helfen könnten: dann wären wir zu beneiden. –
```

[95] https://dracor.org/api/corpora/ger/play/lessing-emilia-galotti/spoken-text
[96] https://dracor.org/api/corpora/ger/play/lessing-emilia-galotti/spoken-text-by-character

```
Emilia? Eine Emilia? – Aber eine Emilia Bruneschi – nicht Galotti. Nicht Emilia
Galotti! – Was will sie, diese Emilia Bruneschi?
…
]}, … ]
```

Stage Directions

The text of stage directions can be retrieved from the endpoints `/corpora/{corpusname}/play/{playname}/stage-directions`[97], which returns all stage directions of a play excluding the speaker labels and `/corpora/{corpusname}/play/{playname}/stage-directions-with-speakers`[98] which includes speaker labels. Both endpoints return the data in plain text format.

To demonstrate the filtering in the output, the same excerpt from the play "Emilia Galotti" (https://dracor.org/id/ger000088) is used. Stage directions are encoded using the TEI element `<stage>` in the source file and are included in the API output. The following snippet is taken from the TEI[99], stage directions are in bold script, the speaker label in `<speaker>` is in italics:

```
<div type="act">
    <head>Erster Aufzug</head>
    <stage>Die Szene, ein Kabinett des Prinzen.</stage>
    <div type="scene">
        <head>Erster Auftritt</head>
        <sp who="#der_prinz">
              <speaker>DER PRINZ</speaker>
              <stage>an einem Arbeitstische, voller Briefschaften und Papiere,
              deren einige er durchläuft.</stage>
              <p>Klagen, nichts als Klagen! Bittschriften, nichts als
              Bittschriften! – Die traurigen Geschäfte; und man beneidet uns
              noch! – Das glaub' ich; wenn wir allen helfen könnten: dann wären
              wir zu beneiden. – Emilia? <stage>Indem er noch eine von den
              Bittschriften aufschlägt, und nach dem unterschriebnen Namen
              sieht.</stage> Eine Emilia? – Aber eine Emilia Bruneschi – nicht
              Galotti. Nicht Emilia Galotti! – Was will sie, diese Emilia
              Bruneschi? <stage>Er lieset.</stage> <!-- … --></p>
        </sp>
```

[97] The interactive documentation is available at: https://dracor.org/doc/api#/public/play-stage-directions. The endpoint is implemented with the function `api:stage-directions` <48>, which generates the plaintext data by getting the text nodes contained in the TEI element `<stage>` <49>.
[98] The interactive documentation is available at: https://dracor.org/doc/api#/public/play-stage-directions-with-speakers. The endpoint is implemented with the function `api:stage-directions-with-speakers` <50>, which also generates the returned data.
[99] Cf. <47>.

```
            <!-- ... -->
        </div>
    </div>
```

The response of the `/corpora/{corpusname}/play/{playname}/stage-directions` includes all stage directions separated by newlines[100]:

> Die Szene, ein Kabinett des Prinzen.
> an einem Arbeitstische, voller Briefschaften und Papiere, deren einige er durchläuft.
> Indem er noch eine von den Bittschriften aufschlägt, und nach dem unterschriebnen Namen sieht.
> Er lieset.

The response of the endpoint `/corpora/{corpusname}/play/{playname}/stage-directions-with-speakers` includes the speaker label "DER PRINZ" (here marked with bold font) before the first of the stage directions included in the speech act:[101]

> Die Szene, ein Kabinett des Prinzen.
> **DER PRINZ** an einem Arbeitstische, voller Briefschaften und Papiere, deren einige er durchläuft.
> Indem er noch eine von den Bittschriften aufschlägt, und nach dem unterschriebnen Namen sieht.
> Er lieset.

5.3.3.4 Network Data

Several endpoints offer the data of the extracted co-presence networks. The network metrics that are calculated upon the ingest of a corpus into the database with the help of the DraCor Metrics Service are provided by the endpoint `/corpora/{corpusname}/play/{playname}/metrics` (see also section 5.3.3.2 on play features and the paragraphs on character features in section 5.3.3.3 above).[102] The data is returned as JSON.

The network data is also available in a tabular form as CSV. The response of the endpoint `/corpora/{corpusname}/play/{playname}/networkdata/csv`[103] includes information

[100] https://dracor.org/api/corpora/ger/play/lessing-emilia-galotti/stage-directions
[101] https://dracor.org/api/corpora/ger/play/lessing-emilia-galotti/stage-directions-with-speakers
[102] The interactive documentation is available at: https://dracor.org/doc/api#/public/play-metrics. The endpoint is implemented with the function `api:play-metrics` <51>. The data is loaded from the stored files in the collection `/db/data/dracor/metrics/{corpusname}` and prepared for the response with the function `dutil:get-play-metrics` <52>.
[103] The interactive documentation is available at: https://dracor.org/doc/api#/public/network-csv. The endpoint is implemented with the function `api:networkdata-csv` <53>. The data is generated in this

on the edges of the network in the columns “Source”, “Type”, “Target” and “Weight”. The endpoint `/corpora/{corpusname}/play/{playname}/networkdata/gexf`[104] provides the network data in the XML based format GEXF[105] ready to be used with the network analysis tool *Gephi*[106]. The endpoint `/corpora/{corpusname}/play/{playname}/networkdata/graphml`[107] returns data as XML in the GraphML[108] format.

The designated functions that generate the network data rely on an intermediary structure. In the case of the above-mentioned endpoints, it is a custom XML; the case of the function that prepares the data for posting it to the metrics service, the XQuery data types `map` and `sequence` are used, that can be serialized to JSON objects and arrays. This structure is then transformed to the requested serializations in the designated rendering functions[109]:

```
<segments>
    <sgm>
        <spk>character_id_1</spk>
        <spk>character_id_2</spk>
     <sgm>
     <!-- … -->
</segments>
```

Another type of network is based on the social relations (available as play feature [P52]) that are encoded for some of the plays in the TEI element `<listRelation>` in the `<listPerson>`. The example below is a snippet from the code of the play “Emilia Galotti” (https://dracor.org/id/ger000088):[110]

```
<listRelation type="personal">
```

function based on segments and speakers retrieved from the corresponding utility functions `dutil:get-segments` and `dutil:distinct-speakers`.

[104] The interactive documentation is available at: https://dracor.org/doc/api#/public/network-gexf. The endpoint is implemented with the function `api:networkdata-gexf` <54>.

[105] The GEXF file format is described at https://gexf.net. Unfortunately, there seems to be no official MIME-Type available for this format, cf. https://www.iana.org/assignments/media-types/media-types.xhtml.

[106] https://gephi.org.

[107] The interactive documentation is available at: https://dracor.org/doc/api#/public/network-graphml. The endpoint is implemented with the function `api:networkdata-graphml` <55>.

[108] Specification of the GraphML format: http://graphml.graphdrawing.org.

[109] <56> (CSV), <57> (GEXF), <58> (GraphML), https://github.com/dracor-org/dracor-api/blob/main/modules/metrics.xqm#L62-L72 <59> (in the function `metrics:get-network-metrics` that prepares the data for sending to metrics service). A similar structure is available from the deprecated segmentation endpoint, cf. https://dracor.org/doc/api#/public/play-segments-xml for documentation.

[110] Cf. <60>.

```
        <relation name="parent_of" active="#odoardo #claudia" passive="#emilia"/>
        <relation name="associated_with" active="#marinelli" passive="#der_prinz"/>
        <relation name="associated_with" active="#camillo_rota"
    passive="#der_prinz"/>
    </listRelation>
```

The endpoint `/corpora/{corpusname}/play/{playname}/relations/csv`[111] provides these relations as tabular data (in a CSV file) containing the columns “Source”, “Type”, “Target” and “Label”. The social relations in the play “Emilia Galotti” are provided in the response of the endpoint as follows:[112]

```
Source,Type,Target,Label
odoardo,Directed,emilia,parent_of
claudia,Directed,emilia,parent_of
marinelli,Directed,der_prinz,associated_with
camillo_rota,Directed,der_prinz,associated_with
```

This information can be requested in two other XML formats to be used with network analysis software. The endpoint `/corpora/{corpusname}/play/{playname}/relations/gexf`[113] returns the relation data of a play as GEXF and `/corpora/{corpusname}/play/{playname}/relations/graphml`[114] as GraphML.

5.3.3.5 Cross-Corpora Queries

There is one endpoint that provides a means to query across several plays and/or corpora. The endpoint `/character/{id}` lists plays having a character identified by a Wikidata ID.[115] This endpoint can be used to retrieve characters in plays that are actualizations of a common archetype, e.g., the “Faust” character. In the source TEI the reference to an entry on Wikidata is

[111] The interactive documentation is available at: https://dracor.org/doc/api#/public/relations-csv. The endpoint is implemented with the function `api:relations-csv` <61> that generates the social relations data using the function `dutil:get-relations` <62>.
[112] https://dracor.org/api/corpora/ger/play/lessing-emilia-galotti/relations/csv
[113] The interactive documentation is available at: https://dracor.org/doc/api#/public/relations-gexf. The endpoint is implemented with the function `api:relations-gexf` <63> that uses the social relations data that is included in the data returned of the function `dutil:get-play-info` <64>.
[114] The interactive documentation is available at: https://dracor.org/doc/api#/public/relations-graphml. The endpoint is implemented with the function `api:relations-graphml` <65> that uses the social relations data that is included in the data returned of the function `dutil:get-play-info` <64>.
[115] The interactive documentation is available at: https://dracor.org/doc/api#/public/plays-with-character. The endpoint is implemented with the function `api:plays-with-character` <66> that generates its data using the function `dutil:get-plays-with-character` <67>.

included in the attribute `@ana`. The code snippet below shows the encoding of the character "Faust" in Goethe's play:[116]

```
<person xml:id="faust" sex="MALE" ana="http://www.wikidata.org/entity/Q76918">
    <persName>Faust</persName>
</person>
```

The Wikidata ID can be used as the value of the path parameter "id" the endpoint as such: https://dracor.org/api/character/Q76918. The response includes a JSON array that lists 21 dramas with a Faust character linked to Wikidata. These range from early plays such as Weidmann's "Johann Faust" (1775)[117], Goethe's both "Faust" parts[118] and Vischer's "Faust III" (1862)[119], to mashups such as Grabbe's "Don Juan and Faust" (1829)[120] and a version with a female Faust, Wilhelm Schäfer's "Faustine" (1898)[121].

The endpoint also returns results from different corpora, as can be seen when querying for "Antigone" characters: https://dracor.org/api/character/Q131351, which returns results from the Greek, Roman, Italian and Russian Drama Corpora.

5.3.4 API Wrappers

For the programming languages Python and R, the packages "pydracor"[122] and "rdracor"[123] serve as API wrappers. For information on the two packages, see the CLS INFRA work package 7 Deliverable D7.2.

[116] Cf. <68>.
[117] https://dracor.org/id/ger000014.
[118] https://dracor.org/id/ger000243, https://dracor.org/id/ger000201.
[119] https://dracor.org/id/ger000064.
[120] https://dracor.org/id/ger000209.
[121] https://dracor.org/id/ger000302.
[122] Code repository on GitHub: https://github.com/dracor-org/pydracor. The package is published on PyPI ("Python Package Index") https://pypi.org/project/pydracor and can be installed with the command `pip3 install pydracor` on a machine with Python and pip available.
[123] Code repository on GitHub: https://github.com/dracor-org/rdracor; The package is published on CRAN ("Comprehensive R Archive Network") and with R the library can be installed with a single line of code `install.packages("rdracor")` and can be included into R projects with `library(rdracor)`.

5.4 DraCor Metrics Service

The network metrics provided by several API endpoints are calculated with a separate microservice implemented in Python.[124] This microservice exposes an API with a single endpoint `/metrics` that accepts POST requests. It expects the structural data extracted from the source TEI file as JSON in the body of the HTTP POST request. Below is an example of the data that can be posted to the service:

```
{"segments":[{"speakers":["speaker_1","speaker_2"]},{"speakers":["speaker_1","s
peaker_3"]}]}
```

For the sample data above, the service will return a JSON object, that will look like this:

```
{
    "size": 3,
    "density": 0.6666666666666666,
    "diameter": 2,
    "averagePathLength": 1.3333333333333333,
    "averageDegree": 1.3333333333333333,
    "averageClustering": 0.0,
    "maxDegree": 2,
    "maxDegreeIds": [
        "speaker_1"
    ],
    "numConnectedComponents": 1,
    "numEdges": 2,
    "nodes": {
        "speaker_1": {
            "degree": 2,
            "weightedDegree": 2,
            "betweenness": 1.0,
            "closeness": 1.0,
            "eigenvector": 0.707106690085642
        },
        "speaker_2": {
            "degree": 1,
            "weightedDegree": 1,
            "betweenness": 0.0,
            "closeness": 0.6666666666666666,
            "eigenvector": 0.5000000644180599
```

[124] The GitHub repository can be accessed at: https://github.com/dracor-org/dracor-metrics. The version reviewed in this report is 1.2.0, cf. https://github.com/dracor-org/dracor-metrics/releases/tag/v1.2.0.The code is in the file "main.py". The API is implemented with the package "hug" (https://pypi.org/project/hug).

```
                },
                "speaker_3": {
                    "degree": 1,
                    "weightedDegree": 1,
                    "betweenness": 0.0,
                    "closeness": 0.6666666666666666,
                    "eigenvector": 0.5000000644180599
                }
            }
        }
```

When using the Metrics Service as a component of the DraCor system, the eXist-db application will handle the extraction, send the request to the specified endpoint URL and decode the returned data.[125]

All metrics are calculated using the Python package *networkX* (Hagberg et al. 2008)[126] based on a Graph `G` that is constructed when the respective function is called.[127] Tab. 05 lists all metrics, whereas the first column contains the key of the field with the calculated value in the returned JSON object, the second column is the name of the metric and in column 3 the corresponding play (P) or character feature (Ch) is referenced (see Tab. 02 and 03 in section 5.3.3.2 resp. 5.3.3.3):

Key	**Metric**	**Feature**
size[128]	Network Size	P55
density[129]	Network Density	P58
diameter[130]	Network Diameter	P59

[125] This functionality is implemented with the function `metrics:get-network-metrics` <69>. The returned metrics are stored in a designated collection in the database `/db/data/dracor/metrics/{corpusname}`.
[126] https://networkx.org. The documentation provides extensive information on the algorithms implemented in the package: https://networkx.org/documentation/stable/reference.
[127] In the code, the graph is created here: <70>.
[128] Calculated by counting a list of nodes, see code <71>; Documentation: https://networkx.org/documentation/stable/reference/generated/networkx.classes.function.nodes.html#networkx.classes.function.nodes.
[129] Calculated with the networkx function `density` (https://networkx.org/documentation/stable/reference/generated/networkx.classes.function.density.html#networkx.classes.function.density).
[130] Cf. <72>.

numEdges[131]	Number of Edges	P56
averagePathLength[132]	Average Path Length	P60
averageDegree[133]	Average Degree	P57
averageClustering[134]	Average Clustering Coefficient	P61
maxDegree[135]	Maximum Degree	P63
maxDegreeIds[136]	Maximum Degree IDs	P64
numConnectedComponents[137]	Number of Connected Components	P62
nodes.*.degree[138]	Degree Centrality	Ch7
nodes.*.weightedDegree[139]	Weighted Degree	Ch10
nodes.*.betweenness[140]	Betweenness Centrality	Ch6
nodes.*.closeness[141]	Closeness Centrality	Ch8

[131] Uses the networkx function `number_of_edges` (https://networkx.org/documentation/stable/reference/generated/networkx.classes.function.number_of_edges.html#networkx.classes.function.number_of_edges).
[132] Cf. <73>.
[133] Cf. <74>.
[134] Cf. <75>; Documentation: https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.cluster.average_clustering.html#networkx.algorithms.cluster.average_clustering.
[135] Cf. <76>.
[136] Cf. <77>.
[137] Uses the networkx function `number_connected_components` (https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.components.number_connected_components.html#networkx.algorithms.components.number_connected_components).
[138] Uses the networkx `function degree` (https://networkx.org/documentation/stable/reference/generated/networkx.classes.function.degree.html#networkx.classes.function.degree).
[139] Uses the networkx `function degree` (https://networkx.org/documentation/stable/reference/generated/networkx.classes.function.degree.html#networkx.classes.function.degree).
[140] Uses the networkx function `betweenness_centrality` (https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.centrality.betweenness_centrality.html#networkx.algorithms.centrality.betweenness_centrality.
[141] Uses the networkx function `closeness_centrality` (https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.centrality.closeness_centrality.html#networkx.algorithms.centrality.closeness_centrality).

nodes.*.eigenvector[142]	Eigenvector centrality	Ch9

Tab. 05: Network metrics provided by the DraCor Metrics Service

5.5 DraCor SPARQL Endpoint

Data on corpora and plays is transformed to RDF upon loading corpora into the database. This data is not only stored in an XML serialization inside the eXist-db[143], but also loaded into a triple store[144]. The current setup uses *Apache Jena Fuseki*[145] that provides a SPARQL endpoint.[146] YASGUI is used to provide a GUI.[147] YASGUI[148] not only supports writing SPARQL queries, but it also has several capabilities to display the results, e.g. a map view (see for example Fig. 04 in section 4.2).

5.6 DraCor Front-End

The central user interface of the DraCor system, that is running on the website https://dracor.org, is implemented as a single page application based on the *React Javascript* library[149] that retrieves its data from the API.[150] In the following section we describe the single components of the front-end[151] and especially emphasize its relation to the API. An overview of all endpoints and information if they are used can be seen in Tab. 06. The columns include the components of the website, the rows list all endpoints supporting HTTP GET requests. An entry marked with "fetched" means that data is actually retrieved from the API and rendered, whereas "linked" indicates that the page contains a link to an endpoint that triggers a download behavior.

[142] Uses the networkx function `eigenvector_centrality` (https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.centrality.eigenvector_centrality.html#networkx.algorithms.centrality.eigenvector_centrality).

[143] Data is stored in the collection `/db/data/dracor/rdf/{corpusname}` and can be retrieved with the `/corpora/{corpusname}/play/{playname}/rdf` endpoint.

[144] Repository of the Jena Fuseki Setup used for DraCor: https://github.com/dracor-org/dracor-fuseki

[145] https://jena.apache.org; Fuseki: https://jena.apache.org/documentation/fuseki2/index.html.

[146] The production version of the endpoint is available at https://dracor.org/fuseki/sparql.

[147] It can be accessed at https://dracor.org/sparql.

[148] https://yasgui.triply.cc. GitHub repository: https://github.com/TriplyDB/Yasgui.

[149] Code repository on GitHub: https://github.com/dracor-org/dracor-frontend. The app development was bootstrapped with "Create React App", see GitHub repository https://github.com/facebook/create-react-app.

[150] For sending requests to the API and parsing the responses the package "Apisauce" is used, see GitHub repository https://github.com/infinitered/apisauce.

[151] The GitHub repository of the DraCor front-end app we refer to in this report is version 1.5.0, cf. https://github.com/dracor-org/dracor-frontend/releases/tag/v1.5.0.

Endpoint	Landing Page	Corpus Page	Play Page					
			Play Page overall, Head	Tab "Network"	Tab "Relations"	Tab "Speech distribution"	Tab "Full text"	Tab "Downloads"
/info	fetched							
/corpora	fetched							
/corpora/{corpusname}		fetched						
/corpora/{corpusname}/ metadata		linked						
/corpora/{corpusname}/ metadata/csv		linked						
/corpora/{corpusname}/ play/{playname}			fetched	(already fetched)	(already fetched)	(already fetched)	(already fetched)	
/corpora/{corpusname}/ play/{playname}/metrics				fetched				
/corpora/{corpusname}/ play/{playname}/tei		linked					fetched	linked
/corpora/{corpusname}/ play/{playname}/rdf								
/corpora/{corpusname}/ play/{playname}/cast								linked
/corpora/{corpusname}/ play/{playname}/cast/csv								linked
/corpora/{corpusname}/ play/{playname}/networkdata/csv								linked
/corpora/{corpusname}/ play/{playname}/networkdata/gexf								linked
/corpora/{corpusname}/ play/{playname}/networkdata/graphml								linked
/corpora/{corpusname}/ play/{playname}/relations/csv								linked
/corpora/{corpusname}/ play/{playname}/relations/gexf								linked
/corpora/{corpusname}/ play/{playname}/relations/graphml[152]								

[152] To be included, cf. https://github.com/dracor-org/dracor-frontend/pull/246.

/corpora/{corpusname}/play/{playname}/spoken-text								linked
/corpora/{corpusname}/play/{playname}/spoken-text-by-character								linked
/corpora/{corpusname}/play/{playname}/stage-directions								linked
/corpora/{corpusname}/play/{playname}/stage-directions-with-speakers								linked
/id/{id}			Used in the frontend to redirect {dracor-app-url}/id/{dracor-id} to the play page.					
/character/{id}								
/author/{wikidataId}[153]			fetched					

Tab. 06: API endpoints used in the front-end

On each page on the DraCor front-end there is a header containing the logo and the top navigation[154] with the menu drop-down menus.

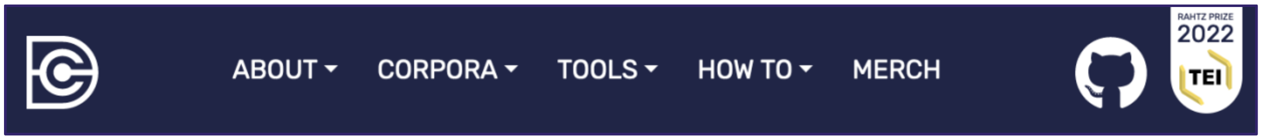


Fig. 11: DraCor menu

The menu item "About" contains links to pages providing information about DraCor, a FAQ, a page listing all people involved in creating the system and the corpora, and the imprint. All corpora are always accessible from the "Corpora" drop-down menu item.

[153] This is a "hidden" API endpoint that supports getting data for an author from Wikidata.
[154] React component "TopNav" <78>.

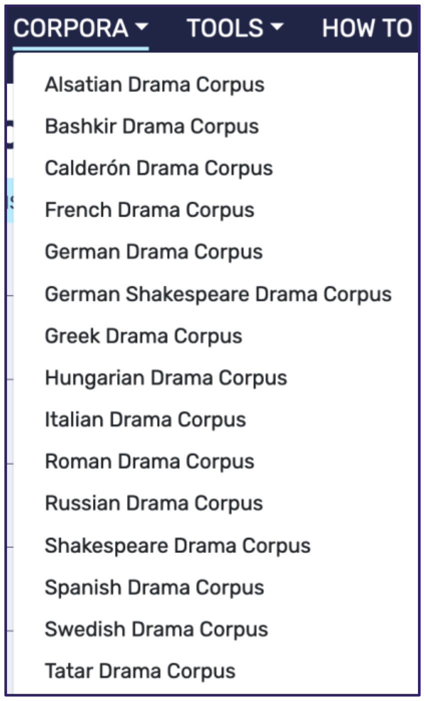


Fig. 12: DraCor corpora drop-down menu

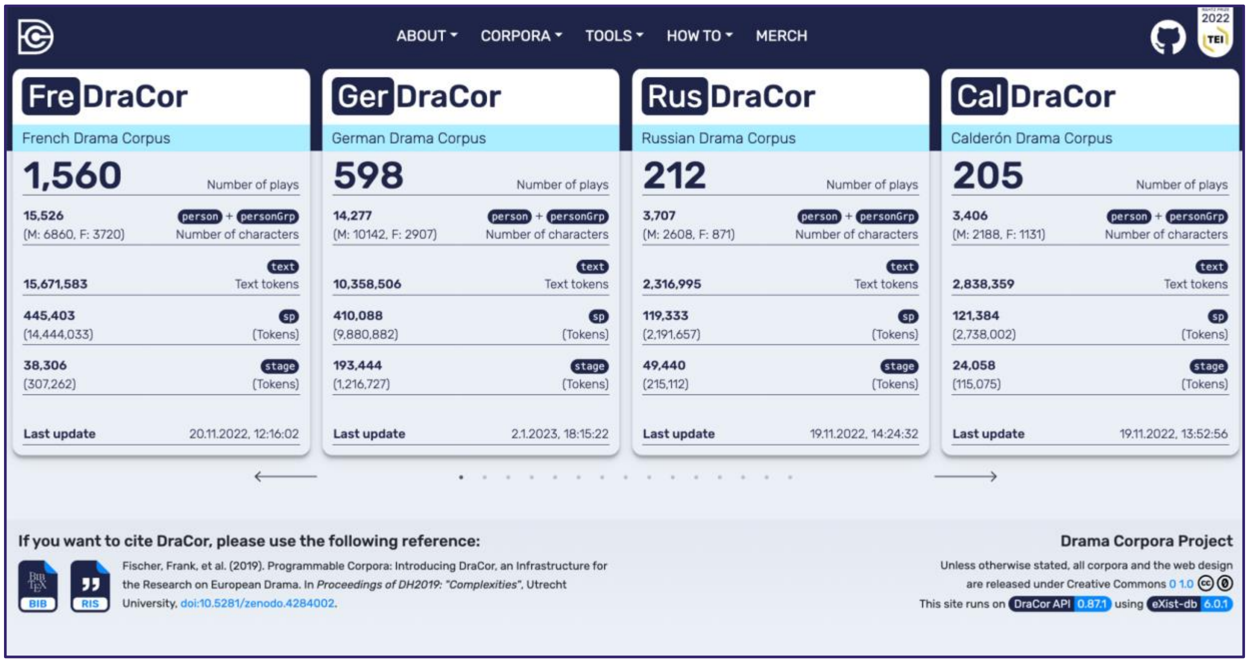
The menu item “Tools” contains links to the interactive API documentation implemented with Swagger UI[155], a link to the SPARQL interface[156] that is built on YASGUI[157], the editor “ezlinavis” (Easy Linavis)[158], that allows to quickly generate CSV files with network data from simple segmentations of a dramatic text written in a Markdown-inspired[159] notation and the web application “Shiny Dracor”[160] which provides an interface to interact and analyze networks. The menu item “How To” includes links to a page listing tutorials and teaching material[161] as well as an extensive list of research based on DraCor and the corpora provided[162]. The last item in the

[155] https://swagger.io/tools/swagger-ui.
[156] https://dracor.org/sparql
[157] https://yasgui.triply.cc, GitHub repository: https://github.com/TriplyDB/Yasgui.
[158] https://ezlinavis.dracor.org, cf. GitHub repository https://github.com/dracor-org/ezlinavis.
[159] Markdown is a simple markup language, invented by John Gruber and the already mentioned Aaron Swartz, cf. https://daringfireball.net/projects/markdown.
[160] https://shiny.dracor.org; the experimental web application is implemented with R and the web framework for interactive data visualizations “Shiny” (https://shiny.rstudio.com), cf. GitHub repository: https://github.com/dracor-org/dracor-shiny.
[161] https://dracor.org/doc/tutorials
[162] https://dracor.org/doc/research

top navigation “Merch”[163] links to a page listing available DraCor merchandising, e.g. the card game “Dramenquartett”[164].

The website’s footer[165] contains a hint how to cite Dracor and a box providing information on the version of the API and the underlying eXist-db. This data is retrieved from the API endpoint `/info` and renders the information accordingly.[166]

5.6.1 Landing Page: List of Corpora



FreDraCor French Drama Corpus
1,560 Number of plays
15,526 (M: 6860, F: 3720) person + personGrp Number of characters
15,671,583 text Text tokens
445,403 (14,444,033) sp (Tokens)
38,306 (307,262) stage (Tokens)
Last update 20.11.2022, 12:16:02

GerDraCor German Drama Corpus
598 Number of plays
14,277 (M: 10142, F: 2907) person + personGrp Number of characters
10,358,506 text Text tokens
410,088 (9,880,882) sp (Tokens)
193,444 (1,216,727) stage (Tokens)
Last update 2.1.2023, 18:15:22

RusDraCor Russian Drama Corpus
212 Number of plays
3,707 (M: 2608, F: 871) person + personGrp Number of characters
2,316,995 text Text tokens
119,333 (2,191,657) sp (Tokens)
49,440 (215,112) stage (Tokens)
Last update 19.11.2022, 14:24:32

CalDraCor Calderón Drama Corpus
205 Number of plays
3,406 (M: 2188, F: 1131) person + personGrp Number of characters
2,838,359 text Text tokens
121,384 (2,738,002) sp (Tokens)
24,058 (115,075) stage (Tokens)
Last update 19.11.2022, 13:52:56

If you want to cite DraCor, please use the following reference:
BIB RIS
Fischer, Frank, et al. (2019). Programmable Corpora: Introducing DraCor, an Infrastructure for the Research on European Drama. In *Proceedings of DH2019: "Complexities"*, Utrecht University, doi:10.5281/zenodo.4284002.

Drama Corpora Project
Unless otherwise stated, all corpora and the web design are released under Creative Commons 0 1.0
This site runs on DraCor API 0.87.1 using eXist-db 6.0.1

Fig. 13: DraCor landing page

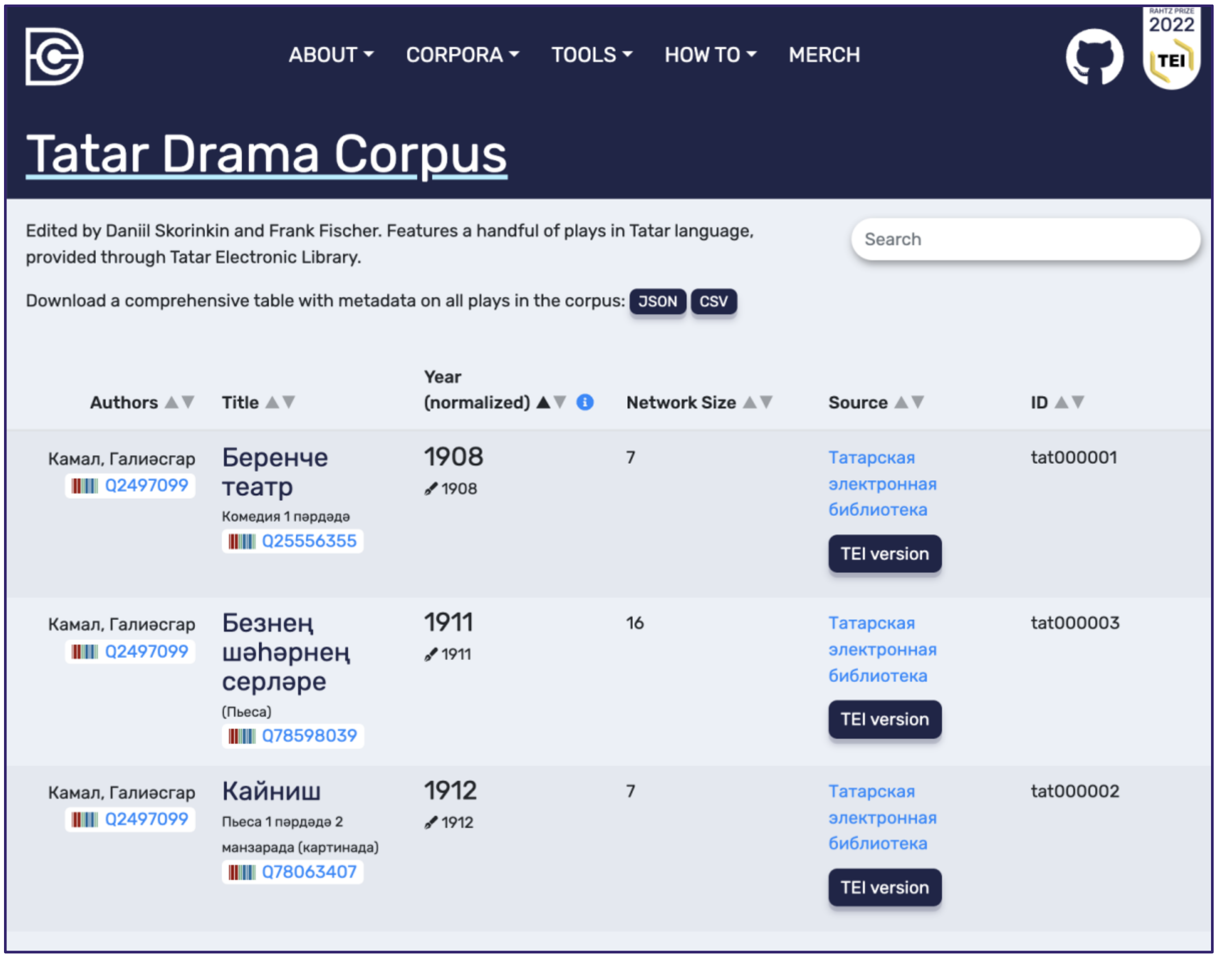
A user accessing the web application’s landing page[167] is presented with an overview page, on which all corpora are displayed as cards including some basic statistics (number of plays, characters, tokens in speeches and stage directions and the date of the last update of the corpus) in a slider. The data is fetched from the API’s endpoint `/corpora` with metrics included by setting the parameter “include” to “metrics”.[168]

---

[163] https://dracor.org/doc/merch
[164] https://dramenquartett.github.io, see the poster for a German version: https://doi.org/10.6084/m9.figshare.5926363.v1 and the poster for an English version: https://doi.org/10.6084/m9.figshare.6667424.v1 Cf. Fischer et al. 2018.
[165] React component “Footer” <79>.
[166] The data is fetched from the API by the function `fetchInfo` in “App.tsx” <80> . It uses a custom type “ApiInfo” defined in the “types.ts” <81>. The information is rendered in the component “Footer” <82>.
[167] React component “Home” <83>.
[168] The interactive documentation of the /corpora endpoint can be accessed at https://dracor.org/doc/api#/public/list-corpora. The React component “Corpora” <84> fetches the corpus

5.6.2 Corpus Page

Selecting a single corpus results in an overview of the corpus contents, which is generated by rendering the response of an API call to the `/corpora/{corpusname}` endpoint[169], e.g. the Tatar Drama Corpus containing three plays will be visualized as a sortable table,[170] as depicted in the following screenshot of https://dracor.org/tat[171].



Tatar Drama Corpus

Edited by Daniil Skorinkin and Frank Fischer. Features a handful of plays in Tatar language, provided through Tatar Electronic Library.

Download a comprehensive table with metadata on all plays in the corpus: JSON CSV

Authors	Title	Year (normalized)	Network Size	Source	ID
Камал, Галиәсгар Q2497099	Беренче театр Комедия 1 пәрдәдә Q25556355	1908 1908	7	Татарская электронная библиотека TEI version	tat000001
Камал, Галиәсгар Q2497099	Безнең шәһәрнең серләре (Пьеса) Q78598039	1911 1911	16	Татарская электронная библиотека TEI version	tat000003
Камал, Галиәсгар Q2497099	Кайниш Пьеса 1 пәрдәдә 2 манзарада (картинада) Q78063407	1912 1912	7	Татарская электронная библиотека TEI version	tat000002

Fig. 14: TatDraCor corpus page

data from the API with the function `fetchData` <85> and displays them as single corpus cards, which are rendered using the component "CorpusCard" <86>.

[169] The interactive documentation of the /corpora/{corpusname} endpoint can be accessed at https://dracor.org/doc/api#/public/list-corpus-content. The data is retrieved with the function `fetchCorpus` <87> in the "Corpus" component <88>. The table is rendered using the component "CorpusIndex" <89>.

[170] The table relies on the package "react-bootstrap-table2": https://react-bootstrap-table.github.io/react-bootstrap-table2; GitHub repository: https://github.com/react-bootstrap-table/react-bootstrap-table2.

[171] The response of the corpora `/corpora/{corpusname}` endpoint for the Tatar Drama Corpus (TatDraCor) can be inspected at https://dracor.org/api/corpora/tat.

The buttons “TEI version” in the table column “Source”, which allows to directly download the TEI-XML file of a single play allow to download the data as returned by the API endpoint `/corpora/{corpusname}/play/{playname}/tei`.[172]

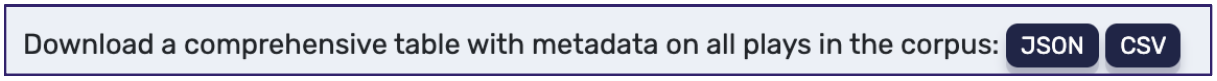

Download a comprehensive table with metadata on all plays in the corpus:

Fig. 15: Download buttons for corpus metadata

Two links at the top of the page provide an easy way to download the corpus metadata in JSON or CSV format. They link directly to the API endpoints `/corpora/{corpusname}/metadata` and `/corpora/{corpusname}/metadata/csv`.[173]

5.6.3 View of a Single Play

There is a designated page that offers information on a single play. It includes several “React” components and renders data fetched (mainly) from the `/corpora/{corpusname}/play/{playname}`[174] as tabs[175], that are displayed below a static header with basic information on the play and its author.[176] The portrait, as well as dates and places of birth and death are retrieved from Wikidata. The front-end uses the “hidden” endpoint `/author/{wikidataId}`.

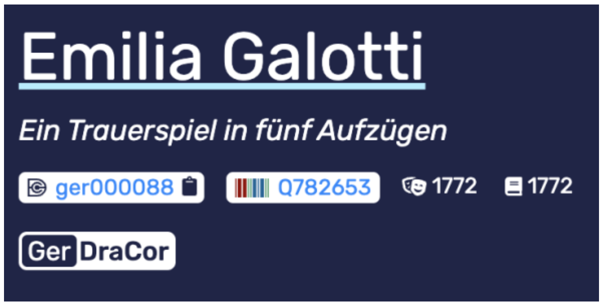
[172] The interactive documentation of the endpoint can be accessed at https://dracor.org/doc/api#/public/play-tei. The component “CorpusIndex” does not fetch the TEI data, but assigned the respective URL of the TEI endpoint to a variable <90>, which is then used in the `href` attribute of the “download button” <91> .

[173] The interactive documentation of the endpoints can be accessed at: https://dracor.org/doc/api#/public/corpus-metadata and https://dracor.org/doc/api#/public/corpus-metadata-csv-endpoint.

[174] The component “PlayInfo” contained in “Play.js” <92> fetches the data using the function `fetchPlay` <93> from the endpoint. The interactive documentation of the endpoint `/corpora/{corpusname}/play/{playname}` can be accessed at https://dracor.org/doc/api#/public/play-info.

[175] In “Play.js” the component “PlayDetailsTab” is used, which includes the content of the tab base on a control flow structure (if/else if/else) <94> evaluating the variable “tab” <95>.

[176] The components that is used to render information on the play (title and subtitle, the various dates and the IDs) is contained in the file “PlayDetailsHeader.js” <96>. It uses the component “AuthorInfo” <97>, which renders the information on a play’s author by sending a request to the (undocumented) API endpoint `/author/{wikidataId}` (see <98>using the function `fetchAuthorInfo` <99>. An example of the response for Gotthold Ephraim Lessing can be seen at https://dracor.org/api/author/Q34628.



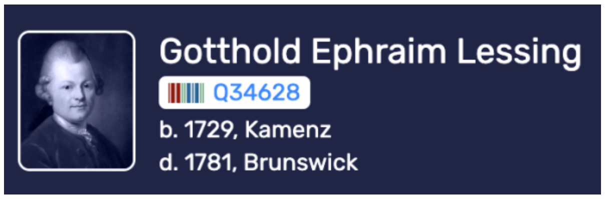
Emilia Galotti

Ein Trauerspiel in fünf Aufzügen

ger000088 Q782653 1772 1772

GerDraCor

Fig. 16: Play info



Gotthold Ephraim Lessing

Q34628

b. 1729, Kamenz

d. 1781, Brunswick

Fig. 17: Author info

5.6.3.1 Tab "Network"

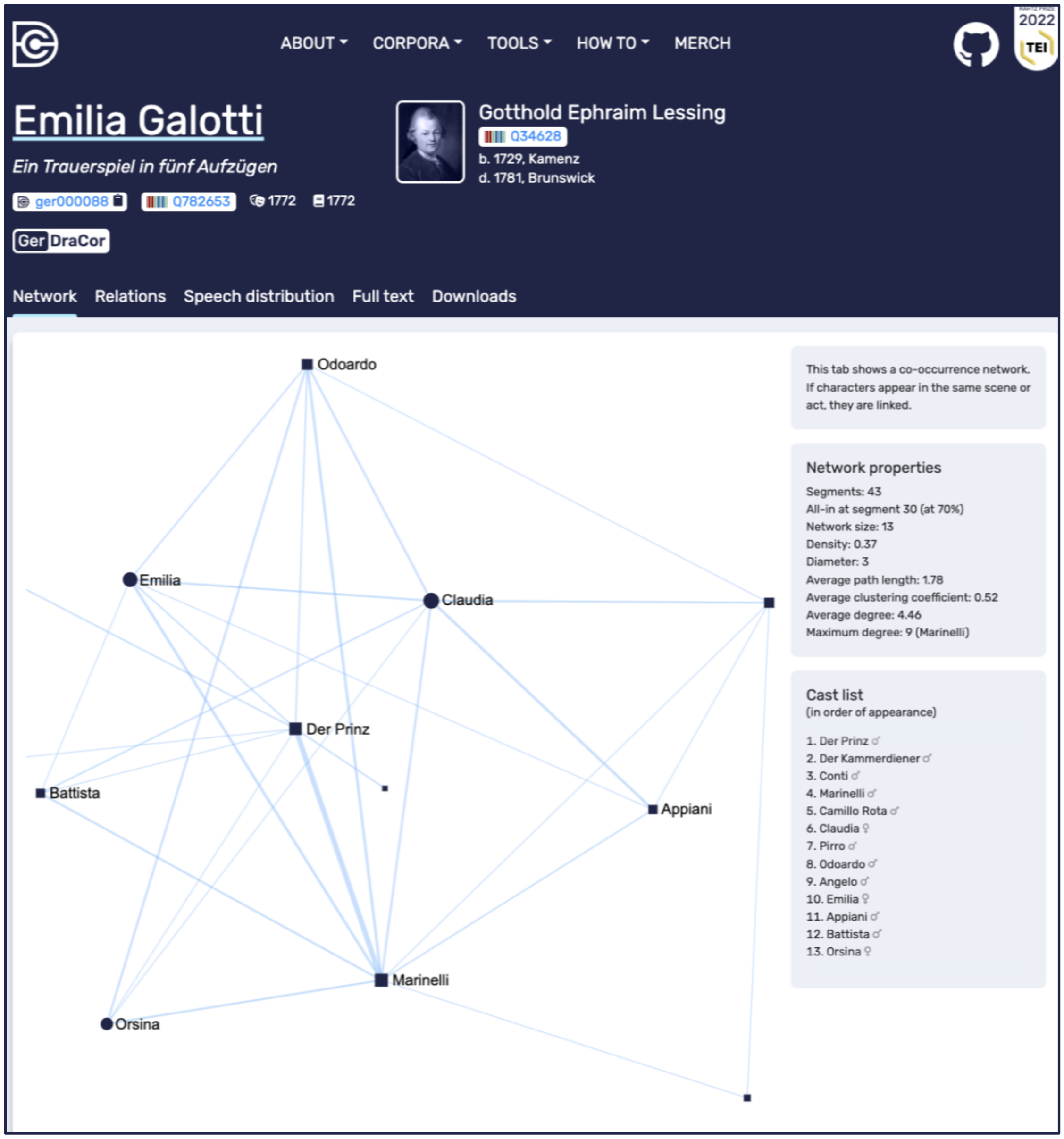
The tab "Network" renders a graph[177] based on co-occurrences of characters in a segment[178] and lists some basic network metrics extracted from the play and its characters. The network metrics are retrieved from the endpoint `/corpora/{corpusname}/play/{playname}/metrics`.[179] The information about the characters of a play that are displayed in the box "Cast List" in the sidebar are fetched from the `/corpora/{corpusname}/play/{playname}` endpoint.[180] The gender of a character is indicated by a symbol following the name, groups of characters are also marked. If a Wikidata identifier of a character is contained in the data an icon with the Wikidata logo as a link to the character's entity page is displayed next to the character's name (see example of Sophocles: Antigone, https://dracor.org/id/greek000025).

[177] The graph visualization is implemented using a React wrapper for the library "SigmaJS" (https://www.sigmajs.org). See https://sim51.github.io/react-sigma for more information on the package. Code repository on GitHub: https://github.com/sim51/react-sigma. The component that is used to render with the component "NetworkGraph" <100>. It uses the layout algorithm "ForceAtlas2" in an implementation for SigmaJS (https://logicatcore.github.io/assets/js/sigma.js-1.2.1/plugins/sigma.layout.forceAtlas2). The actual settings of the layout are defined in the code of the component at <101>.

[178] None of the pre-calculated graph formats, that are provided by the API (endpoints `/corpora/{corpusname}/play/{playname}/networkdata/[csv|gexf|graphml]`) are used. Instead, the graph is created from the character ("cast") and segment ("segments") data that is returned from `/corpora/{corpusname}/play/{playname}`. The file "network.js" contains two functions that are used: Co-occurrences (edges between character nodes) are calculated with the function `getCooccurrences` <102> based on the "segments". The function `makeGraph` <103> creates the graph data that can be rendered with SigmaJS.

[179] The data is rendered with the component "PlayMetrics" in "PlayMetrics.js" <152>. The interactive documentation of the endpoint `/corpora/{corpusname}/play/{playname}/metrics` is available at https://dracor.org/doc/api#/public/play-metrics.

[180] The data is retrieved by the function `fetchPlay` in "Play.js" already the "Play page" is loaded (see footnote 175) and passed to the component "CastList" <104>.



This tab shows a co-occurrence network. If characters appear in the same scene or act, they are linked.

Network properties

Segments: 43
All-in at segment 30 (at 70%)
Network size: 13
Density: 0.37
Diameter: 3
Average path length: 1.78
Average clustering coefficient: 0.52
Average degree: 4.46
Maximum degree: 9 (Marinelli)

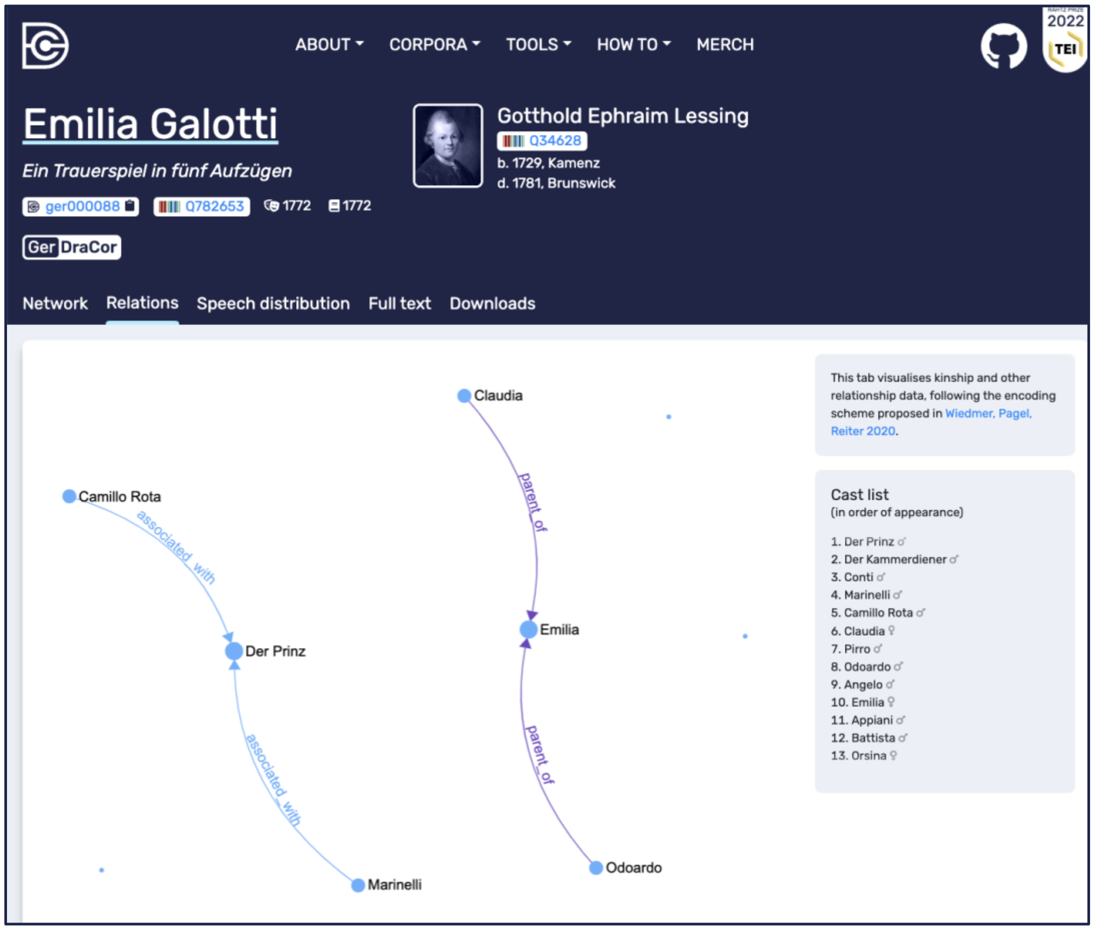
Cast list
(in order of appearance)

1. Der Prinz ♂
2. Der Kammerdiener ♂
3. Conti ♂
4. Marinelli ♂
5. Camillo Rota ♂
6. Claudia ♀
7. Pirro ♂
8. Odoardo ♂
9. Angelo ♂
10. Emilia ♀
11. Appiani ♂
12. Battista ♂
13. Orsina ♀

Fig. 18: Single play view, “Network” tab, for Lessing’s “Emilia Galotti”, https://dracor.org/id/ger000088

5.6.3.2 Tab “Relations”

The tab “Relations” visualizes kinship and other social relationship data, following the encoding scheme proposed in Wiedmer, Pagel, Reiter (2020). The tab is only displayed if this data is available in the response of the endpoint `/corpora/{corpusname}/play/{playname}`.[181] In the sidebar there is a box showing the cast of the play.[182]



This tab visualises kinship and other relationship data, following the encoding scheme proposed in Wiedmer, Pagel, Reiter 2020.

Cast list
(in order of appearance)

1. Der Prinz ♂
2. Der Kammerdiener ♂
3. Conti ♂
4. Marinelli ♂
5. Camillo Rota ♂
6. Claudia ♀
7. Pirro ♂
8. Odoardo ♂
9. Angelo ♂
10. Emilia ♀
11. Appiani ♂
12. Battista ♂
13. Orsina ♀

Fig. 19: Single play view, "Relations" tab, for Lessing's "Emilia Galotti", https://dracor.org/id/ger000088

5.6.3.3 Tab "Speech Distribution"

The tab "Speech Distribution" offers visualizations[183] either of the distribution of speaking characters or of their speeches per segment of a play. The line charts are inspired by three

[181] The returned data of the play "Emilia Galotti" can be accessed at https://dracor.org/api/corpora/ger/play/lessing-emilia-galotti. The data is retrieved by the function `fetchPlay` in "Play.js" already the "Play page" is loaded (see footnote 175) and passed to the component "RelationsGraph" <105>, which uses the library "SigmaJS" to render the characters in "cast" as nodes and the data from "relations" as the edges of a graph. The settings and the options of the force-directed layout which uses the algorithm "ForceAtlas2" (cf. footnote 178) are set in the code of the component <106> as are to colors of the rendered relation types <107> .

[182] The component "CastList" (cf. footnote 181) is used.

[183] In the GitHub repository the React components are contained in the folder /src/components/SpeechDistribution <108> . The charts are made using the library "react-chartjs-2" (https://react-chartjs-2.js.org), an implementation of *Chart.js* (https://www.chartjs.org) for the react

research papers on quantitative drama analysis (Sapogov 1974; Yarkho 2019 [1935–1938]; Trilcke, Fischer 2017). The line charts are rendered in the front-end based on the data of a single play returned from the `/corpora/{corpusname}/play/{playname}` endpoint of the API.

The tab "Speech Distribution" is a prototype in a special way. Conceptually, research-driven microservices are usually implemented within the DraCor ecosystem as dedicated programs, for example in distinct Python scripts[184] or in separate (also Python-based) Jupyter Notebooks.[185] However, the tab "Speech distribution" is an experimental attempt to implement a research-driven microservice in the frontend which in this case actually performs not only the visualization, but also most of the calculations. We discuss the implementation as well as the research on which this research-driven microservice is based in the following excursus.

Excursus: On the Research-Driven Diagrams in the "Speech Distribution" Tab

Sapogov 1974

The chart which is initially displayed when opening the tab "Speech Distribution" shows a visualization that is based on the article "Some Characteristics of the Dramaturgical Structure of A. N. Ostrovsky's Comedy 'The Forest'"[186] by Russian literary critic Vyacheslav Sapogov (1939–1996).[187]

Sapogov's basic assumption of his analysis of Ostrovsky's play is that a "neutral observer", who even though she or he does not understand the text, is nevertheless able to distinguish sequences of the play that are delimited by a character entering or exiting the stage.[188] These

framework. The code is available on GitHub: https://github.com/reactchartjs/react-chartjs-2. The data is displayed as line charts, see the documentation of the component: https://react-chartjs-2.js.org/components/line.

[184] Such as the "DraCor Metrics Service", see GitHub repository https://github.com/dracor-org/dracor-metrics.

[185] Such as the Notebook "To catch a protagonist in DraCor" by Ingo Börner, see GitHub repository https://github.com/dracor-org/dracor-notebooks/tree/main/catch-a-protagonist-in-dracor

[186] The original title is: "Некоторые характеристики драматургического построения комедии А. Н. Островского 'ЛЕС'".

[187] Sapogov's study (Sapogov 1974) has not been translated to English. For a short summary and discussion of his approach in English see the dissertation of Inna Wendell (2021: 34–36). She points out that Sapogov applies the method developed in an earlier russian study by Revzina and Revzin (1971), which is grounded on the work of Solomon Marcus (1970; German edition 1973) (for a discussion of the Revzin's approach c.f. Wendell 2021: 31–34). The actual plotting of the density is first found in Sapogov's work (c.f. Wendell 2021:35). Hence when discussing the DraCor platform, Wendell has a fair point in noting "The DraCor project refers to this plot as 'speech distribution' (as described in Sapogov) but it would be fair to give credit to the Revzins and also to characterize the plot using their terminology, i.e., scene density." (Wendell 2021: 63, fn 68).

[188] Sapogov is clearly following here the considerations of Solomon Marcus (1973: 289–293). Sapogov (as also Yarkho, see next section) seem to have an understanding of "scene", which doesn't necessarily take into consideration the boundaries as marked by paratextual or typographical features (e.g. headings, i.e. "Scene 1"), but solely look at the entering and exiting of characters (probably marked by stage

configurations are recorded in a table (cf. Tab. 07) with the characters as rows and the sequences as columns in which the presence of a character in a segment is marked as “1”, absence “0”.[189] The first part of Sapogov’s “protocol table” representing the first act of “The Forest” looks as such:

	I действие [1st act]							
	1	2	3	4	5	6	7	8
Г [G]	0	0	0	1	1	1	1	1
А [A]	1	1	0	0	0	0	1	0
М [M]	0	0	0	1	1	0	0	0
Б [B]	0	0	0	1	1	0	0	0
В [V]	0	0	0	0	1	1	0	0
П [P]	0	0	0	0	1	1	0	0
Бу [Bu]	0	1	1	1	0	0	0	0
К [K]	1	1	1	1	0	0	1	0
У [U]	0	1	1	0	0	0	1	1
Н [N]	0	0	0	0	0	0	0	0

directions) – “appearances”. However, in the table he seems to rely on the explicit marked boundaries in the text: In the first scene of the first act the characters Karp [K] and Aksyusha [A] are on stage. The end of the first scene is marked by the entrance of the character Bulanov [Bu]. The last stage direction of scene 1 reads “*Аксюша смотрит в окно, Буланов входит.*” [A. looks out of the window, Bu. enters]. In the text follows a title marking the start of scene 2 which starts with a stage direction listing the characters “*Аксюша, Буланов, Карп, потом Улита.*” [A., Bu., K., later U.]. In the table all mentioned characters are marked as present, although the character Ulita [U] enters only later in the scene, which is marked by the stage direction “*Входит Улита и чего-то ищет.*” [U. enters looking for someone.] The “neutral” observer would have split the second scene into two segments, the first (which would be the overall second segment) featuring the characters A., Bu. and K, while the overall third segment would have the characters A., Bu, K. and U. – In DraCor corpora boundaries of segments (scenes and acts) are motivated by typographical and paratextual features and are explicitly encoded using TEI-XML `<div>`s. Because Sapogov does not seem to identify the segment boundaries as his “neutral observer”, the visualization on the DraCor platform matches Sapogov’s chart.

[189] The actual “protocol” table (“таблица-протокол”, Sapogov 1971:60), a configuration matrix, was not included in the edition that was accessible in the Russian National Library (call number: Б-74-13/348). We were able to obtain a photograph of the table. Because of its size it was printed separately on a larger sheet and included with the publication. Sapogov notes in the text proper, that he included all characters of the play in this table, but “Teren’ka” (Теренька, мальчик Восмибратова), which has only one appearance at the beginning of the second act.

C [S]	0	0	0	0	0	0	0	0
Пл. я[190]	2	4	3	5	5	3	4	2

Tab. 07: “Protocol table” showing configurations in Ostrovsky's “The Forest” (cf. Sapogov 1974)

Based on this table Sapogov calculates, what he calls, the “density of acts”, which is calculated by summing up the cells of a segment containing “1” divided by the total number of cells representing a segment. Sapogov reports the density of the acts of the analyzed play as follows: I – 28/88 = 0,32; II – 4/22 = 0,18; III – 43/132 = 0,32; IV – 18/88 = 0,2; V – 32/99 = 0,32. The density of the whole play “The Forest” is 123 / 429 = 0,28.

In his study, Sapogov refers to the line chart “Table 2” (see Fig. 20 below), which, as he states, shows “the change of the density per scene”[191], even though the actual chart does not contain the density normalized as explained and calculated by him in the text proper for the acts, but the total number of characters per scene. The visualization in the front-end of DraCor (see Fig. 21) also only shows the number of characters per segment[192]:

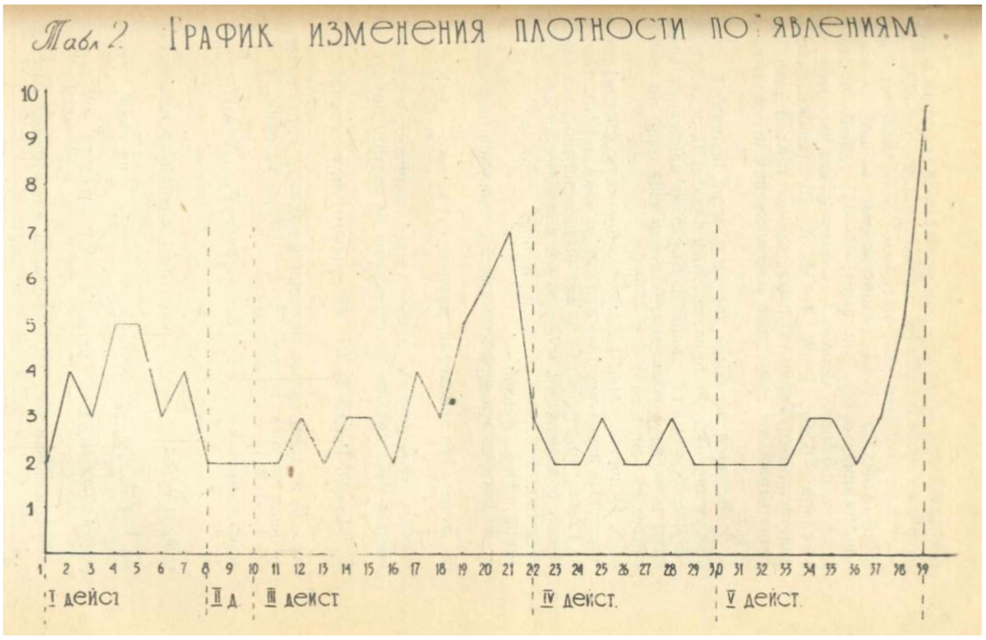


Fig. 20: Line chart, showing “the change of density per scene” (Sapogov 1971: 63)

[190] “плотность явления” [density of scene].

[191] “[...] таблица 2 представляет собой график изменения плотности по отдельным явлениям.” (Sapogov 1971: 61).

[192] The visualization uses the component “Sapogov” in “/components/SpeechDistribution/Sapogov.js” <109>. The data which is retrieved when the “Play page” is loaded (see footnote 175) is passed to the component. The speakers in each segment are counted <110> and visualized using the component “Line”.

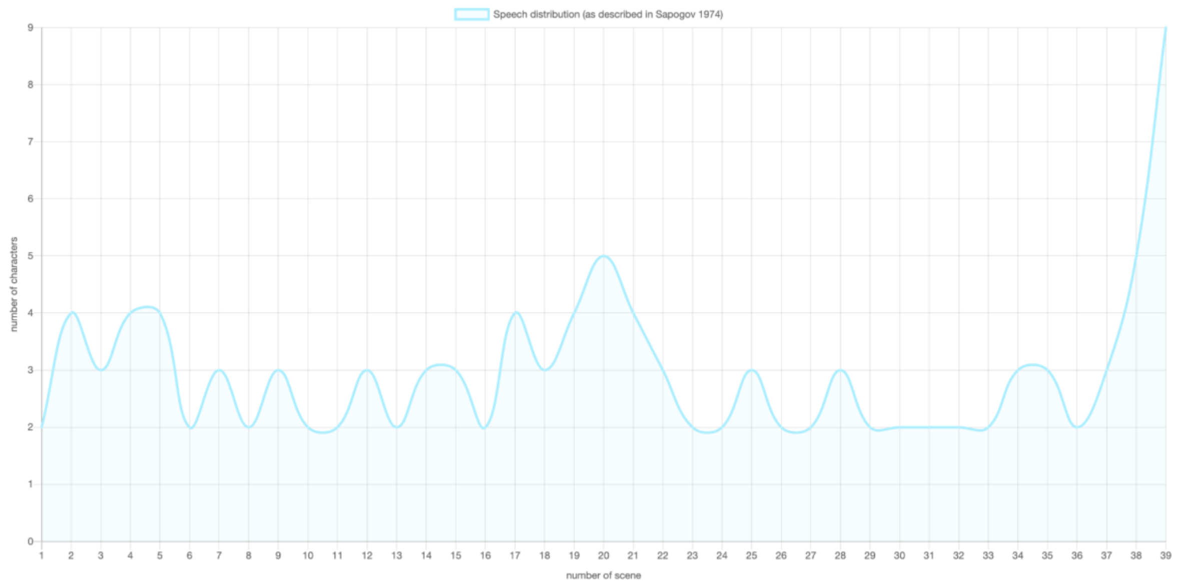


Fig. 21: DraCor chart showing "speech distribution" following Sapogov 1974, https://dracor.org/rus/ostrovsky-les#speech

Yarkho 1997 [2019]

The second visualization[193] is based on the article "Speech Distribution in Five-Act Tragedies (A Question of Classicism and Romanticism)"[194] written by the Russian Formalist Boris Yarkho (1889–1942). Although Yarkho's methodology originated by far in the pre-computer area–the first edition of the article was finished between late 1928 and early 1929–it virtually prompts repetition studies with digital technology today. A detailed discussion of Yarkho's method, an implementation and actualization using up-to-date statistical methods in Python can be found in the dissertation of Inna Wendell, in which Yarkho's method is applied for a study of Russian and French five-act comedies in verse (Wendell 2021; Discussion of Yarkho 1997 cf. Wendell 2021: 43–53).[195] In short, Yarkho assumes that a feature that allows to distinguish between five-act

[193] The visualization uses the component "Yarkho" in "/components/SpeechDistribution/Yarkho.js" <111>. The data which is retrieved when the "Play page" is loaded (see footnote 175) is passed to the component. The function uses two objects ("yarkho" and "nonGroups") to hold the data for two lines in the chart. The number of speakers is used as the keys of the object. While iterating over the segments of the API response data, the function counts the speakers of a segment and then increments a counter variable in the field of the object identified by the speaker-count <112>. The data is visualized using the component "Line".

[194] The original title is: "Распределение речи в пятиактной трагедии (К вопросу о классицизме и романтизме)". The article in Russian can be accessed at https://rvb.ru/philologica/04/04iarxo.htm.

[195] Wendell developed a Python library called "Player" (https://github.com/innawendell/player), that allows for the extraction of the features used for the analysis from PLAINTEXT and TEI-XML files (amongst them 10 comedies from RusDraCor), see <113>. The data and her analyses as notebooks can be found on GitHub as well: https://github.com/innawendell/European_Comedy.

tragedies of classicism and romanticism, is the distribution of speech or dialogue types, which are classified by the number of speaking characters (monologues, dialogues, polylogues) and then counted. It is important to note that Yarkho has a special definition of a segment or scene:

> The start and finish of a scene is, in principle, determined by a narrowing or widening of potential speech distribution, i. e. for the most part, with a character's entrance or exit from the stage. If a character starts talking from off-stage, their appearance is counted from the moment their voice is heard. The entrance or exit of a silent character also sets the bounds of a scene; only if the character (for example, an army cohort) passes upstage, without stopping, do we not consider this a change of scene. The death of a character signifies a potential narrowing of speech distribution, and therefore demarcates a scene. [...] A character may either be an individual or a collective. If the People, an army cohort, etc. ... speak with one voice [...] then they are counted as one character. [...] If the crowd is divided with a precise indication of groups (for example in Grillparzer: “Einige” ... “Andere” ...), each group counts as one character. When »All« are speaking in one voice, “All” are not considered an individual character if they have spoken separately in the same scene; but if at least one of the participants has not spoken separately, “All” are counted as an additional character. (Yarkho 2019: 21f.)

This type of segmentation is not available in the DraCor corpora, thus, in the case of DraCor, the visualization must not be understood as a re-implementation Yarko's method, but could be regarded as inspired by Yarkho's plots, of which there are only two in the article. For example, in Fig. 22 (see below) Yarkho determined percentage shares of segments by the number of speaking characters of Shakespearean tragedies that are compared to a certain subcorpus of tragedies of romanticism and visualized them as a line chart:

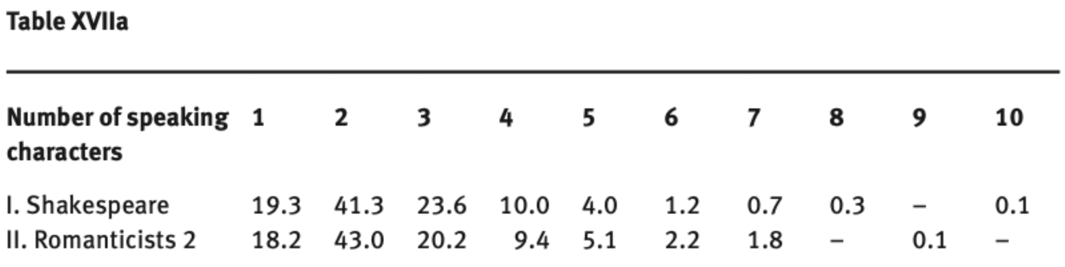


Table XVIIa

Number of speaking characters	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
I. Shakespeare	19.3	41.3	23.6	10.0	4.0	1.2	0.7	0.3	–	0.1
II. Romanticists 2	18.2	43.0	20.2	9.4	5.1	2.2	1.8	–	0.1	–

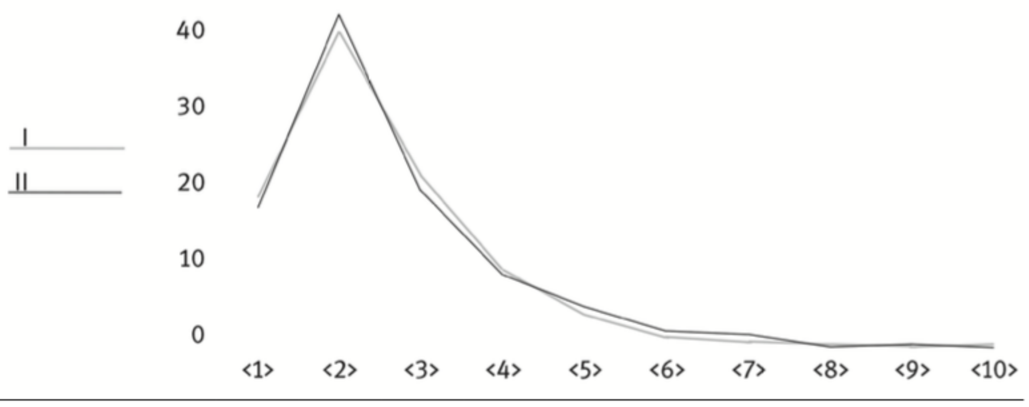
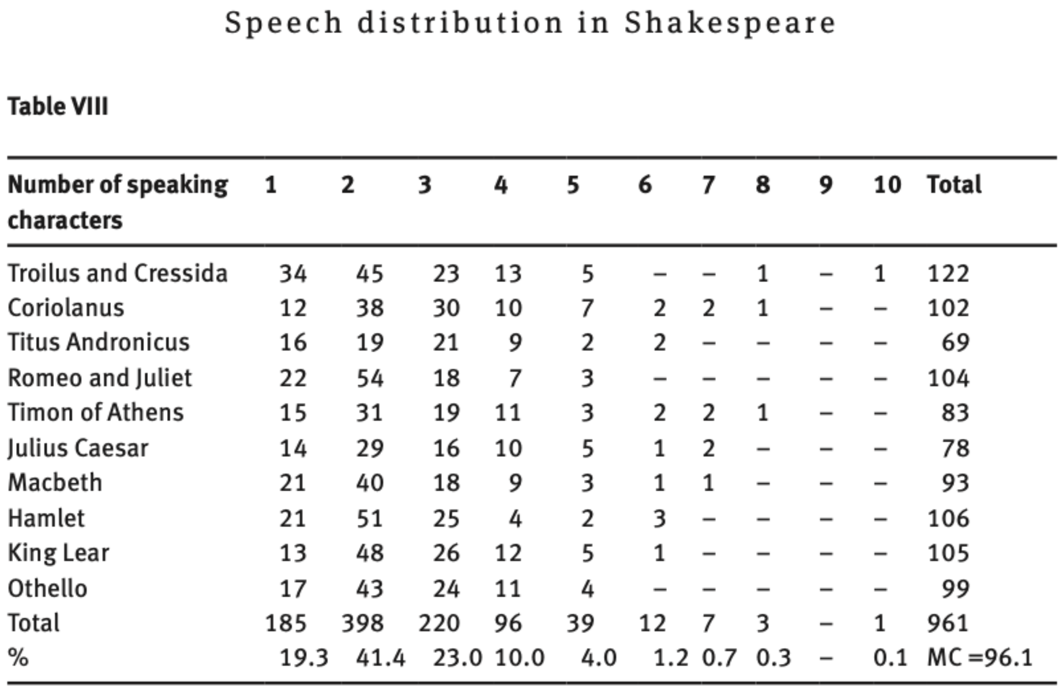


Fig. 22: Chart showing the number of speaking characters (Yarkho 2019:44), for the russian version see *https://rvb.ru/philologica/04/04iarxo_t08.htm*

The visualization that is shown on the DraCor front-end should only be used to compare plays from DraCor corpora with each other. If compared for example to the data that is provided by Yarkho in his article the numbers will not match up, because of different ways of counting segments and speaking characters therein. An example can demonstrate that: In another table (see Fig. 23) Yarkho provides the actual raw numbers of the type of segments of Shakespeare's tragedies, amongst them "Hamlet" (in row 8).



Speech distribution in Shakespeare

Table VIII

Number of speaking characters	1	2	3	4	5	6	7	8	9	10	Total
Troilus and Cressida	34	45	23	13	5	–	–	1	–	1	122
Coriolanus	12	38	30	10	7	2	2	1	–	–	102
Titus Andronicus	16	19	21	9	2	2	–	–	–	–	69
Romeo and Juliet	22	54	18	7	3	–	–	–	–	–	104
Timon of Athens	15	31	19	11	3	2	2	1	–	–	83
Julius Caesar	14	29	16	10	5	1	2	–	–	–	78
Macbeth	21	40	18	9	3	1	1	–	–	–	93
Hamlet	21	51	25	4	2	3	–	–	–	–	106
King Lear	13	48	26	12	5	1	–	–	–	–	105
Othello	17	43	24	11	4	–	–	–	–	–	99
Total	185	398	220	96	39	12	7	3	–	1	961
%	19.3	41.4	23.0	10.0	4.0	1.2	0.7	0.3	–	0.1	MC =96.1

Fig. 23: Speech distribution in Shakespeare (Yarkho 2019: 34)

If we compare this to the respective DraCor visualization of "Hamlet" (https://dracor.org/id/shake000032) from the Shakespeare Drama Corpus (ShakeDraCor), we can clearly see the mismatch already when looking at the number of monologues: Yarkho identified 21 segments with a single speaker, whereas in DraCor only one non-group character is delivering a single monologue:

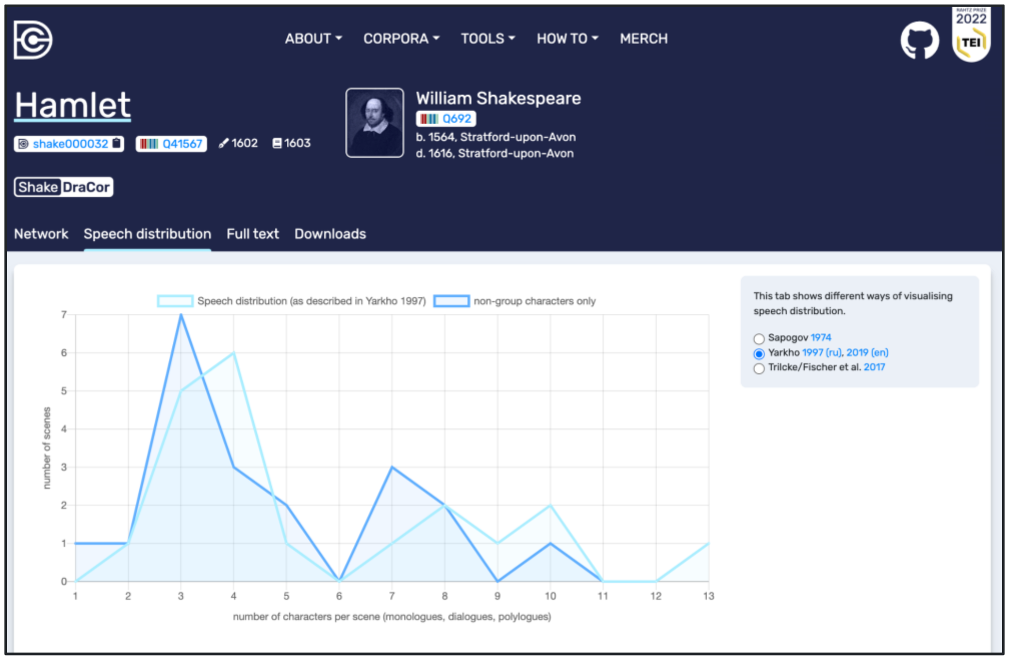


Fig. 24: DraCor chart showing speech distribution following Yarkho 1997

In her study Wendell uses data from the Russian Drama Corpus (RusDraCor), but adds additional markup to mark the boundaries of "scenes" in the sense of Yarkho by splitting up `<div>`s and attaching an additional attribute `@type` with the value `extra_scene`.[196] While simply splitting the respective `<div>`s is a pragmatic solution, in corpora, that are used not solely for this research purpose a less invasive encoding solution would have to be found, e.g. using the TEI element `<milestone>` with an attribute `@unit` to mark the boundaries of appearances.

Wendell also includes a series of visualizations but uses bar plots to show the speech distribution. Below is an example of her plots:

[196] See documentation of the additional tags introduced by Wendell: <114>. The xml extractor functions for RusDraCor files in her Python library "Player" <115> operate on the adapted markup. An example of a scene that is split after a character leaves the stage can be found at <116>.

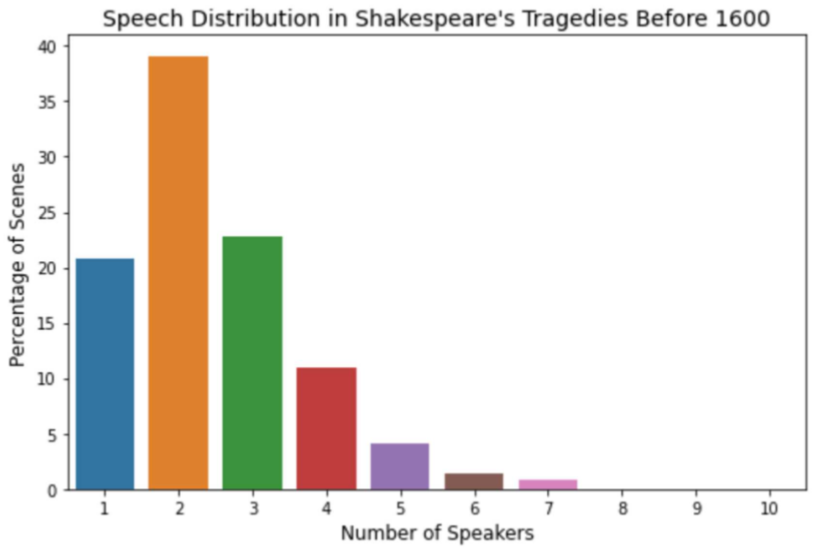
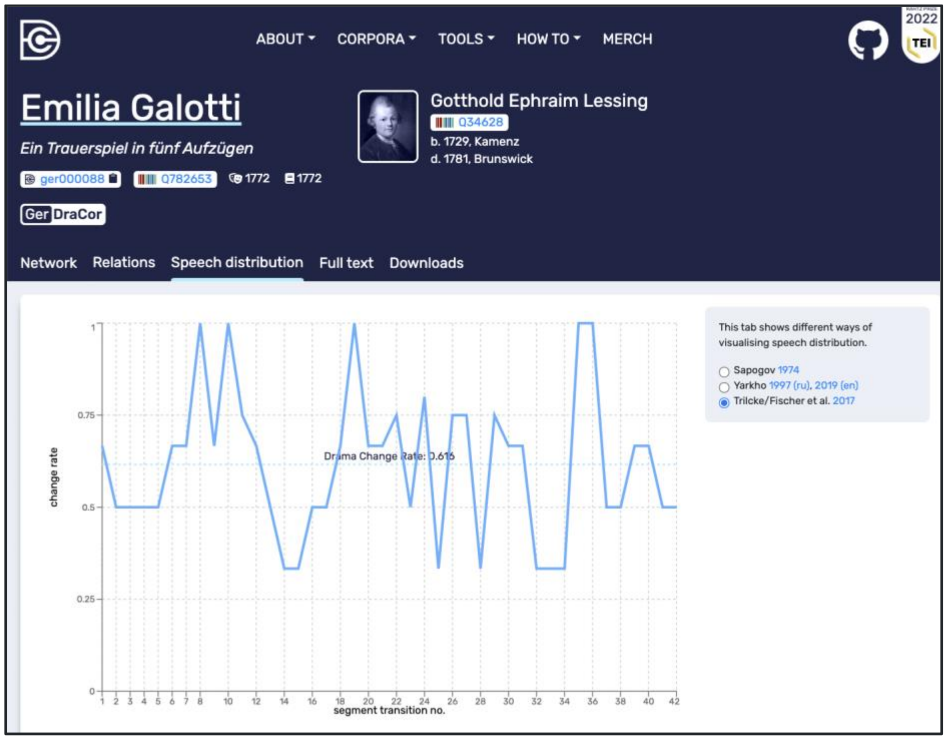


Fig. 25: Speech distribution in Shakespeare's tragedies before 1600 following Wendell (Wendell 2021: 342)

Trilcke, Fischer et al. 2017

The third chart (cf. Fig. 26) can be used to assess the dynamic of a play. It allows for identifying highly dynamic plays with a frequent alternation of characters on stage, which would be reflected in a line that often swings with high amplitudes, or low-dynamic plays with only small changes taking place between scenes, which becomes visible in a more or less straight running line.



This tab shows different ways of visualising speech distribution.

- ○ Sapogov 1974
- ○ Yarkho 1997 (ru), 2019 (en)
- ● Trilcke/Fischer et al. 2017

Fig. 26: DraCor chart showing drama change rate following Trilcke, Fischer et al. 2017

The metric underlying the visualization is the so-called "drama change rate". It was first implemented in the Tool "Dramavis"[197] and is described in the paper "Network Dynamics, Plot Analysis: Approaching Progressive Structuration of Literary Texts" (Trilcke, Fischer et al. 2017). The algorithm calculates an edit distance similar to the Levenshtein Distance[198]. For each segment of a play the cast of a scene (only speaking characters, though) is compared to the following segment. The operations that are necessary to get from the first state to the latter (add a character, remove a character) are summed up, resulting in a metric, that is called the "segment change rate", which can be normalized by dividing the resulting number through the number of all characters present in the consecutive segments ("normalized segment change rate"). To obtain a metric for the entire play, the drama change rate, the sum of all segment change rates is calculated and divided by the number of all segment change rates. The progression of the segment change rate can be represented in a chart (see Fig. 26).[199]

[197] The Python tool "Dramavis" (https://github.com/lehkost/dramavis) was developed in the predecessor project to DraCor "DLINA" (https://dlina.github.io) as a tool to analyze derived structural and network data from plays ("LINA" for "Literary Network Analysis" in a custom XML format, example of the play 'Emilia Galotti': https://dlina.github.io/linas/88/). The algorithm to calculate the drama change rate was added to "Dramavis" with this commit: <117>.

[198] See https://en.wikipedia.org/wiki/Levenshtein_distance. While the Levenshtein Distance is normally used to access the similarity of string, the described algorithm is a "graph edit distance".

[199] The visualization on the front-end uses the component "TrilckeFischer" <118>. The data is retrieved when the "Play page" is loaded (see footnote 175) and passed to the component. The function

The excursus exemplified an attempt to directly implement research-driven visualizations in the front-end. The goal of this experimental implementation in DraCor is also to pay more attention to the long but little-noticed tradition of quantitative analysis of dramas and thus, in a way, to highlight the antecedents of DraCor. Of course, as shown, this presupposes an in-depth understanding of the respective research articles.

We now return to the documentation of the less research-driven tabs in the Single Play view.

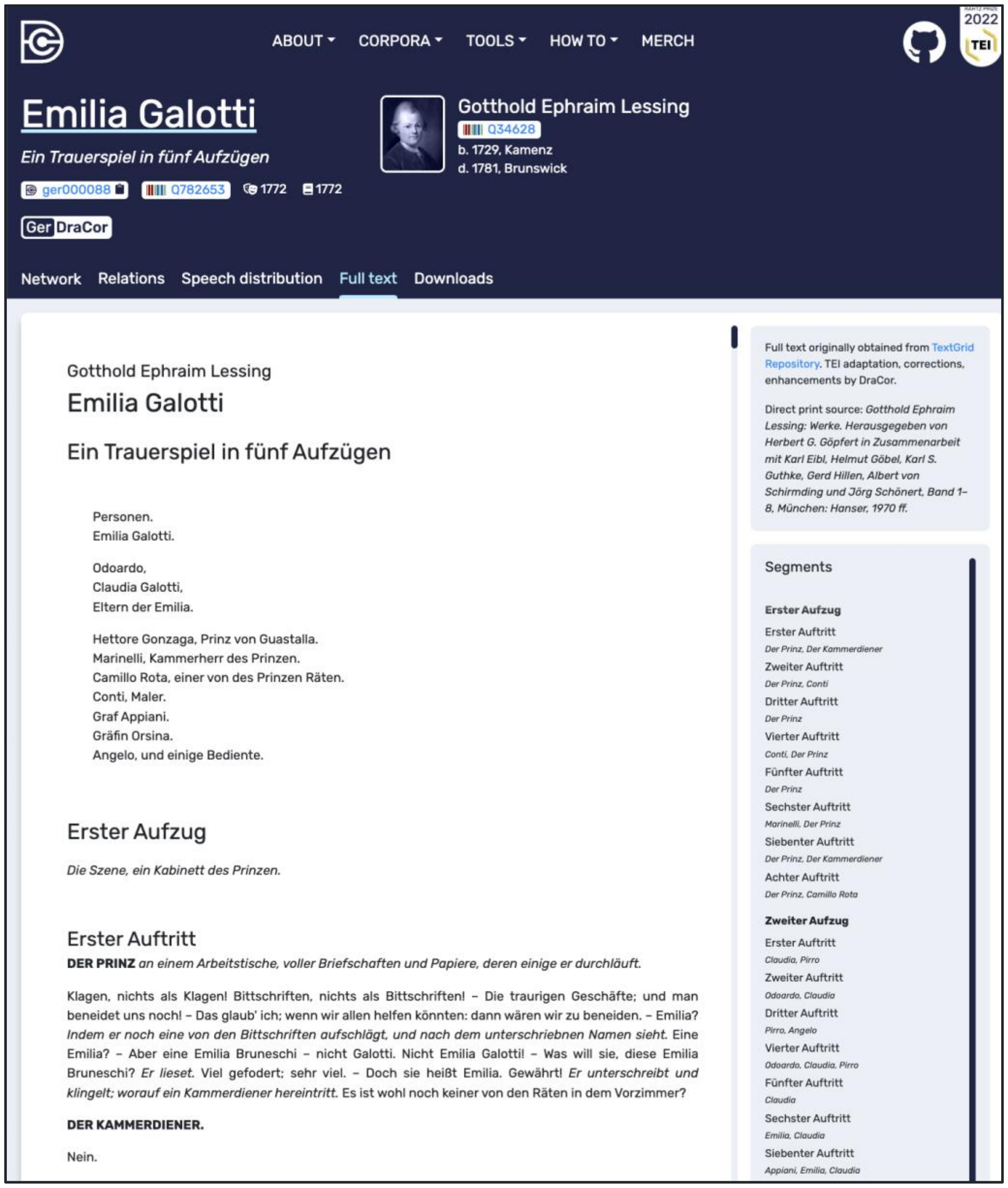
5.6.3.4 Tab "Full Text"

The tab "Full text" (see Fig. 27) allows for reading the text of a single play. The source TEI-XML is retrieved from the endpoint `/corpora/{corpusname}/play/{playname}/tei` and rendered using the library "CETEIcean"[200], which transforms TEI elements to custom HTML 5. The advantage of using this library is that there is no need for an additional transformation step (e.g., by using a custom XSLT to transform TEI-XML to HTML). The rich information from the source is kept.[201]

"calcChangeRates" <119> iterates over the segments and passes two consecutive segments to the function "calcSegmentChangeRate" <120>, which is then used in the rendering using the component "LineChart" <121>.

[200] Code repository on GitHub: https://github.com/TEIC/CETEIcean. For an introduction to the library see http://teic.github.io/CETEIcean/Balisage-CETEIcean.html and (Cayless and Viglianti 2018).

[201] The URL of the API endpoint is set in the component "Play.js" when the page of a single play is loaded <122>. This URL is passed in <123> to the component "TEIPanel" in the file "TEIPanel.js" <124>. This component renders the TEI-XML using the "CETEIcean" library.


ABOUT CORPORA TOOLS HOW TO MERCH

Emilia Galotti

Ein Trauerspiel in fünf Aufzügen

ger000088 Q782653 1772 1772

Ger DraCor

Gotthold Ephraim Lessing

Q34628

b. 1729, Kamenz

d. 1781, Brunswick

Network Relations Speech distribution Full text Downloads

Gotthold Ephraim Lessing

Emilia Galotti

Ein Trauerspiel in fünf Aufzügen

Personen.
Emilia Galotti.

Odoardo,
Claudia Galotti,
Eltern der Emilia.

Hettore Gonzaga, Prinz von Guastalla.
Marinelli, Kammerherr des Prinzen.
Camillo Rota, einer von des Prinzen Räten.
Conti, Maler.
Graf Appiani.
Gräfin Orsina.
Angelo, und einige Bediente.

Erster Aufzug

Die Szene, ein Kabinett des Prinzen.

Erster Auftritt

DER PRINZ *an einem Arbeitstische, voller Briefschaften und Papiere, deren einige er durchläuft.*

Klagen, nichts als Klagen! Bittschriften, nichts als Bittschriften! – Die traurigen Geschäfte; und man beneidet uns noch! – Das glaub' ich; wenn wir allen helfen könnten: dann wären wir zu beneiden. – Emilia? *Indem er noch eine von den Bittschriften aufschlägt, und nach dem unterschriebnen Namen sieht.* Eine Emilia? – Aber eine Emilia Bruneschi – nicht Galotti. Nicht Emilia Galotti! – Was will sie, diese Emilia Bruneschi? *Er lieset.* Viel gefodert; sehr viel. – Doch sie heißt Emilia. Gewährt! *Er unterschreibt und klingelt; worauf ein Kammerdiener hereintritt.* Es ist wohl noch keiner von den Räten in dem Vorzimmer?

DER KAMMERDIENER.

Nein.

Full text originally obtained from TextGrid Repository. TEI adaptation, corrections, enhancements by DraCor.

Direct print source: *Gotthold Ephraim Lessing: Werke. Herausgegeben von Herbert G. Göpfert in Zusammenarbeit mit Karl Eibl, Helmut Göbel, Karl S. Guthke, Gerd Hillen, Albert von Schirmding und Jörg Schönert, Band 1–8, München: Hanser, 1970 ff.*

Segments

Erster Aufzug
Erster Auftritt
Der Prinz, Der Kammerdiener
Zweiter Auftritt
Der Prinz, Conti
Dritter Auftritt
Der Prinz
Vierter Auftritt
Conti, Der Prinz
Fünfter Auftritt
Der Prinz
Sechster Auftritt
Marinelli, Der Prinz
Siebenter Auftritt
Der Prinz, Der Kammerdiener
Achter Auftritt
Der Prinz, Camillo Rota

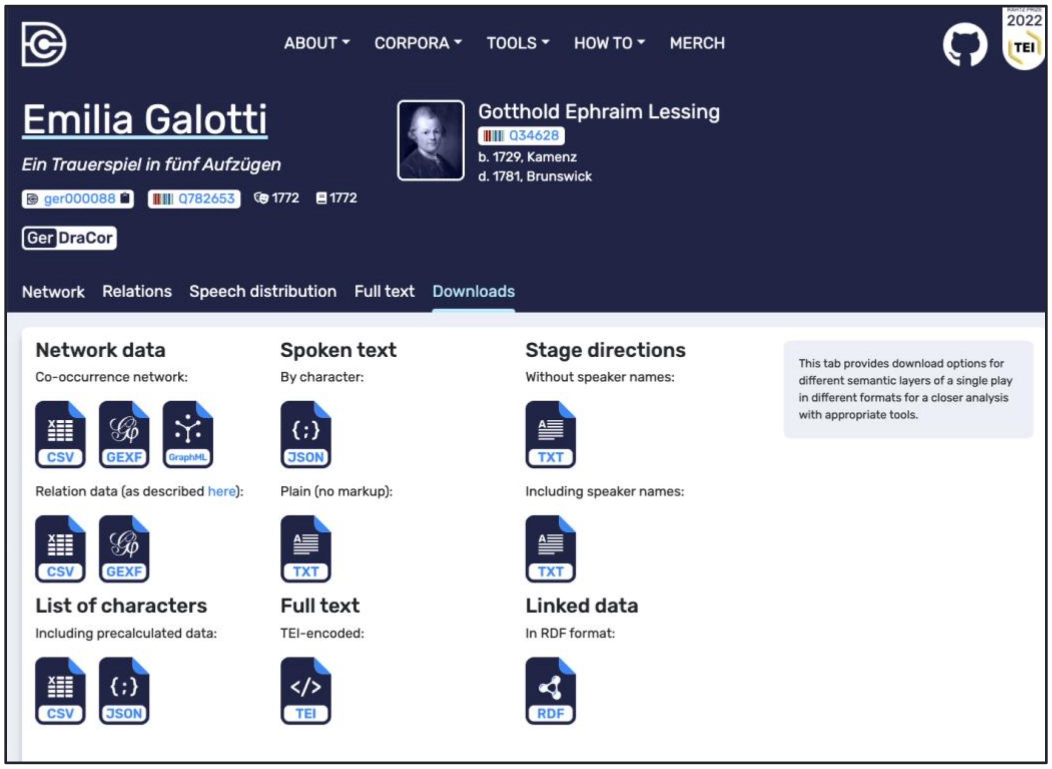
Zweiter Aufzug
Erster Auftritt
Claudia, Pirro
Zweiter Auftritt
Odoardo, Claudia
Dritter Auftritt
Pirro, Angelo
Vierter Auftritt
Odoardo, Claudia, Pirro
Fünfter Auftritt
Claudia
Sechster Auftritt
Emilia, Claudia
Siebenter Auftritt
Appiani, Emilia, Claudia

Fig. 27: Single play view, “Full text” tab, for Lessing’s “Emilia Galotti”, https://dracor.org/id/ger000088

The sidebar displays two info boxes alongside the text: The first info box provides information about the source of the enriched digital text that was used to produce the DraCor TEI-XML file and a bibliographic reference to the printed source.[202] Below the source box there is an I nfo box listing the segments and the characters speaking therein.[203]

5.6.3.5 Tab “Downloads”

The tab “Downloads” (Fig. 28) provides several serializations of semantic layers of a single play to be downloaded in several different formats.[204]



This tab provides download options for different semantic layers of a single play in different formats for a closer analysis with appropriate tools.

Fig. 28: Single play view, “Downloads” tab, for Lessing’s “Emilia Galotti”, https://dracor.org/id/ger000088

Technically this page provides links to the designated API endpoints. While manually navigating to an endpoint of the API–e.g. https://dracor.org/api/corpora/ger/play/lessing-emilia-galotti/tei – results in the web browser displaying the file, clicking on links in the “Download” tab triggers the file download behavior of the web browser: So, for example, the link to the TEI source of the play

[202] The info box is created using the component “SourceInfo” <125>. When the page of the play is loaded the component in “Play.js” passes the data that is retrieved from the endpoint (see footnote 175) to the “SourceInfo” component, which uses the information contained in “source” and “originalSource” <126> .
[203] The code generating the overview of the segments of a play is located in the file “Segments.tsx” <127> The function is written in TypeScript. It produces the segments structure of the play by evaluating “segments” and “cast” in the response data retrieved from the endpoint when the page of a single play is loaded (see footnote 175).
[204] The component “DownloadLinks” <128> is used to display the links which are defined inside the component <129>.

"Emilia Galotti" in the play's "Download" tab (https://dracor.org/ger/lessing-emilia-galotti#downloads) results in the browser downloading the file "ger000088-lessing-emilia-galotti.tei.xml".[205]

Network Data

Network data can be downloaded in three formats that are offered by the endpoints following the pattern

`/corpora/{corpusname}/play/{playname}/networkdata/[csv|gexf|graphml]`

A simple representation of the extracted network from a play is provided by the endpoint `/corpora/{corpusname}/play/{playname}/networkdata/csv` as CSV. The provided CSV file contains the data in the columns "Source", "Type", "Target" and "Weight".

The other two formats provided are based on XML: The GEXF file is fetched from the endpoint `/corpora/{corpusname}/play/{playname}/networkdata/gexf` and can be directly loaded into the network analysis software "Gephi"[206]. GraphML[207] is a popular format that is supported by a wide range of open-source network analysis tools. It is provided by the endpoint `/corpora/{corpusname}/play/{playname}/networkdata/graphml`

Data about social relations[208] amongst the characters can be downloaded if encoded in the source TEI of a play. This data can be obtained as CSV or GEXF[209] that are provided by the respective endpoints `/corpora/{corpusname}/play/{playname}/relations/csv` and `/corpora/{corpusname}/play/{playname}/relations/gexf`

List of Characters

Data on characters can be downloaded in CSV and JSON formats by providing the data that is returned by the endpoints[210] `/corpora/{corpusname}/play/{playname}/cast` (returns JSON) and `/corpora/{corpusname}/play/{playname}/cast/csv`.

[205] The file names contain the play ID, the playname and an optional slug of the semantic layer, e.g. "cast" separated by either dots or dashes: `{id}.{playname}[.|-]{layer slug}.{file format ending}` See for example `{id}-{playname}-cast.csv` <130>. The file names are defined in the component.
[206] Gephi: https://gephi.org; the GEXF file format is described at https://gexf.net.
[207] Specification of the GraphML format: http://graphml.graphdrawing.org
[208] Cf. section 5.6.3.2.
[209] The API offers this data additionally in GraphML format: Interactive documentation of the endpoint `/corpora/{corpusname}/play/{playname}/relations/graphml` see https://dracor.org/doc/api#/public/relations-graphml. There is an open issue on GitHub to fix this inconsistency: https://github.com/dracor-org/dracor-frontend/issues/245
[210] The interactive documentation of the endpoints can be accessed at https://dracor.org/doc/api#/public/get-cast and https://dracor.org/doc/api#/public/get-cast-csv.

The returned data includes information that is encoded in the source TEI, i.e. gender, but also derived text- or structure-based metrics thereof, i.e. the number of segments a character appears in (“numOfScenes”) or the number of speeches (“numOfSpeechActs”) and spoken words (“numOfWords”), as well as network-based metrics which are calculated with the help of the metrics service (“degree”, “weightedDegree”, “closeness”, “betweenness” and “eigenvector centrality”, see also section 5.4 on the “DraCor Metrics Service”).

Spoken Text

Files containing the spoken text per character are available for download: Linking to the endpoint `/corpora/{corpusname}/play/{playname}/spoken-text-by-character` the data can be downloaded in JSON format. The data is provided as an array containing objects representing the spoken text by a character.

The spoken text (excluding stage directions and speaker labels) as PLAINTEXT is made available via the endpoint[211] `/corpora/{corpusname}/play/{playname}/spoken-text`

Full Text

The source TEI-XML file can be downloaded. This is implemented by providing a link to the `/corpora/{corpusname}/play/{playname}/tei` endpoint of the API.[212]

Stage Directions

Stage directions can be downloaded as PLAINTEXT either including or excluding the speaker labels. The endpoints[213] are `/corpora/{corpusname}/play/{playname}/stage-directions` and `/corpora/{corpusname}/play/{playname}/stage-directions-with-speakers`

6. Prototyping APIs for CLS. Some Reflections and Two Additional API Experiments

In the previous section we reported on API prototyping in the context of the development of DraCor, our Programmable Corpora prototype. At this point, we want to briefly point out where

[211] The interactive documentation of the endpoints can be accessed at https://dracor.org/doc/api#/public/play-spoken-text-by-character and https://dracor.org/doc/api#/public/play-spoken-text.
[212] The interactive documentation of the endpoint can be accessed at https://dracor.org/doc/api#/public/play-tei.
[213] The interactive documentation of the endpoints can be accessed at https://dracor.org/doc/api#/public/play-stage-directions and https://dracor.org/doc/api#/public/play-stage-directions-with-speakers.

we see potentials of the idea of Programmable Corpora and thus of the design and exposure of APIs for literary corpora.

- **APIs can support the stabilization of epistemic objects in CLS.** While metadata formats or initiatives such as the "Text Encoding Initiative" continue to make a central contribution to the standardization of digital text objects, API endpoints help to reference specified aggregations as well as specified elements of research objects. For example, if you are researching the stage direction in Lessing's “Emilia Galotti”, your research object is addressable at `https://dracor.org/api/corpora/ger/play/lessing-emilia-galotti/stage-directions`
- **APIs can facilitate the standardization of (some) CLS workflows.** Much of the work in CLS is about (pre)processing data. In the operations that are implemented in APIs, such processing workflows can be standardized (which in turn leads to the stabilization of processed research objects). For example, if one is researching co-occurrence networks of plays, one can access (the output of) a standardized workflow for generating such a network via an API endpoint like `https://dracor.org/api/corpora/ger/play/lessing-emilia-galotti/networkdata/graphml`
- **APIs can simplify working with digital materials and applying digital methods in CLS.** APIs make it possible in principle to interact with data-enabled literary texts without having to resolve in detail issues such as the storage of data or the concrete technical implementation of algorithms for analysis. Thus, APIs can lower the technical barriers to digital work in CLS. For example, someone working on network analysis of plays can obtain essential network analysis metrics (in a CSV file that can be easily opened in e.g., Excel or OpenOffice Calc) for the plays in a corpus without any programming effort of their own by calling an endpoint like `https://dracor.org/api/corpora/ger/metadata/csv`
- **APIs can be possible solutions for working with copyright-protected material in CLS.** APIs also allow the underlying data to be hidden, so that you only get specifically transformed or derived representations of the data through the API. Thus, by means of an appropriately configured API, the concept of "Derived Text Formats" developed by Schöch et al. (2020) could in principle be implemented for text and data mining with copyrighted materials. In CLS work package 7, there will be a dedicated task (T7.5) in which we will experiment with styles of API design that could address such research scenarios.

DraCor as a prototype for Programmable Corpora—that is, corpora that expose an API to make texts machine-actionable—demonstrates these potentials of APIs for CLS research without already being a fully mature subsystem of a future infrastructural ecosystem for CLS. To conclude this report, and thus to conclude the first phase of Programmable Corpora prototyping in CLS

INFRA, let us briefly systematize the agile development of DraCor and especially of the DraCor API.

The––to once again quote Aaron Swartz—”natural growth” of the DraCor API out of our data, the drama corpora, is driven first by the structure of the data resp. the TEI documents (for example by the possibility of extracting specific information), second by research questions (such as the calculation of specific network metrics), third by front-end requirements and hence our approach to enable GUI-based low-technical accessibility and navigability of the data. In this sense, it can be said that the DraCor API, in its current state of implementation is

- document-based,
- research-driven,
- front-end-oriented.

This functional perspective on the API and the resulting functional differentiation of its endpoints can explain some of the dynamics in the “growth” of the API. At the same time, it emphasizes that the DraCor API is evolving in a specific environment of practices (such as conducting new studies or implementing new front-end functionalities), which also means that it must be understood as a specific realization of the idea of Programmable Corpora, one that has also become more specific as its growth has progressed.

It remains open at this point whether this increasing specificity of an API in the course of its "natural growth" in an environment of practices, as Swartz had in mind, can be avoided at all. One approach that will play a more important role in further work on the DraCor prototype, and possibly will lead to an increase in structure and system in the growth of the API (and maybe make it more generic), is ontology-based modeling of the data domain to which the API is dedicated. We will address these issues in detail in CLS INFRA Report D7.4, which we will publish in 2025 and which will focus on critical issues for the implementation of Programmable Corpora.

While the structuring of API development by aligning, on the one hand, the documentation of API functionalities and data returned in API responses with, on the other hand, domain-specific ontologies appears to be one strategy for future development, during the reporting phase, we experimented with yet another approach. This approach is based on the ideas of Distributed Text Services (DTS)[214] and the corresponding API specifications. DTS which is also TEI-based, offers the possibility to define standardized endpoints for APIs that bring a high degree of genericity. With the implementation of these DTS endpoints, which has already been done experimentally and is currently in internal review,[215] the individualization of the DraCor API in its specific ecosystem can be complemented by standardized endpoints.

[214] https://distributed-text-services.github.io/specifications; cf. Almas et al. 2023.
[215] https://github.com/dracor-org/dracor-api/pull/172

To develop a deeper understanding of design styles for APIs and to practically evaluate different approaches in the process of prototyping DraCor and the DraCor API, we also ran two API experiments, which we will report on in the following sections.

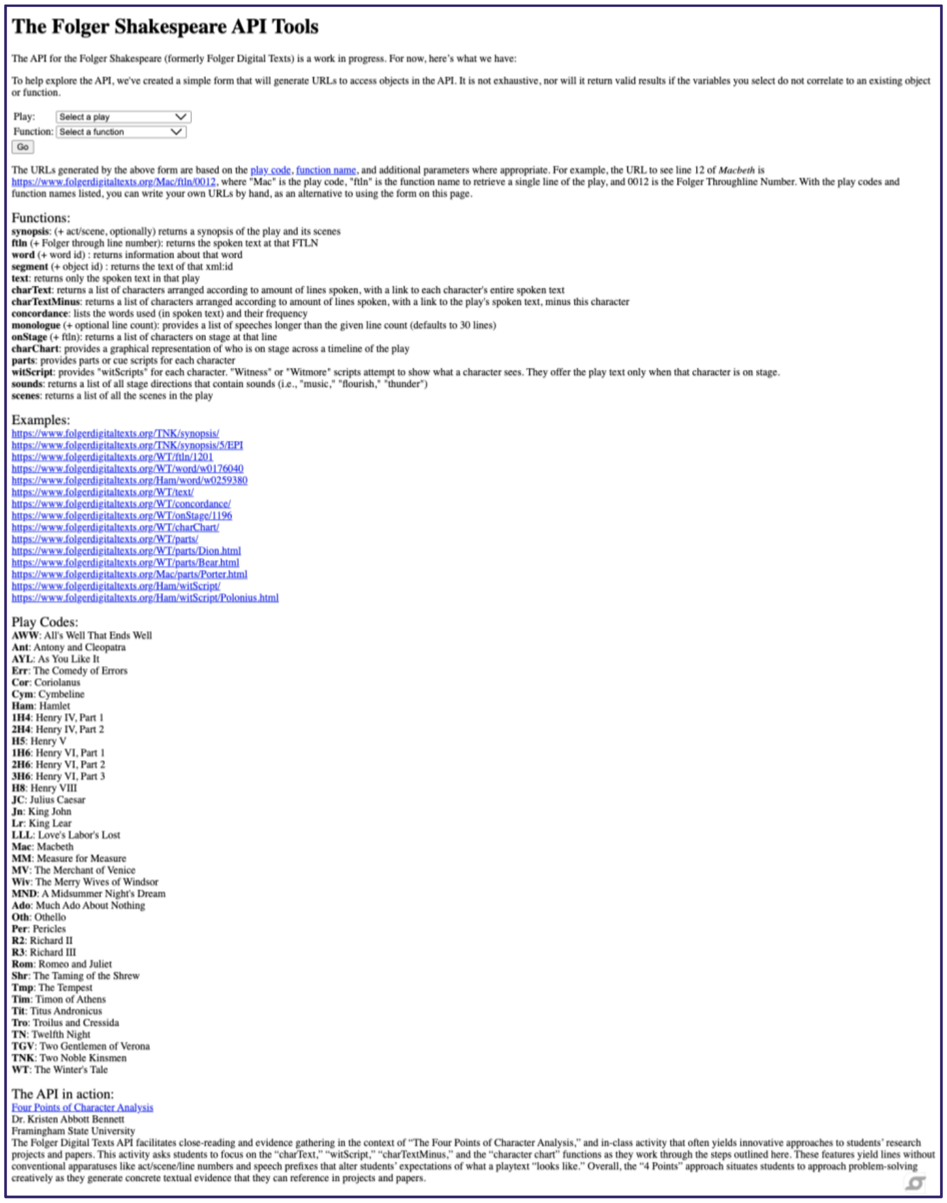
The first experiment (cf. section 6.1) is dedicated to one of the oldest APIs for literary corpora to our knowledge, the Shakespeare Folger API. Here we tried to, so to speak, "update" the API by using the already mentioned OpenAPI documentation (cf. section 5.3.2), thereby making it comparable to the DraCor API in selected (end)points. With the second experiment (cf. section 6.2), we focused on the POSTDATA ecosystem,[216] which is dedicated to providing and analyzing corpora of poems. Here we tried to develop a DraCor-like API that connects to the POSTDATA Knowledge Graph containing Linked Linguistic Open Data (Chiarcos et al. 2013) and complements the "PoetryLab API" provided by POSTDATA. We wanted to test whether the DraCor approach to design an API can be transferred to other corpora and/or digital ecosystems, i.e., whether our prototype can serve as a proof of concept for a CLS ecosystem connecting multiple components.

6.1 Relating APIs using OpenAPI: The Example of the Folger Shakespeare API

6.1.1 Discussion of the Folger Shakespeare API

In the following, we will discuss the API ("Folger Shakespeare API Tools", hence "Folger API")[217] of the Folger Shakespeare Online Library[218] which we consider an "early adopter" of adding an API to a corpus of literary texts, in this case the plays of William Shakespeare (Niles and Poston 2016). The URL of the HTML documentation of the Folger Shakespeare API Tools has been indexed by the Internet Archive's Wayback machine with the first crawl dating back to the year 2015[219].

[216] The project "POSTDATA – Poetry Standardization and Linked Open Data" received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 679528).
[217] https://www.folgerdigitaltexts.org/api
[218] https://shakespeare.folger.edu
[219] Cf. https://web.archive.org/web/20150410034823/https://www.folgerdigitaltexts.org/api. Since then the overall structure of the documentation has been unchanged, although extended. For example, the 2015 documentation lists only 10 functions ("ftln", "word", "segment", "text", "concordance", "monologue", "onStage", "charChart", "parts" and "witScript") and doesn't include the drop downs to build a query. The fact that the original endpoints are still included today with unchanged functionality and syntax shows that the API already has been stable for 7 years, even though the interface and the layout of the other parts of the Folger Shakespeare Library have evolved rapidly. This underpins the argument that APIs could foster the stability of research because, if implemented properly, they are quite robust.



The Folger Shakespeare API Tools

The API for the Folger Shakespeare (formerly Folger Digital Texts) is a work in progress. For now, here's what we have:

To help explore the API, we've created a simple form that will generate URLs to access objects in the API. It is not exhaustive, nor will it return valid results if the variables you select do not correlate to an existing object or function.

Play: Select a play
Function: Select a function
Go

The URLs generated by the above form are based on the play code, function name, and additional parameters where appropriate. For example, the URL to see line 12 of *Macbeth* is https://www.folgerdigitaltexts.org/Mac/ftln/0012, where "Mac" is the play code, "ftln" is the function name to retrieve a single line of the play, and 0012 is the Folger Throughline Number. With the play codes and function names listed, you can write your own URLs by hand, as an alternative to using the form on this page.

Functions:
synopsis: (+ act/scene, optionally) returns a synopsis of the play and its scenes
ftln (+ Folger through line number): returns the spoken text at that FTLN
word (+ word id) : returns information about that word
segment (+ object id) : returns the text of that xml:id
text: returns only the spoken text in that play
charText: returns a list of characters arranged according to amount of lines spoken, with a link to each character's entire spoken text
charTextMinus: returns a list of characters arranged according to amount of lines spoken, with a link to the play's spoken text, minus this character
concordance: lists the words used (in spoken text) and their frequency
monologue (+ optional line count): provides a list of speeches longer than the given line count (defaults to 30 lines)
onStage (+ ftln): returns a list of characters on stage at that line
charChart: provides a graphical representation of who is on stage across a timeline of the play
parts: provides parts or cue scripts for each character
witScript: provides "witScripts" for each character. "Witness" or "Witmore" scripts attempt to show what a character sees. They offer the play text only when that character is on stage.
sounds: returns a list of all stage directions that contain sounds (i.e., "music," "flourish," "thunder")
scenes: returns a list of all the scenes in the play

Examples:
https://www.folgerdigitaltexts.org/TNK/synopsis/
https://www.folgerdigitaltexts.org/TNK/synopsis/5/EPI
https://www.folgerdigitaltexts.org/WT/ftln/1201
https://www.folgerdigitaltexts.org/WT/word/w0176040
https://www.folgerdigitaltexts.org/Ham/word/w0259380
https://www.folgerdigitaltexts.org/WT/text/
https://www.folgerdigitaltexts.org/WT/concordance/
https://www.folgerdigitaltexts.org/WT/onStage/1196
https://www.folgerdigitaltexts.org/WT/charChart/
https://www.folgerdigitaltexts.org/WT/parts/
https://www.folgerdigitaltexts.org/WT/parts/Dion.html
https://www.folgerdigitaltexts.org/WT/parts/Bear.html
https://www.folgerdigitaltexts.org/Mac/parts/Porter.html
https://www.folgerdigitaltexts.org/Ham/witScript/
https://www.folgerdigitaltexts.org/Ham/witScript/Polonius.html

Play Codes:
AWW: All's Well That Ends Well
Ant: Antony and Cleopatra
AYL: As You Like It
Err: The Comedy of Errors
Cor: Coriolanus
Cym: Cymbeline
Ham: Hamlet
1H4: Henry IV, Part 1
2H4: Henry IV, Part 2
H5: Henry V
1H6: Henry VI, Part 1
2H6: Henry VI, Part 2
3H6: Henry VI, Part 3
H8: Henry VIII
JC: Julius Caesar
Jn: King John
Lr: King Lear
LLL: Love's Labor's Lost
Mac: Macbeth
MM: Measure for Measure
MV: The Merchant of Venice
Wiv: The Merry Wives of Windsor
MND: A Midsummer Night's Dream
Ado: Much Ado About Nothing
Oth: Othello
Per: Pericles
R2: Richard II
R3: Richard III
Rom: Romeo and Juliet
Shr: The Taming of the Shrew
Tmp: The Tempest
Tim: Timon of Athens
Tit: Titus Andronicus
Tro: Troilus and Cressida
TN: Twelfth Night
TGV: Two Gentlemen of Verona
TNK: Two Noble Kinsmen
WT: The Winter's Tale

The API in action:
Four Points of Character Analysis
Dr. Kristen Abbott Bennett
Framingham State University
The Folger Digital Texts API facilitates close-reading and evidence gathering in the context of "The Four Points of Character Analysis," and in-class activity that often yields innovative approaches to students' research projects and papers. This activity asks students to focus on the "charText," "witScript," "charTextMinus," and the "character chart" functions as they work through the steps outlined here. These features yield lines without conventional apparatuses like act/scene/line numbers and speech prefixes that alter students' expectations of what a playtext "looks like." Overall, the "4 Points" approach situates students to approach problem-solving creatively as they generate concrete textual evidence that they can reference in projects and papers.

Fig. 29: The Folger Shakespeare API Tools, https://www.folgerdigitaltexts.org/api

The book chapter "Mediating the Shakespeare User's Digital Experience" (Redick and Johnson 2022) gives a brief characterization of the "Folger API" and its usefulness for research:

> Finally, a small segment of the website is entirely focused on activities that require digital technology to perform: the Folger Shakespeare API suite [...]. [T]he API [...] enables features such as concordances and character charts, among others. With the concordance feature, *users can view* every word spoken in a play, and with the character charts, users can identify which characters are on stage and/or speaking, even if they are dead (as ghosts). [...] Much of the Folger Shakespeare site facilitates the most common activity (across media) related to the play texts: reading them. The ability to create links and contextual content across the site, such as audio recording annotations, downloadable files, and *API features that support data mining* makes the

> experience of reading the texts more flexible and usable for a wider audience than the printed books could manage. (Redick and Johnson 2022: 281; our emphases)

The “Folger API” is documented by a plain HTML page (cf. Fig. 29) which aims at making a user familiar with the functionality. It provides a basic introduction to the API and makes clear that it is “work in progress”. Users can start building queries by selecting a play and a function from two dropdown select fields or choose from 15 examples[220]. The documentation explains the pattern of how queries should be constructed by appending at least the parameters `play code` and `function name` in some cases followed by optional parameters to the base URL https://www.folgerdigitaltexts.org.

The play codes identifying the 38 Shakespearean plays are also listed in the documentation[221] and the following 15 endpoints are documented as follows:[222]

- `synopsis` (+ act/scene, optionally): returns a synopsis of the play and its scenes
- `ftln` (+ Folger through line number): returns the spoken text at that FTLN
- `word` (+ word id): returns information about that word
- `segment` (+ object id): returns the text of that xml:id
- `text`: returns only the spoken text in that play
- `charText`: returns a list of characters arranged according to amount of lines spoken, with a link to each character's entire spoken text
- `charTextMinus`: returns a list of characters arranged according to amount of lines spoken, with a link to the play's spoken text, minus this character
- `concordance`: lists the words used (in spoken text) and their frequency
- `monologue` (+ optional line count): provides a list of speeches longer than the given line count (defaults to 30 lines)
- `onStage` (+ ftln): returns a list of characters on stage at that line
- `charChart`: provides a graphical representation of who is on stage across a timeline of the play
- `parts`: provides parts or cue scripts for each character

[220] The documentation does not provide examples for all the listed endpoints: There are no examples for the endpoints `segment`, `charText`, `charTextMinus`, `monologue` and `sounds`.
[221] Almost every response of an endpoint includes a title of the play for which data is requested, but in the case of the playcodes “AWW”, “Err”, “LLL”, “MV”, “Wiv”, “MND”, “Shr”. and “WT” the title included in the documentation doesn’t match the title from the response, e.g. “The Winter’s Tale” vs. “Winter’s Tale”. This is not a problem for a human viewer who can easily understand which play is intended, but this inconsistency has to be taken into account when trying to match a list derived from the documentation and the titles included in the output in a program.
[222] Cf. Folger API Documentation: “Functions”, https://www.folgerdigitaltexts.org/api

- `witScript`: provides "witScripts" for each character. "Witness" or "Witmore" scripts attempt to show what a character sees. They offer the play text only when that character is on stage.
- `sounds`: returns a list of all stage directions that contain sounds (i.e., "music," "flourish," "thunder")
- `scenes`: returns a list of all the scenes in the play

While the original, simple documentation of the Folger API allows a human reader to easily get started using the functionality provided and analyzing the returned results by manual inspection, it must be said that the affordances of modern use of APIs in the paradigm of the “Programmable Web” have changed.

6.1.2 ‘Swaggerization’ of the Folger Shakespeare API

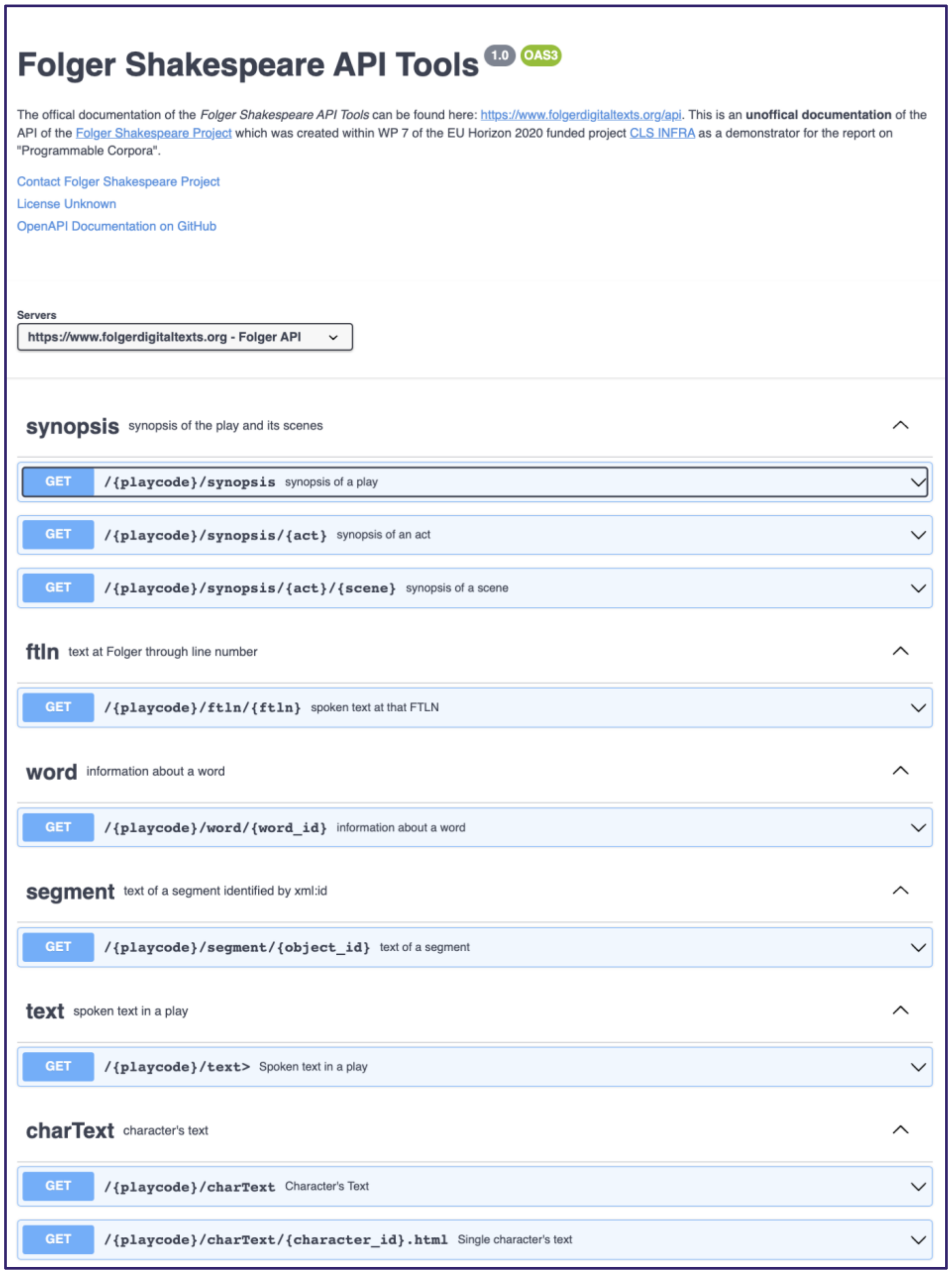
As a demonstrator of what can be done with state-of-the-art ways of API documentation and with the aim of making the Folger API comparable by using a standardized format of documentation, we re-documented the Folger API in the OpenAPI format.

The OpenAPI specification provides a standardized format for describing an API, including the available endpoints and the supported HTTP methods (GET, POST, …), the expected input and output for each endpoint, which can be described with schemas, and many other details. The OpenAPI specification is language agnostic, meaning it can be used to describe APIs written in any programming language. APIs described with OpenAPI are interoperable, because the documentation is machine-readable and can be easily understood and consumed by a wide range of tools, such as API development frameworks or documentation generators (e.g., Swagger UI). It is also possible to automatically generate client libraries for many programming languages from an OpenAPI specification.[223] The OpenAPI specification serves as a contract between the API provider and consumers, clearly defining what the API does and does not provide (for OpenAPI and DraCor see section 5.3.2).

When developing the prototype, we used the Python package “APISpec”[224] that allowed us to generate an OpenAPI conformant specification from the docstring annotations to functions in the Python code[225]. After we had generated the documentation as a YAML file[226], we used the Swagger Editor[227] which displays the documentation as an interactive website, from which queries

[223] https://swagger.io/tools/swagger-codegen.
[224] https://github.com/marshmallow-code/apispec; Documentation: https://apispec.readthedocs.io
[225] See code on GitHub: https://github.com/ingoboerner/folger-shakespeare-openapi. A fork of this repository is available at https://github.com/dh-network/folger-shakespeare-openapi. The re-documentation is done with the Jupyter Notebook “folger-shakespeare-api-doc.ipynb” <131>.
[226] Cf. <132>.
[227] https://editor.swagger.io

to the documented endpoints can be sent. We also used this editor to automatically generate a client (API wrapper) for the programming language Python, which drastically simplifies the task of querying the Folger API.



Folger Shakespeare API Tools 1.0 OAS3

The offical documentation of the *Folger Shakespeare API Tools* can be found here: https://www.folgerdigitaltexts.org/api. This is an **unoffical documentation** of the API of the Folger Shakespeare Project which was created within WP 7 of the EU Horizon 2020 funded project CLS INFRA as a demonstrator for the report on "Programmable Corpora".

Contact Folger Shakespeare Project
License Unknown
OpenAPI Documentation on GitHub

Servers
https://www.folgerdigitaltexts.org - Folger API

synopsis synopsis of the play and its scenes

GET `/{playcode}/synopsis` synopsis of a play
GET `/{playcode}/synopsis/{act}` synopsis of an act
GET `/{playcode}/synopsis/{act}/{scene}` synopsis of a scene

ftln text at Folger through line number

GET `/{playcode}/ftln/{ftln}` spoken text at that FTLN

word information about a word

GET `/{playcode}/word/{word_id}` information about a word

segment text of a segment identified by xml:id

GET `/{playcode}/segment/{object_id}` text of a segment

text spoken text in a play

GET `/{playcode}/text>` Spoken text in a play

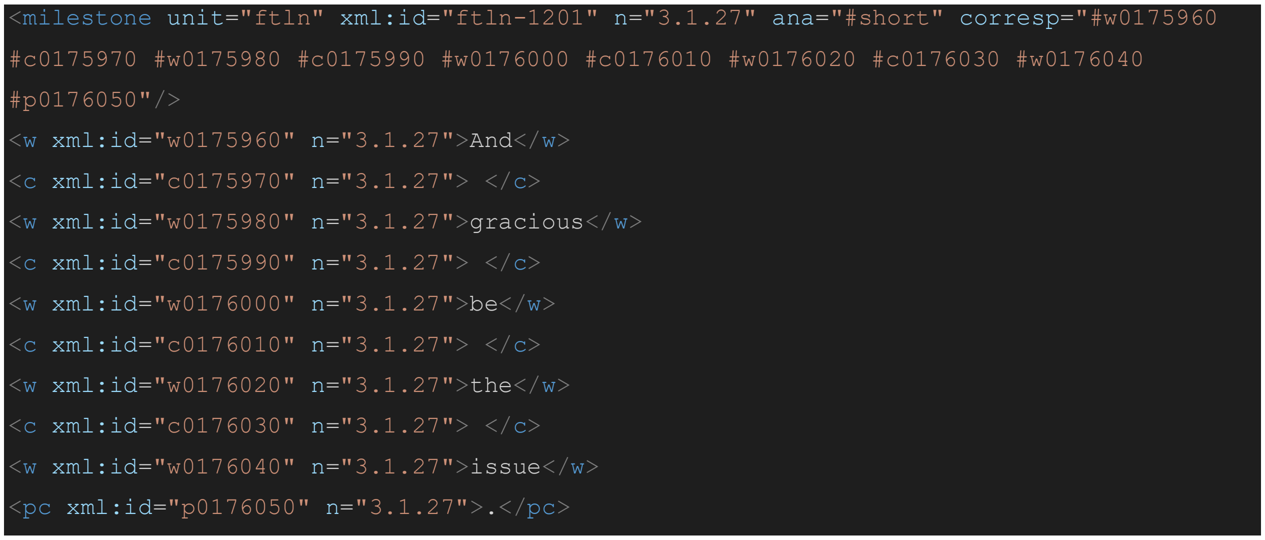
charText character's text

GET `/{playcode}/charText` Character's Text
GET `/{playcode}/charText/{character_id}.html` Single character's text

Fig. 30: OpenAPI specification rendered with SwaggerUI

Querying the API from the interactive Swagger documentation and from within Python scripts makes one thing obvious: the Folger API's features are geared more towards users reading the output in a web browser and not so much with practitioners of CLS in mind, that would not "view" the response of an endpoint but rather process it by using custom scripts. Exploiting the rich and useful features of the API is hindered by the decision of relying solely on HTML as the format of resource representation in the API's responses. If a user would not only want to "view" the result of a query to a specific endpoint but process it with a script she or he would have to first parse the response into some kind of data structure that could then be easily processed in the program. While it is evident that most users will consult the Folger Shakespeare Online Library with the intent of reading (or downloading) the texts, it would drastically enhance the usefulness of the Folger API suite if some sort of content negotiation would be implemented to allow for requesting other formats of resource representation, e.g., JSON right away.

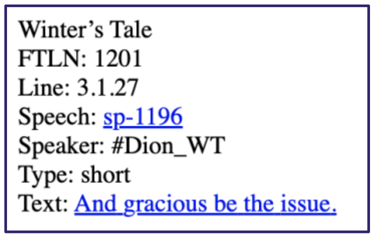
We will explain this with an example: The endpoint `ftln`[228] allows a user to retrieve some metadata and the textual content of a text line identified by the so-called "Folger Through Line Number" (FTLN), which is a concept that relates back to the printed Folger Shakespeare editions. The number is included in the underlying TEI-XML encoded play, which makes use of the TEI element `<milestone>` to mark the beginnings of these lines. In the example below we show the snippet of the TEI file[229] (Fig. 31) which is the basis of the data returned by the example of a query to this endpoint in the documentation "https://www.folgerdigitaltexts.org/WT/ftln/1201". The ID of the line identified by the FTLN of "1201" is included in the attribute `xml:id`. The `<milestone>` has other attributes of which `@n`, `@ana` and `@corresp` contain information that is included in the HTML response of the endpoint (see Fig. 32 and 33).



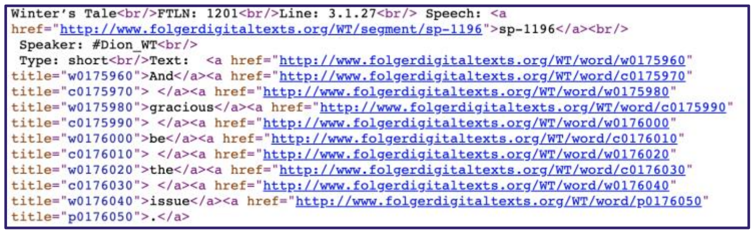
```
<milestone unit="ftln" xml:id="ftln-1201" n="3.1.27" ana="#short" corresp="#w0175960
#c0175970 #w0175980 #c0175990 #w0176000 #c0176010 #w0176020 #c0176030 #w0176040
#p0176050"/>
<w xml:id="w0175960" n="3.1.27">And</w>
<c xml:id="c0175970" n="3.1.27"> </c>
<w xml:id="w0175980" n="3.1.27">gracious</w>
<c xml:id="c0175990" n="3.1.27"> </c>
<w xml:id="w0176000" n="3.1.27">be</w>
<c xml:id="c0176010" n="3.1.27"> </c>
<w xml:id="w0176020" n="3.1.27">the</w>
<c xml:id="c0176030" n="3.1.27"> </c>
<w xml:id="w0176040" n="3.1.27">issue</w>
<pc xml:id="p0176050" n="3.1.27">.</pc>
```

[228] It is called `tln` in the original documentation, which is misleading, because it is actually `ftln` in the URL of the endpoint.
[229] We use the XML file of the play "The Winter's Tale" as an example. It can be retrieved here: https://shakespeare.folger.edu/downloads/xml/the-winters-tale_XML_FolgerShakespeare.zip

Fig. 31: Response of Folger's `ftln` endpoint



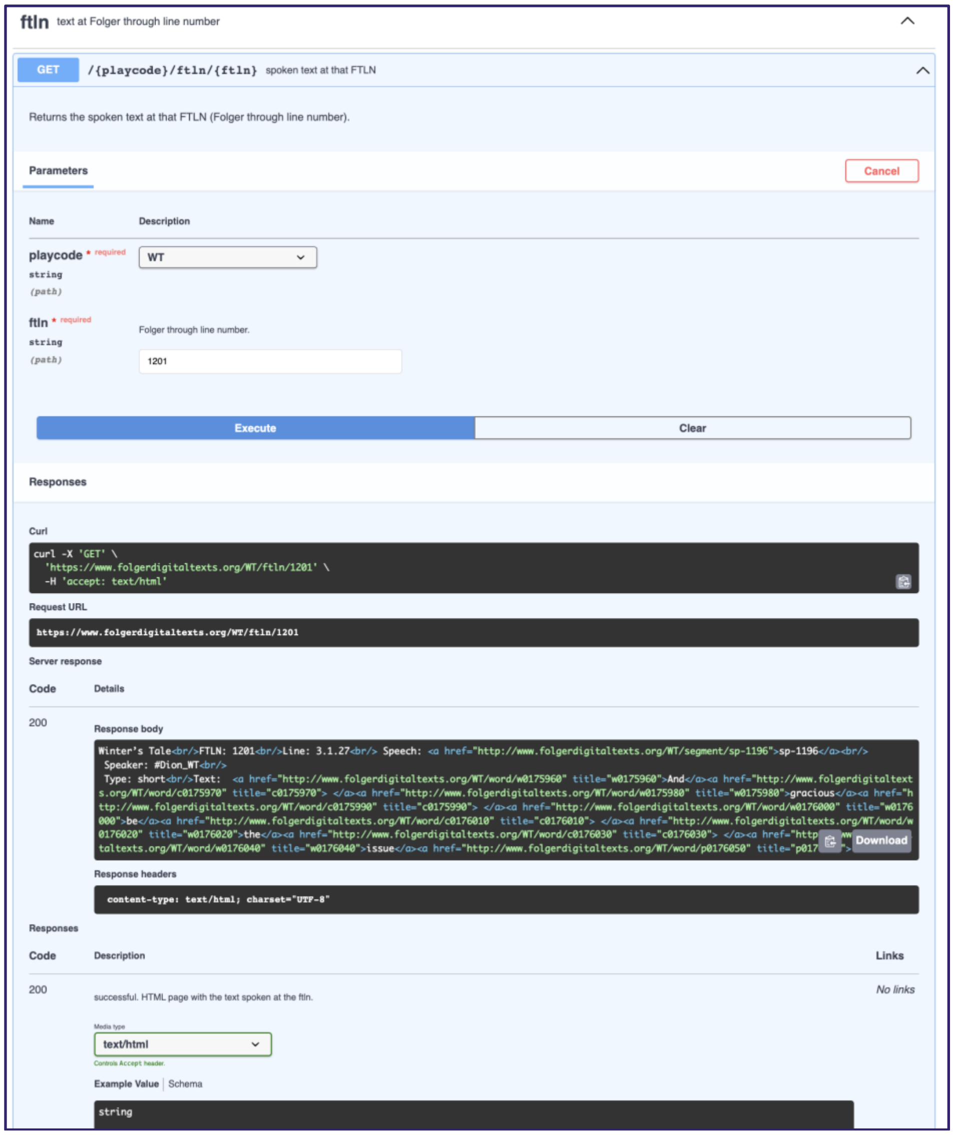
Winter's Tale
FTLN: 1201
Line: 3.1.27
Speech: sp-1196
Speaker: #Dion_WT
Type: short
Text: And gracious be the issue.



```
Winter's Tale<br/>FTLN: 1201<br/>Line: 3.1.27<br/> Speech: <a
href="http://www.folgerdigitaltexts.org/WT/segment/sp-1196">sp-1196</a><br/>
 Speaker: #Dion_WT<br/>
 Type: short<br/>Text:  <a href="http://www.folgerdigitaltexts.org/WT/word/w0175960"
title="w0175960">And</a><a href="http://www.folgerdigitaltexts.org/WT/word/c0175970"
title="c0175970"> </a><a href="http://www.folgerdigitaltexts.org/WT/word/w0175980"
title="w0175980">gracious</a><a href="http://www.folgerdigitaltexts.org/WT/word/c0175990"
title="c0175990"> </a><a href="http://www.folgerdigitaltexts.org/WT/word/w0176000"
title="w0176000">be</a><a href="http://www.folgerdigitaltexts.org/WT/word/c0176010"
title="c0176010"> </a><a href="http://www.folgerdigitaltexts.org/WT/word/w0176020"
title="w0176020">the</a><a href="http://www.folgerdigitaltexts.org/WT/word/c0176030"
title="c0176030"> </a><a href="http://www.folgerdigitaltexts.org/WT/word/w0176040"
title="w0176040">issue</a><a href="http://www.folgerdigitaltexts.org/WT/word/p0176050"
title="p0176050">.</a>
```

Fig. 32: Folger API response as displayed in the browser

Fig. 33: Underlying HTML source code for response in Fig. 32



ftln text at Folger through line number

GET `/{playcode}/ftln/{ftln}` spoken text at that FTLN

Returns the spoken text at that FTLN (Folger through line number).

Parameters

Name	Description
playcode * required string (path)	WT
ftln * required string (path)	Folger through line number. 1201

Responses

Curl

```
curl -X 'GET' \
  'https://www.folgerdigitaltexts.org/WT/ftln/1201' \
  -H 'accept: text/html'
```

Request URL

```
https://www.folgerdigitaltexts.org/WT/ftln/1201
```

Server response

Code	Details
200	Response body

```
Winter's Tale<br/>FTLN: 1201<br/>Line: 3.1.27<br/> Speech: <a href="http://www.folgerdigitaltexts.org/WT/segment/sp-1196">sp-1196</a><br/>
 Speaker: #Dion_WT<br/>
 Type: short<br/>Text:  <a href="http://www.folgerdigitaltexts.org/WT/word/w0175960" title="w0175960">And</a><a href="http://www.folgerdigitaltext
s.org/WT/word/c0175970" title="c0175970"> </a><a href="http://www.folgerdigitaltexts.org/WT/word/w0175980" title="w0175980">gracious</a><a href="h
ttp://www.folgerdigitaltexts.org/WT/word/c0175990" title="c0175990"> </a><a href="http://www.folgerdigitaltexts.org/WT/word/w0176000" title="w0176
000">be</a><a href="http://www.folgerdigitaltexts.org/WT/word/c0176010" title="c0176010"> </a><a href="http://www.folgerdigitaltexts.org/WT/word/w
0176020" title="w0176020">the</a><a href="http://www.folgerdigitaltexts.org/WT/word/c0176030" title="c0176030"> </a><a href="http
taltexts.org/WT/word/w0176040" title="w0176040">issue</a><a href="http://www.folgerdigitaltexts.org/WT/word/p0176050" title="p017
```

Response headers

```
content-type: text/html; charset="UTF-8"
```

Responses

Code	Description	Links
200	successful. HTML page with the text spoken at the ftln.	*No links*

Media type: text/html
Controls Accept header.

Example Value | Schema

```
string
```

Fig. 34: Query in the interactive documentation provided by the Swagger Editor

If we inspect the response returned from the endpoint (see Fig. 32 and 33) we can observe the following:

- the format of resource representation is HTML;
- only a snippet of a fully valid HTML document is returned[230];
- the information is organized in lines separated by the HTML element `<br>`;
- some of the lines contain the information as PLAINTEXT. Here the relevant value is prepended by a label followed by a colon, e.g., "Type: ". For others, the information is dispersed amongst HTML attributes of the element `<a>` and its textual content;
- in the response other endpoints are referenced, e.g., the `segment` and the `word` endpoints;
- from only one example, one cannot assume, that the output is always structured in the same way, e.g., the information on the speaker of the text that includes the requested line can be taken from the fifth line of the response; in the current example there is only one "Speaker", but, one can expect, that there are lines in a play that are spoken by more than one character;
- the response contains the title of the play only, not the play code that is used to identify the parent play in the context of the Folger API.

Based on this and other observations (inspection of multiple responses) a user wanting not only to "read" the response of the Folger API, but to process it algorithmically, would have to parse the response to extract the relevant information. As our endeavor of re-documenting the API has shown, it is quite easy to come up with a function that requests data from an endpoint, whereas developing a parser function that would transform the HTML response into a data structure native to Python is fairly more complex.

Overall, the Folger API can be classified as a research-driven API. The functionality provided by the various endpoints helps users to answer very specific questions relevant for dramatic texts, e.g., the endpoint `witScript` provides the text "when that character is on stage", or the endpoint `sounds` lists all stage directions that are classified as containing sounds.

Conversely, there are hardly any functions that could be labeled as "document-driven", except for the endpoint `segment` which allows for querying the text of a segment by its `xml:id`. This endpoint is referenced in the responses from other endpoints, e.g., `ftln` and `word` to allow for requesting the context of the given unit under investigation (see example of `ftln` above). Still, there is no discovery endpoint implemented that would allow for algorithmically retrieving all the

[230] The completeness of the HTML document varies from endpoint to endpoint, some also return not only snippets, but full-fledged HTML pages with `<html>`, `<head>` and `<body>` containing the result, e.g., `charText`.

IDs of segments of a play which are eligible in a query for an individual segment. Likewise, an answer to the question of how many segments there are in a given text, is almost impossible to get.

6.1.3 Mappings of ShakeDraCor and the Folger Shakespeare API

The Folger Shakespeare Corpus has also been ingested into the DraCor system as "ShakeDraCor"[231]. The original encoding of the `<text>` was kept, but the metadata in the `<teiHeader>` was adapted to seamlessly integrate with the DraCor system, especially to facilitate the extraction of character networks. With having both the Folger API operating on the corpus of the Folger Shakespeare Library and the DraCor API operating on the slightly adapted ShakeDraCor, this enables an interesting use case of being able to request data from those two APIs that implement different functionalities. It also allows us to contrast both approaches of designing APIs. Because both APIs are now documented in the same standardized format, we have a uniform way of addressing the endpoints by their "operationId" in the OpenAPI file[232].

folger_endpoint	**folger_operationId**	**dracor_endpoint**	**dracor_operationId**
/{playcode}/text	get_text	/corpora/{corpusname}/play/{playname}/spoken-text	play-spoken-text
/{playcode}/charText	get_character_texts	/corpora/{corpusname}/play/{playname}/cast	get-cast
/{playcode}/charText/{character_id}.html	get_character_text_by_id	/corpora/{corpusname}/play/{playname}/spoken-text-by-character	play-spoken-text-by-character
/{playcode}/charText	get_character_texts	/corpora/{corpusname}/play/{playname}/cast/csv	get-cast-csv

Tab. 08: Mapping Folger endpoints to DraCor endpoints

Out of the 21 DraCor API endpoints that accept GET requests, four could be considered offering similar functionality to the endpoints implemented in the Folger API (see Tab. 08). They either provide a list of characters of a play in different formats or return the spoken text by a character. All other API endpoints offer supplementary functionality, which means that by combining the two APIs a user wanting to analyze Shakespearean plays has additional possibilities in comparison to using only one of the APIs. However, conducting actual research that would combine APIs entails a lot of additional work, in particular writing parsers for the Folger API HTML responses.

[231]Cf. https://dracor.org/shake; cf. GitHub repository: https://github.com/dracor-org/shakedracor

[232] If an OpenAPI documentation is published like the one on dracor.org, it is possible to address an endpoint with the "operationId" as fragment identifier appended to the URL, e.g. the endpoint `corpora` of the DraCor API can be located at https://dracor.org/doc/api#/public/list-corpora, where `public` being the tag. This URL could also serve as an identifier (URI) in a RDF based setup.

6.2 Bridging POSTDATA and DraCor as Programmable Corpora[233]

In this section we give a brief overview of the POSTDATA system developed by the ERC funded project "POSTDATA"[234] focusing on the "PoetryLab API", which allows for connecting a React front-end to a triple store used for storing and querying the data on the analysis of poetry. We then report on a demonstrator that we developed in the CLS INFRA project which extends the functionality of the already implemented POSTDATA system and integrates POSTDATAs Linked Open Linguistic Data (LLOD) (Chiarcos et al. 2013) via an API to an adapted DraCor-like front-end. This exemplary implementation can also be seen as a way on how to bridge the gap between the worlds of the "Semantic" and the "Programmable Web".

6.2.1 Overview of POSTDATA components

The POSTDATA project developed a suite of open-source tools[235] for

- A) the analysis of Spanish poetry, providing functionality for syllabification, scansion, enjambment detection, rhyme detection, historical named entity recognition,
- B) for building new corpora,
- C) for corpora conversion to RDF.

A) The tools for the analysis of Spanish poetry are accessible using an API that contains calls to a pipeline using the following tools:

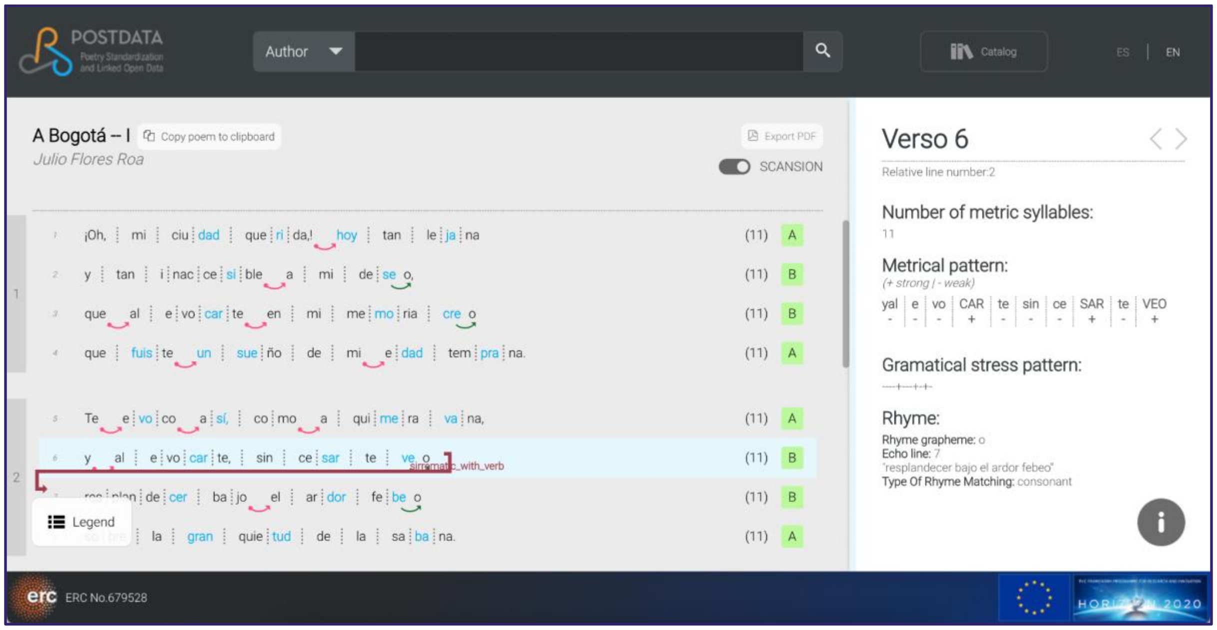
- Rantanplan[236], that is used for automatically analyzing the scansion of poems in Spanish, which involves the measurement of quantitative characteristics of verses, the detection of metrical patterns of verse lines, and the classification of stanza types. By relying on state-of-the-art NLP technology (*spaCy* and the developed spaCy affixes[237] for Spanish), Rantanplan is able to identify up to 45 different stanza types of Spanish poetry.[238]
- Jollyjumper[239], that detects enjambements, a common poetic device of extending a syntactic unit (a sentence) over the boundary of a metric unit (a verse).

[233] We thank Salvador Ros, Álvaro Pérez Pozo and Omar Khalil Gómez for the valuable insight into and access to the POSTDATA system, without which the implementation of the prototype would not have been possible.
[234] https://postdata.linhd.uned.es
[235] https://postdata.linhd.uned.es/results/poetrylab
[236] https://github.com/linhd-postdata/rantanplan
[237] https://github.com/linhd-postdata/spacy-affixes
[238] Cf. https://postdata.linhd.uned.es/results/poetrylab/rantanplan/
[239] https://github.com/linhd-postdata/jollyjumper

B) To retrieve and create new corpora, **Averell**[240] is used. This tool allows downloading corpora from several[241] multilingual sources and transform the data into a common JSON format, which is then further processed with the other components of the pipeline.

C) Finally, in order to convert the corpus to RDF, **Horace**[242] is used. This tool transforms the acquired results of the automatic analyses to a RDF representation, which is ingested into a Stardog[243] triple store instance, that serves as the central data store for the system.

With POSTDATAs central user-facing component, **PoetryLab application**[244] a user of the platform can search for poems by title or browse the collection by poem and author. After selecting a single poem the results of a manual—which is not available for all poems—and an automatic scansion analysis can be examined in detail (cf. Fig. 35).



A Bogotá -- I
Julio Flores Roa

1
¡Oh, mi ciu dad que ri da,! hoy tan le ja na (11) A
y tan i nac ce si ble a mi de se o, (11) B
que al e vo car te en mi me mo ria cre o (11) B
que fuis te un sue ño de mi e dad tem pra na. (11) A

2
Te e vo co a sí, co mo a qui me ra va na, (11) A
y al e vo car te, sin ce sar te ve o (11) B
sirrematic_with_verb
res plan de cer ba jo el ar dor fe be o (11) B
la gran quie tud de la sa ba na. (11) A

Verso 6

Relative line number:2

Number of metric syllables:
11

Metrical pattern:
(+ strong | - weak)

yal	e	vo	CAR	te	sin	ce	SAR	te	VEO
-	-	-	+	-	-	-	+	-	+

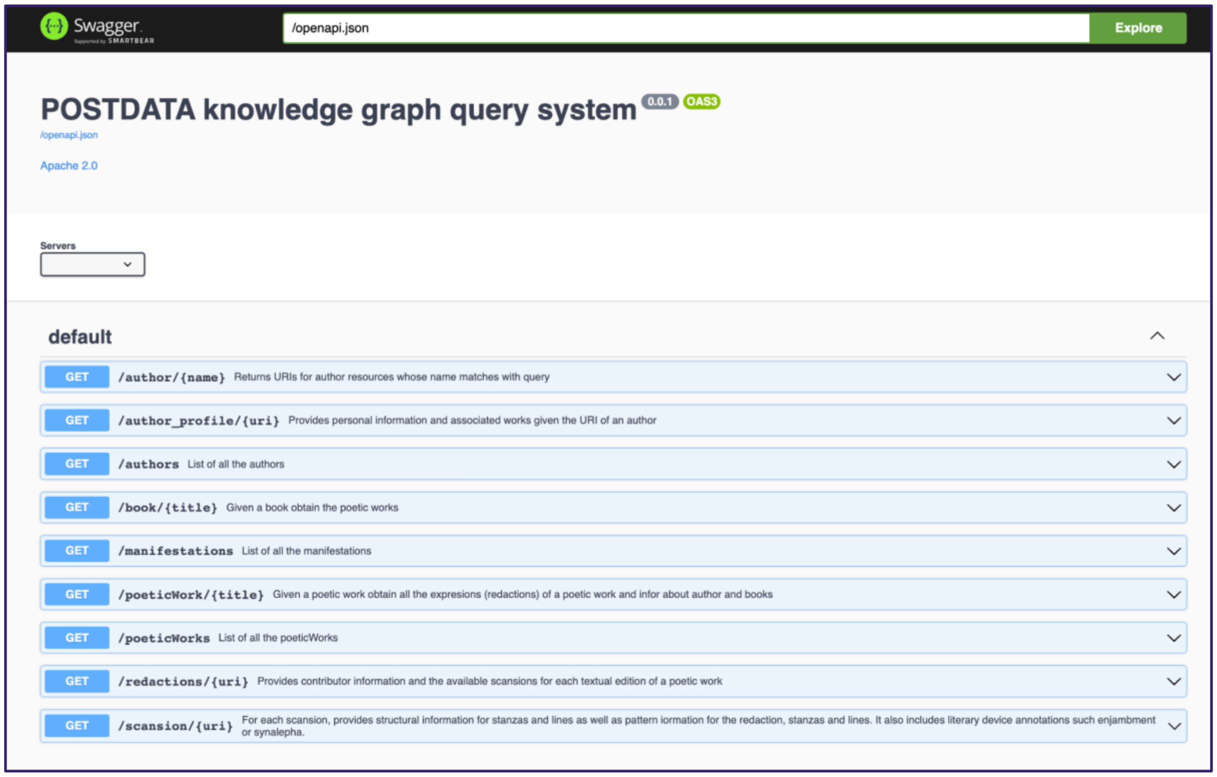
Gramatical stress pattern:
----+---+-+-

Rhyme:
Rhyme grapheme: o
Echo line: 7
"resplandecer bajo el ardor febeo"
Type Of Rhyme Matching: consonant

Fig. 35: Analysis view of a single poem in POSTDATAs Poetry Lab App

240 https://github.com/linhd-postdata/averell
241 "10" according to the description of Averall on the POSTDATA website https://postdata.linhd.uned.es/results/poetrylab/averell, but the overview of "Corpora and Datasets" contains links to 12 repositories on GitHub and one dataset on Hugging Face, cf. https://postdata.linhd.uned.es/corpus-y-datasets. Not all this data is available in the Knowledge Graph.
242 https://github.com/linhd-postdata/horace
243 https://www.stardog.com. While all the developed tools within the POSTDATA project are open-source tools and are licensed under open licenses (e.g., Apache 2.0), *Stardog* is a proprietary software solution.
244 http://poetry.linhd.uned.es:3000/en, Code available on GitHub: https://github.com/linhd-postdata/poetrylab

The user interface is powered by an underlying RESTful API[245], which comprises of nine[246] endpoints. The endpoints are mainly geared towards returning the data that is consumed by the PoetryLab front-end. The API is documented with an OpenAPI specification, that is visualized with SwaggerUI.



POSTDATA knowledge graph query system 0.0.1 OAS3

/openapi.json

Apache 2.0

default

GET `/author/{name}` Returns URIs for author resources whose name matches with query

GET `/author_profile/{uri}` Provides personal information and associated works given the URI of an author

GET `/authors` List of all the authors

GET `/book/{title}` Given a book obtain the poetic works

GET `/manifestations` List of all the manifestations

GET `/poeticWork/{title}` Given a poetic work obtain all the expresions (redactions) of a poetic work and infor about author and books

GET `/poeticWorks` List of all the poeticWorks

GET `/redactions/{uri}` Provides contributor information and the available scansions for each textual edition of a poetic work

GET `/scansion/{uri}` For each scansion, provides structural information for stanzas and lines as well as pattern iormation for the redaction, stanzas and lines. It also includes literary device annotations such enjambment or synalepha.

Fig. 36: : PoetryLab API visualized with SwaggerUI

In the following, we discuss some of the endpoints in their relation front-end functionality and also touch upon some design decisions that were taken when implementing the API.

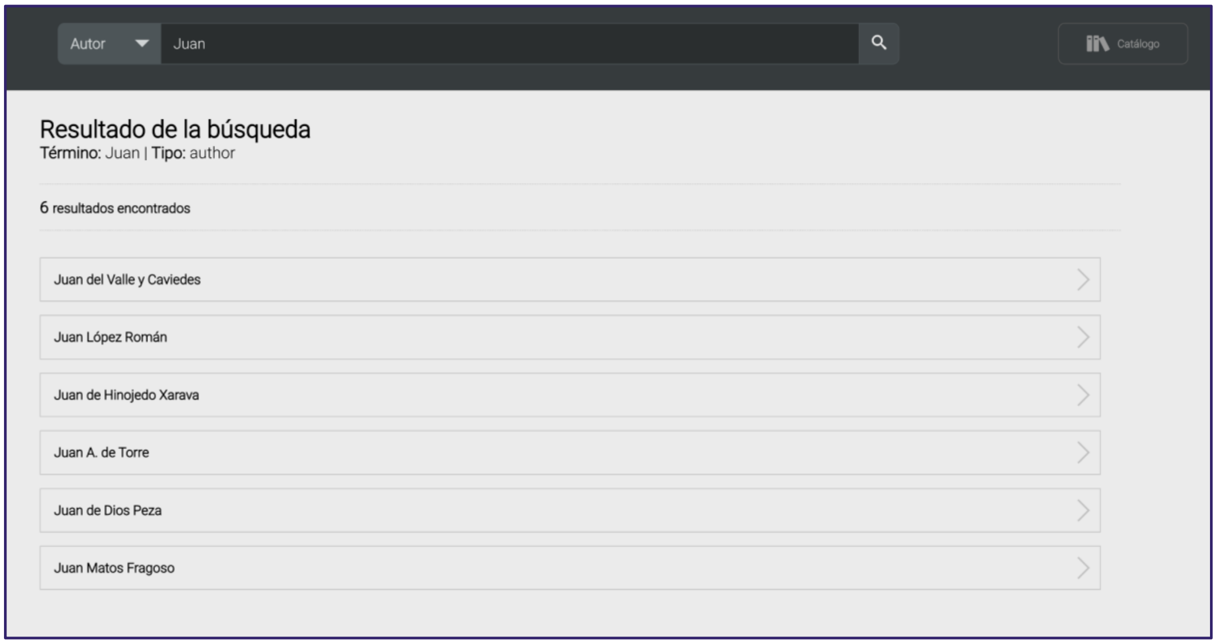
The PoetryLab application contains a search box in the header on almost every page which allows for searching for authors (by name) and poems (by title). The functionality to query for a certain author is provided by the API endpoint `/author/{name}`[247]. For example, if a user

[245] The code of the API is available on GitHub: https://github.com/linhd-postdata/knowledge-graph-queries. It can be accessed at http://poetry.linhd.uned.es:5005. The interactive Swagger documentation is published at http://poetry.linhd.uned.es:5005/ui with the underlying OpenAPI specification, that is referenced in this report, being available at <133>.

[246] As of writing of this report only eight endpoints worked as expected. The endpoint `/manifestations` is not fully implemented. Instead of returning a "200" status code and an object containing "TODO" as value and key in the response body, the information, that the endpoint is planned but has not been fully implemented yet, would be better conveyed by returning a status code of 501 "Not implemented".

[247] As an example of how API and front-end are interlinked in the case of POSTDATA we discuss this endpoint in more detail. The interactive documentation of the endpoint is available at

enters the string “Juan” into the search box and hits the search button, the results of the search are displayed as shown in the screenshot below.



Resultado de la búsqueda
Término: Juan | Tipo: author

6 resultados encontrados

Juan del Valle y Caviedes
Juan López Román
Juan de Hinojedo Xarava
Juan A. de Torre
Juan de Dios Peza
Juan Matos Fragoso

Fig. 37: Search results as displayed in the PoetryLab front-end

These are the rendered results, that are returned by a query to the above mentioned API endpoint when requesting data from the URL http://poetry.linhd.uned.es:5005/author/Juan. The response body contains a JSON array containing six results (of which we here include the first three as an example only[248]), ranked by the value of the field with the key `score`:

```
[
 {
  "@id": "http://postdata.linhd.uned.es/resource/p_juan-del-valle-y-caviedes",
  "birthDate": "1652-01-01T00:00:00Z",
  "deathDate": "1692-01-01T00:00:00Z",
  "name": "Juan del Valle y Caviedes",
  "score": 1.0
 },
 {
  "@id": "http://postdata.linhd.uned.es/resource/p_juan-lopez-roman",
```

http://poetry.linhd.uned.es:5005/ui/#/default/knowledge_graph_queries.get_author. The endpoint to query for a poem by title `/poeticWork/{title}` works in a similar way. Refer to the interactive documentation of the endpoint at http://poetry.linhd.uned.es:5005/ui/#/default/knowledge_graph_queries.get_poeticWork.

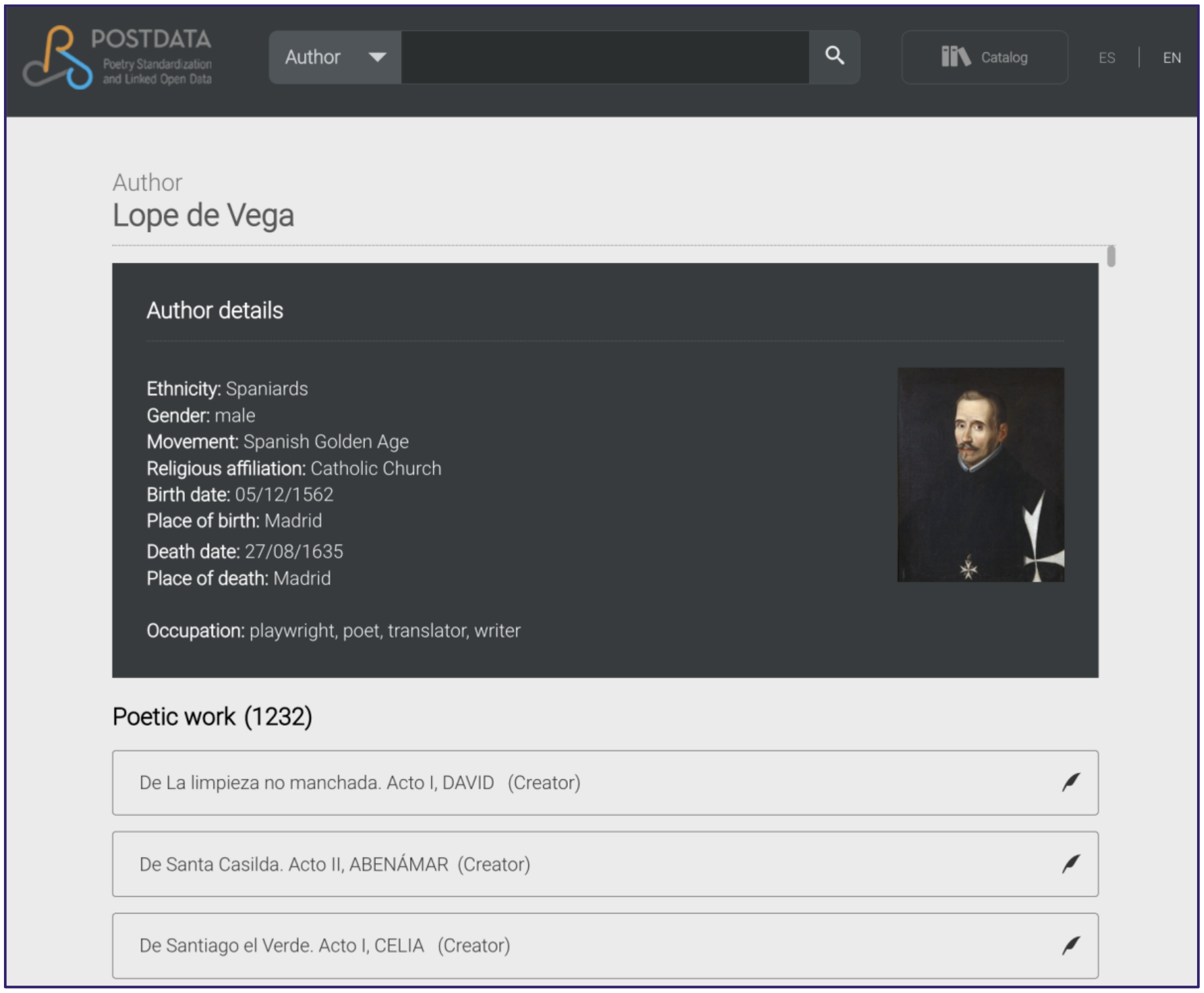
[248] It would also be possible to achieve such a result by using the optional query parameter `limit`: http://poetry.linhd.uned.es:5005/author/Juan?limit=3

```
    "birthDate": "1583-06-24T00:00:00Z",
    "deathDate": "1663-09-24T00:00:00Z",
    "name": "Juan L\u00f3pez Rom\u00e1n",
    "score": 1.0
  },
  {
    "@id": "http://postdata.linhd.uned.es/resource/p_juan-de-hinojedo-xarava",
    "name": "Juan de Hinojedo Xarava",
    "score": 1.0
  }
]
```

Apart from "score", which contains a ranking value generated by the full-text search algorithm of the Stardog triple store, all other fields are populated with data from the Knowledge Graph, e.g. the value of the field `@id` contains the URI of an instance of the class `pdc:Person` from the "Ontopoetry Core Ontology".[249] Retrieving or knowing the URI is crucial, because the other endpoint `/author_profile/{uri}`, which returns information on a single author, expects an URI as the value of the path parameter `uri`.

One of the pages with more detailed information on a single author (including a profile picture, that is fetched from Wikidata) is the profile of the Spanish poet Lope de Vega, which can be accessed at http://poetry.linhd.uned.es:5005/author_profile/http%3A%2F%2Fpostdata.linhd.uned.es%2Fresource%2Fp_lope-de-vega (see Fig. 38).

[249] The documentation of the ontology can be accessed at https://postdata.linhd.uned.es/ontology/postdata-core/documentation/index-en.html. The person class is defined here: https://postdata.linhd.uned.es/ontology/postdata-core/documentation/index-en.html#Person.



Author
Lope de Vega

Author details

Ethnicity: Spaniards
Gender: male
Movement: Spanish Golden Age
Religious affiliation: Catholic Church
Birth date: 05/12/1562
Place of birth: Madrid
Death date: 27/08/1635
Place of death: Madrid

Occupation: playwright, poet, translator, writer

Poetic work (1232)

De La limpieza no manchada. Acto I, DAVID (Creator)

De Santa Casilda. Acto II, ABENÁMAR (Creator)

De Santiago el Verde. Acto I, CELIA (Creator)

Fig. 38: Profile page of Lope de Vega in PoetryLab application

In the example, special characters in the URI identifying the author (http://postdata.linhd.uned.es/resource/p_lope-de-vega) are escaped when used in the path parameter of the API call. Although the URL "http://poetry.linhd.uned.es:5005/author_profile/http://postdata.linhd.uned.es/resource/p_lope-de-vega" equally works, we on purpose used the escaped URI which was generated by the Swagger UI. It illustrates the challenge in designing meaningful endpoints for APIs that use HTTP-URIs as identifiers for entities, which seems suitable when designing REST APIs for Linked Data. According to RFC 3986 "[a] percent-encoding mechanism is used to represent a data octet in a component when that octet's corresponding character is outside the allowed set or is being used as a delimiter of, or within, the component."[250] In the case of the resulting URLs of the API endpoint, the slash character "/" is used both as part of the identifier and as a delimiter in the endpoint URL and should therefore generally be escaped. From an API design perspective using

[250] https://www.rfc-editor.org/rfc/rfc3986#section-2.1

full URIs in path parameters of APIs is a bit confusing and makes the URLs quite hard to read (for humans). This might not be an argument from a technical perspective, but if one would—as we do—argue that APIs should be considered not only interfaces for machines being consumed by scripts and tools, but should also be used by human agents, then readability of URIs for human users becomes a quite important aspect.

In the case of POSTDATA, in the PoetryLab front-end short IDs for authors are used consequently, e.g., “lope-de-vega”, which might be a good practice. Although, in the case of the POSTDATA Knowledge Graph these short IDs do not seem to be explicitly included (e.g., as an CIDOC `crm:E42_Identifier` or a subclass thereof). Even though they are obviously derived from the URI by some formal mechanism, the knowledge about the identifiers is moved from the Knowledge Graph into the processing and thus makes it harder to go from front-end resp. API back to the Knowledge Graph, e.g., when manually writing SPARQL Queries. A user wanting to do so, would have to transform the short ID back into a full URI.

6.2.2 POSTDATAs Specification-first Approach in API Development

In developing the REST API, the POSTDATA project followed a specification-first approach that was implemented by using “Connexion”[251]. The Python package takes an OpenAPI specification and automatically maps the defined endpoints to custom Python functions based on the “operationId” field in the specification.

This specification-first development approach prioritizes the API design process and thus could be seen as a role model in implementing APIs for Programmable Corpora. The approach allows for tightly integrating the modeling of literary objects and the API design process. In this case, an intermediate step of programming an API—as is needed in code-first API development approaches—is not necessary. The process of designing the API can take place before the implementation, so that not technically adept domain experts (e.g., scholars of literary studies) can be actively involved. However, a prerequisite for this is that the right tools (that do not necessarily involve coding work)[252] are available. This approach might not only speed up the development process resulting in prototypes to be tested earlier, but also would allow literary scholars to actively shape the API development process from the beginning.

[251] https://pypi.org/project/connexion. Code repository on GitHub: https://github.com/spec-first/connexion
[252] After a first evaluation of platforms that are usually used in API design and development, it seems that Postman (https://www.postman.com) could serve as a platform, based on which such a design process could be implemented. Postman can be considered the market leader in this segment, but it should be noted that it is a commercial product and thus is not the ideal choice for a scholarly project that wants to use freely available, self-hosted open-source tools.

In the following, we will briefly highlight some of the technical aspects of the implementation of POSTDATAs PoetryLab API.[253] We will look at the `/poeticWorks` endpoint that returns a list of instances of `pd:PoeticWork`[254], i.e., the available poems.

In the module “core.py”, there are the functions that issue SPARQL queries to the triple store and return the data as the response of the designated endpoint. The following Python code snippet implements the above-mentioned endpoint[255]:

```
def get_poeticWorks():
        """ Method corresponding to the poeticWorks endpoint
        :return: JSON with the list of all poeticWorks in the knowledge graph
        """
    conn = get_db()
    query = QUERIES['poeticWorks']
    results = conn.graph(query, content_type=stardog.content_types.LD_JSON)
    jsonld_results = json.loads(results)
    compacted = jsonld.compact(jsonld_results, CONTEXT)
    graph = compacted.get("@graph")
    return graph
```

At first, a connection with the triple store is established using a designated function.[256] Then, a SPARQL query[257] is loaded from a separate module, which includes a dictionary with all pre-defined SPARQL queries[258]. The query that is used in the function is stored as a string, accessible from the dictionary “QUERIES” by the key `poeticWorks`[259].

```
PREFIX kos: <http://postdata.linhd.uned.es/kos/>
PREFIX pdc: <http://postdata.linhd.uned.es/ontology/postdata-core#>
CONSTRUCT{
    ?work pdc:title ?title;
```

[253] The GitHub repository is available at https://github.com/linhd-postdata/knowledge-graph-queries. The OpenAPI specification that is the foundation in the workflow can be found here <133>. The setup of the API is performed in the module “app.py” <134>, where connexion is used to instantiate an API based on the OpenAPI specification. The functions that are used to define the endpoints are in the module “core.py” <135>.
[254] https://postdata.linhd.uned.es/ontology/postdata-core/documentation/index-en.html#PoeticWork.
[255] Cf. <136>.
[256] The package “pystardog” is used to communicate with the triple store: https://pypi.org/project/pystardog. See the examples in the README on how to establish a connection to the triple store. In the case of POSTDATA it is handled by the function `get_db` <137>, which, in turn, relies on the function `connect_to_database` <138>.
[257] Cf. <139>.
[258] Cf. module “queries.py” <140>.
[259] Cf. <141>.

```
            pdc:author ?creator;
            pdc:date ?date.
    }
    # FROM <tag:stardog:api:context:local>
    WHERE {
        ?work pdc:title ?title.
        ?work a pdc:PoeticWork.
        ?creation pdc:initiated ?work;
        pdc:hasAgentRole ?ag.
        ?ag pdc:hasAgent ?person;
            pdc:roleFunction kos:Creator.
        ?person pdc:name ?creator.
        OPTIONAL{
            ?creation pdc:hasTimeSpan ?sp.
            ?sp pdc:date ?date.
        }
    } ORDER BY ?title
```

This SPARQL CONSTRUCT query is executed against the SPARQL endpoint of the triple store and the results are transformed to a JSON-LD[260] file , which is returned in the response of the API endpoint. A single poem object that is contained in the returned array looks as follows:

```
    {
        "@id": "http://postdata.linhd.uned.es/resource/pw_mira-de-amescua_viii-de-
    los-prodigios-de-la-vara-y-capitan-de-israel-jornada-primera-faraon-rey",
        "author": "Mira de Amescua",
        "title": " - VIII - De Los prodigios de la vara y capitán de Israel.
    Jornada primera, Faraón rey "
      }
```

6.2.3 Prototyping a DraCor-like API for POSTDATA

By developing this prototype, we intended to test how a DraCor-like REST API can be added on top of POSTDATAs infrastructure, thus laying the foundation of what could possibly become "PoeCor", short for "Poetry Corpora". With this prototype we are transferring the concept of Programmable Corpora not only from drama to poetry, but there are also some crucial transformations in regard to the underlying data model and the technology stack. The following differences between the projects were to consider:

[260] https://json-ld.org. The specification of JSON-LD can be accessed at https://www.w3.org/TR/json-ld. The package "PyLD" is used to simplify the processing, e.g., the compacting and expanding, of this format: https://pypi.org/project/PyLD

Core Entities: POSTDATA has poems and authors as its core entities, while DraCor focuses on corpora and theater plays.

In the DraCor context, the entity "author" is not of such relevance, e.g., DraCor does not have its own identifiers for authors, but uses Wikidata (or other external identifiers), whereas POSTDATA has identifiers for authors (also because by using LOD technology, having URIs to identify things is necessary), and, in the PoetryLab App front-end, uses URL patterns that explicitly include author names.

The entity "corpus", on the other hand, is not included in POSTDATAs ontology and does not play a role in organizing the data. The data model is very expressive in regard to the analysis process (Scansion Process[261]) and the results thereof (Scansion[262]) but does not provide information on provenance or about a poem being part of a collection or corpus. So even though the division of the given poems into subcorpora would make sense it cannot be performed automatically with the metadata at hand. In DraCor, the notion of "corpus" is very relevant to the way data is organized, which, for example, manifests itself in the pattern of the URLs of the API endpoints, that in almost all cases address a single play in regard to the corpus it is part of.

In the case of the API prototype, we developed based on POSTDATAs infrastructure, we theoretically allow for having multiple corpora, but hard-coded a single corpus "postdata" (as the corpus name path parameter) into the API.

Data format: While DraCor is based on TEI-XML, POSTDATA relies on a homogenized JSON serialization that is the basis for analysis and then transformed into RDF. Also, the textual data is included in the triple store as literals, e.g. the content of a single verse line is attached to the verse entity with a designated property (`pdp:content`[263]).

Data store: DraCor stores the data in an XML database; POSTDATA in a triple store.[264]

[261] https://postdata.linhd.uned.es/OntoPoetry/Poetic/documentation/index-en.html#ScansionProcess
[262] https://postdata.linhd.uned.es/OntoPoetry/Poetic/documentation/index-en.html#Scansion
[263] https://postdata.linhd.uned.es/OntoPoetry/Poetic/documentation/index-en.html#content.
[264] The cooperation within the CLS INFRA project made it possible to directly access the production instance of POSTDATAs Stardog. We also tested scenarios in which we set up a local instance of *Stardog* and ingested (parts of) POSTDATAs RDF data, which also proved to be possible. The POSTDATA project provides docker-copose/Docker files in the repositories of the individual components of the system, that allow for a replication of the infrastructure. See for example the docker-compose of the "knowledge-graph-queries" repository <142>. For a Dockerfile to run the Stardog triple store cf. <143>. For development we used our own Docker image of Stardog: https://github.com/dh-network/stardog-docker-compose. In this case, the database "PD_KG" had to be manually created using *Stardog Studio* (https://www.stardog.com/studio), the data was ingested by running the following command in the terminal from the unzipped folder containing the RDF data of the Knowledge Graph: `for f in *.ttl; do curl -X POST -H 'Content-Type:application/x-turtle' --data-binary @$f http://admin:admin@localhost:5820/PD_KG ; done`

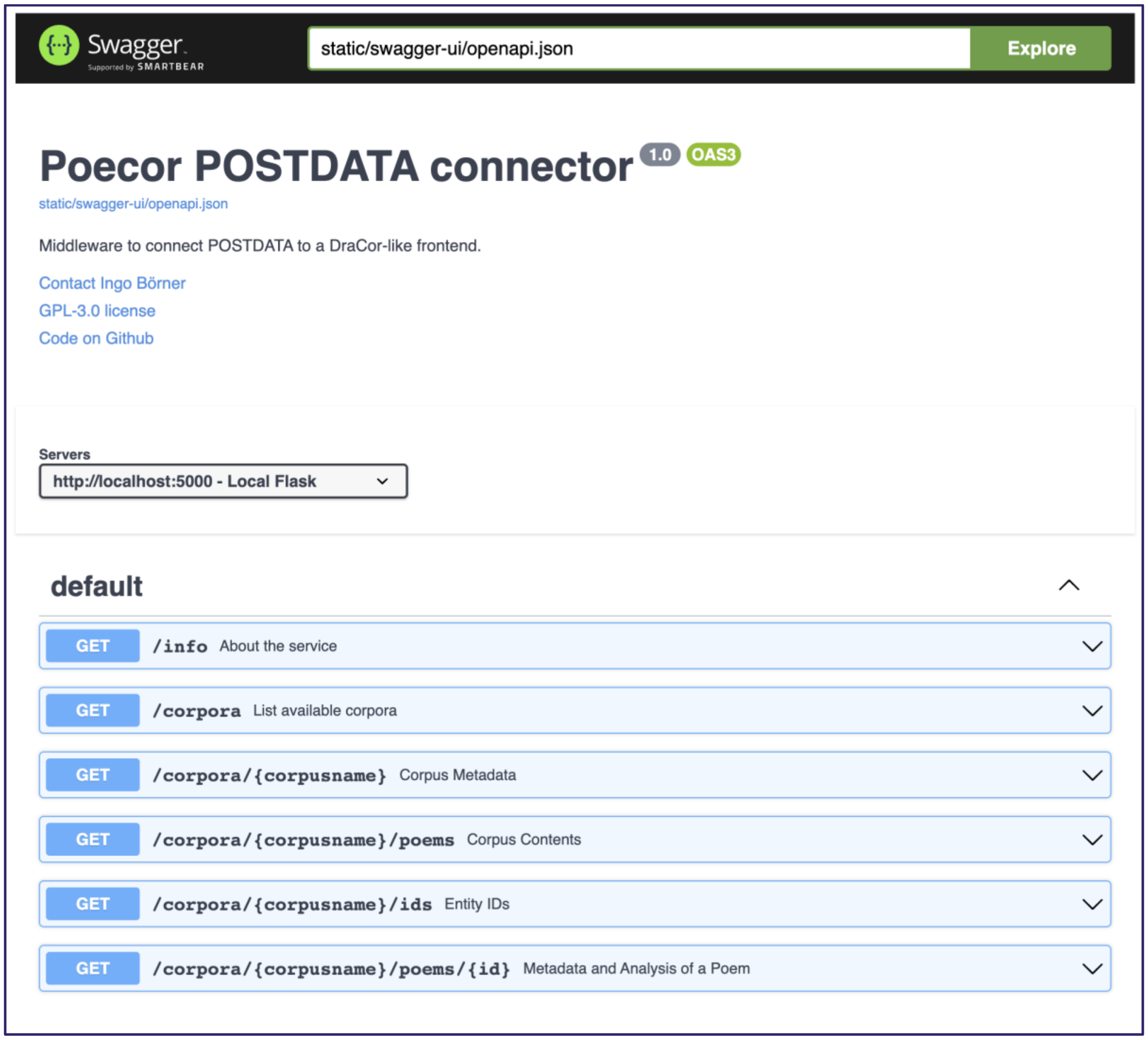
Programming Language: While the front ends of both systems are based on the Javascript library “React”, DraCor relies mainly on XQuery, POSTDATA on Python (in the implementation of the backend and especially the API). Although it would be possible to manipulate RDF data[265] also in XQuery (especially, in its XML serialization), because of its flexibility and the wide range of available packages (especially, the “pystardog”, provided by the developers of Stardog to natively connect to the triple store) the decision to use Python as a programming language is obvious.

Processing: In the case of DraCor the analysis data (e.g., network- and count-based metrics) need to be calculated based on the XML data or extracted network graphs. POSTDATA provides ready-to-use analysis data in its Knowledge Graph, and therefore it is not necessary to implement an elaborated processing layer that extracts and calculates these metrics. Still, there is a need to aggregate metrics. For example, the information on the number of word tokens is available for a single verse line. So in order to be able to return the number of words for a single stanza (or a single poem), first we need to count the words connected to a single line (`pdp:has_word`[266]), then the results of each line of the stanza need to be summed up.

The prototype we developed provides an API that feeds a front-end that was built based on the DraCor front-end (cf. section 5.6)[267]. The “PoeCor POSTDATA connector API[268] can also be used stand-alone to investigate and further analyze the corpus, that is represented by POSTDATAs Knowledge Graph.

In the current state of development, the API has six endpoints which are documented by an OpenAPI specification[269] (see Fig. 39 for a visualization in SwaggerUI). A description of the functionality of the endpoints (and some thoughts on their design) can be found below:

[265] Cf. for example the module “rdf.xqm” of the DraCor eXist application, that transforms TEI-XML to RDF.
[266] http://postdata.linhd.uned.es/ontology/postdata-poeticAnalysis#hasWord.
[267] GitHub Repository: https://github.com/dracor-org/poecor-frontend. The front-end has been deployed to https://poecor.org, but operates on mock data that has been pre-generated with a Jupyter Notebook <144>.
[268] GitHub Repository: https://github.com/dh-network/postdata-2-dracor-api. The API has not been deployed to the server yet. It can be run locally as a Docker container.
[269] An OpenAPI specification is available at: <145>, see also Fig. 39.



static/swagger-ui/openapi.json

Poecor POSTDATA connector 1.0 OAS3

static/swagger-ui/openapi.json

Middleware to connect POSTDATA to a DraCor-like frontend.

Contact Ingo Börner
GPL-3.0 license
Code on Github

Servers
http://localhost:5000 - Local Flask

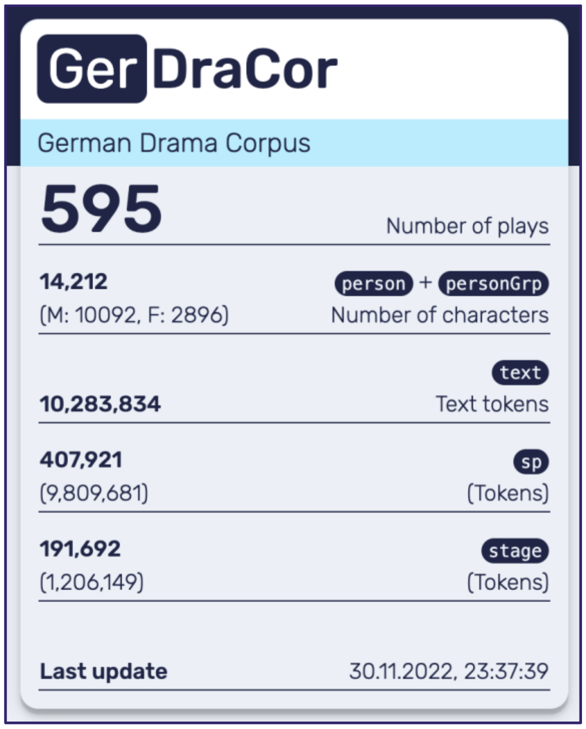
default

GET `/info` About the service
GET `/corpora` List available corpora
GET `/corpora/{corpusname}` Corpus Metadata
GET `/corpora/{corpusname}/poems` Corpus Contents
GET `/corpora/{corpusname}/ids` Entity IDs
GET `/corpora/{corpusname}/poems/{id}` Metadata and Analysis of a Poem

Fig. 39: OpenAPI specification of the API displayed in SwaggerUI

`/info`: returns information about the API, including the “name” of the service, a “description” and a “version” number.

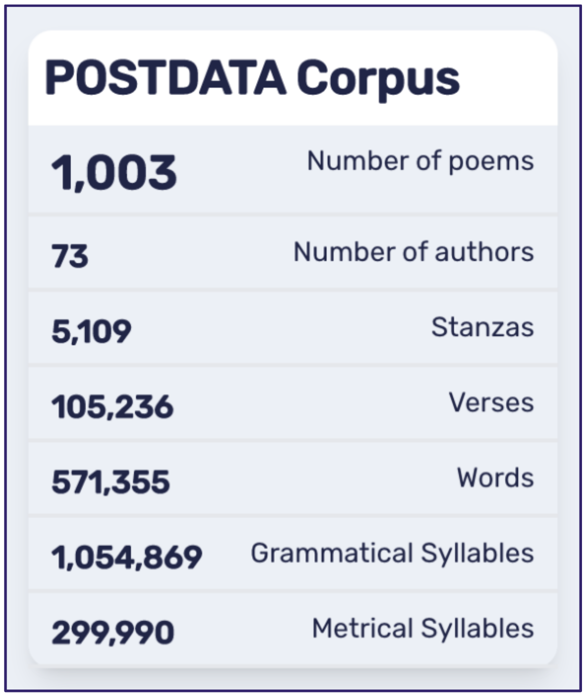
`/corpora`: provides an overview of available corpora. In the prototype, there is only the corpus “postdata” available. Metrics per corpus can be included in the response by setting the optional parameter “include” to “metrics”, which includes counts of “authors”, “poems”, “stanzas”, “verses”, word tokens “words” and syllables (“grammatical_syllables” and “metrical_syllables”). This endpoint returns the data that is used to render the cards of individual corpora. Fig. 40 shows a card from the DraCor front-end next to its preliminary adaption in the “PoeCor” front-end (Fig. 41).



GerDraCor

German Drama Corpus

595	Number of plays
14,212 (M: 10092, F: 2896)	person + personGrp Number of characters
10,283,834	text Text tokens
407,921 (9,809,681)	sp (Tokens)
191,692 (1,206,149)	stage (Tokens)
Last update	30.11.2022, 23:37:39



POSTDATA Corpus

1,003	Number of poems
73	Number of authors
5,109	Stanzas
105,236	Verses
571,355	Words
1,054,869	Grammatical Syllables
299,990	Metrical Syllables

Fig. 40: DraCor corpus overview ("card view")

Fig. 41: PoeCor corpus overiew ("card view")

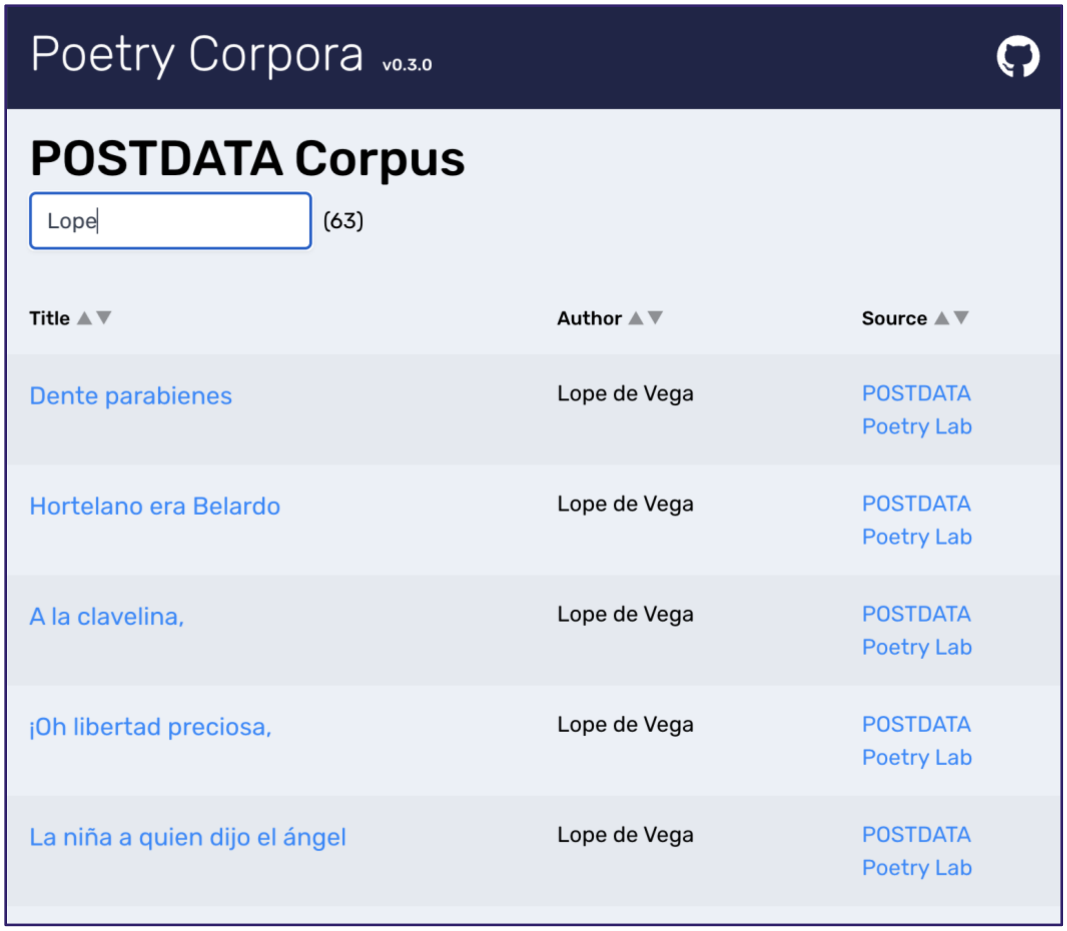
`/corpora/{corpusname}`: returns metadata on a single corpus with metrics included (see `/corpora` endpoint). Unlike the corresponding endpoint in the DraCor API (cf. section 5.3), this endpoint does not return the contents of a corpus.

`/corpora/{corpusname}/poems`: returns a list of the poems in a corpus. In designing the endpoints we decided to exclusively use plural nouns when devising URL patterns of endpoints.[270] This allowed to move the functionality to list a corpus' contents to a designated endpoint, which now not only clearly distinguishes between metadata and contents, i.e. the metadata of individual poems, but also allows to attach additional functionality to this new endpoint, like requesting paged results with the parameters "limit" and "offset" (per default, a set of 500 results is returned, i.e. offset=0, limit=500). Additionally, with the parameter "include" set to "authors" information on the author of each returned poem can be included in the response. A further step in the development of this endpoint will be to implement a mechanism of filtering the returned poems by criteria, that are calculated on the basis of individual poems and returned by the

[270] The DraCor API uses "play", e.g., `/corpora/{corpusname}/play/{playname}`. The use of plural nouns in the URL of endpoints is encouraged by guides on API design, e.g., "Rule: A plural noun should be used for collection names" (Massé 2012: 17).

`/corpora/{corpusname}/poems/{id}` endpoint, e.g., number of stanzas or verses, rhyme schema, and more.

A DraCor-like overview of a single corpus would probably have to combine data from the endpoints `/corpora/{corpusname}` and `/corpora/{corpusname}/poems`. Fig. 42 shows an overview table that has been developed based on the corpus page in DraCor (cf. section 5.6.2). Currently, the page available on https://poecor.org/corpora/postdata renders mock data that is not coming directly from an API.



POSTDATA Corpus

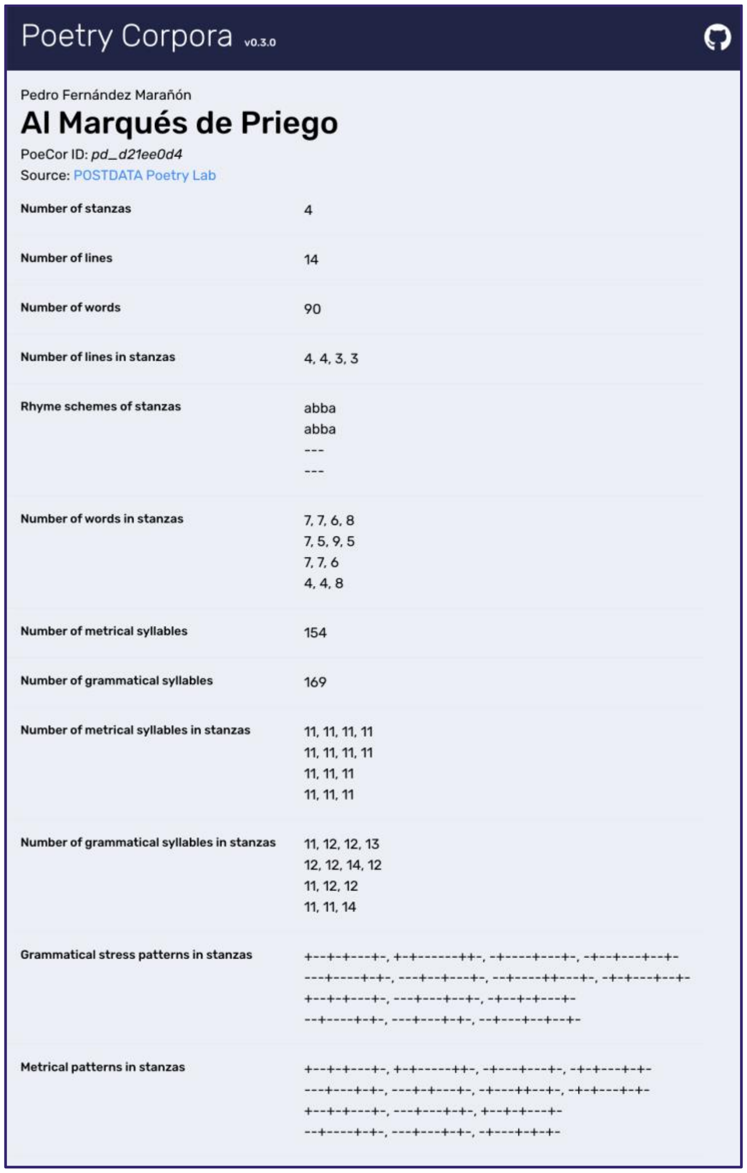
Title ▲▼	Author ▲▼	Source ▲▼
Dente parabienes	Lope de Vega	POSTDATA Poetry Lab
Hortelano era Belardo	Lope de Vega	POSTDATA Poetry Lab
A la clavelina,	Lope de Vega	POSTDATA Poetry Lab
¡Oh libertad preciosa,	Lope de Vega	POSTDATA Poetry Lab
La niña a quien dijo el ángel	Lope de Vega	POSTDATA Poetry Lab

Fig. 42: Corpus overview table with filter functionality

`/corpora/{corpusname}/ids`: returns a list of IDs of a certain entity type, e.g. poem or author. In the current version “poem” is the default and only allowed value of the parameter “type”. This endpoint is the result of a consequent separation of functionalities, but also a pragmatic necessity. We decided not to use the full URIs of entities in the API as identifiers in the request URLs, but derive shorter IDs by creating a truncated md5 hash of the URIs, e.g., of poems. This method allows to generate unique identifiers (on the basis of unique identifiers, i.e., the URIs used in the

Knowledge Graph), which are all of equal length and relatively easy to compute. This approach also has its downsides: An additional endpoint will have to be implemented that provides a means of translating back from a generated ID to the full URI. Although hashing a string always returns the same result and is thus a good means of creating unique identifiers of already unique strings, it is not possible to transform the hash back to the full string.

`/corpora/{corpusname}/poems/{id}`: returns metadata and the results of the analysis of some structural features of a poem. The analysis is based exclusively on an automatic scansion generated by POSTDATAs system. This endpoint does complement the `/scansion` endpoint that is available from PoetryLab API. For analysis purposes, POSTDATAs endpoint and the corresponding interface in the PoetryLab application is best suited. Our endpoint summarizes data that would otherwise had to be aggregated from the nested JSON-LD. The analysis data (in a field with the key “analysis”) comprises information on the overall number of stanzas, lines, words, and syllables (metrical and grammatical) of the whole poem, as well as aggregated metrics per stanza (rhyme scheme, number of lines, number of words and syllables per line, metrical and grammatical stress patterns).



Poetry Corpora v0.3.0

Pedro Fernández Marañón

Al Marqués de Priego

PoeCor ID: *pd_d21ee0d4*
Source: POSTDATA Poetry Lab

Number of stanzas	4
Number of lines	14
Number of words	90
Number of lines in stanzas	4, 4, 3, 3
Rhyme schemes of stanzas	abba abba --- ---
Number of words in stanzas	7, 7, 6, 8 7, 5, 9, 5 7, 7, 6 4, 4, 8
Number of metrical syllables	154
Number of grammatical syllables	169
Number of metrical syllables in stanzas	11, 11, 11, 11 11, 11, 11, 11 11, 11, 11 11, 11, 11
Number of grammatical syllables in stanzas	11, 12, 12, 13 12, 12, 14, 12 11, 12, 12 11, 11, 14
Grammatical stress patterns in stanzas	+--+-+---+-, +-+------++-, -+----+---+-, -+--+---+--+- ---+----+-+-, ---+--+---+-, --+----++---+-, -+-+---+--+- +--+-+---+-, ---+---+--+-, -+--+-+---+- --+----+-+-, ---+---+-+-, --+----+--+-+-
Metrical patterns in stanzas	+--+-+---+-, +-+-----++-, -+---+---+-, -+-+---+-+- ---+---+-+-, ---+-+---+-, -+---++--+-, -+-+---+-+- +--+-+---+-, ---+---+-+-, +--+-+---+- --+----+-+-, ---+---+-+-, -+---+-+-+-

Fig. 43: *Front-end rendering of the data returned by* `/corpora/{corpusname}/poems/{id}`

6.2.4 Notes on the Implementation

A likewise possible, but different way to dock an adapted DraCor front-end to POSTDATAs PoetryLab API could have been to simply implement additional endpoints. In that case, the straight-forward specification-first approach of POSTDATA would have made the following steps necessary:

- design additional endpoints and adapt the OpenAPI specification accordingly,
- write additional functions for the module "core.py" (or make explicit, that this is added functionality by adding the functions as a new module)

- make sure that the “operationId” of the operation (path + method, e.g. GET) matches the name of function[271],
- register the functions “in _init_.py”.

Even if this, in general, could have been implemented with less effort, there would be still additional things to consider in regard to implementation: POSTDATA uses the bare compacted JSON-LD to feed the front-end, which is not foreseen with, for example, the DraCor front-end. Therefore, it would still have been necessary to implement a mechanism that would further simplify the JSON-LD response further and to create a new data structure that would suffice the needs of the DraCor front-end. It didn’t seem feasible to introduce this processing logic to the POSTDATA Knowledge Graph Query system as implemented for the PoetryLab API, because it would have unnecessarily blown up the otherwise sleek setup. We therefore decided to keep the APIs separate, at least for now.

In the case of the eXist-db XQuery-based implementation of the DraCor API, the central aspects are the XPath expressions and the XQuery functions (the “operations”) that are evaluated or executed to extract information from the TEI encoded texts, resulting in the “features” that are then available from several endpoints (cf. section 5.3 on DraCor API). In the case of POSTDATAs approach, in which the data is stored in a Knowledge Graph, the foundation on which the implementation rests are the SPARQL CONSTRUCT queries[272], that define which information to retrieve and how the JSON-LD object is structured in the response.

We followed POSTDATAs approach in foregrounding the SPARQL queries insofar as we set up a separate structure to store the SPARQL queries. But, in the case of POSTDATA, the queries are kept in a relatively flat data structure—a dictionary, which only includes the query as a string. However, we wanted to implement the queries in a way that would allow us to attach additional forms of documentation (a “label” and a “description”) and functionality (“inject URIs into a template”, “execute a query”) to them. Therefore, the queries are created as subclasses `PdStardogQuery`[273] of a class `SparqlQuery`[274] with the actual query hardcoded as an attribute of either “query” or “template”, depending on whether there is a need of injecting URIs into a template before execution. These classes inherit methods from `SparqlQuery` that allow to

[271] Just to give one exampleIn the specification the endpoint `/poeticWorks` of the PoetryLab API has the operationID “knowledge_graph_queries.get_poeticWorks” <146>; the function is called `get_poeticWorks` in “core.py”. The mapping is handled by “connexion”, cf. chapter on “Routing” in the documentation of connexion (https://connexion.readthedocs.io/en/latest/routing.html).
[272] https://www.w3.org/TR/sparql11-query/#construct.
[273] The class is defined in the module “pd_stardog_queries.py” <147>, which also contains its subclasses representing the individual SPARQL queries.
[274] The SPARQL related classes (DB, SparqlQuery, SparqlResults) are contained in the module “sparql.py” <148>.

manipulate a query template, send them to the triple store, and so on. The of the Stardog triple store are instantiated as instances of a designated class `SparqlResults` that has methods, which allow to "simplify"[275] the results in the "SPARQL Query Results JSON Format"[276].

Below is an example of such a "SPARQL Query Class" `CountPoeticWorks`: a simple query that counts the number of poems in the knowledge graph (Fig. 44).

```
class CountPoeticWorks(PdStardogQuery):
    """SPARQL Query: Count instances of class pdc:PoeticWork"""

    label = "Number of Poems"

    description = """
    Count all instances of the class pdc:PoeticWork. These should be
all the poems in the graph.
    """

    query = """
    SELECT (COUNT(?poeticWork) AS ?count) FROM
<tag:stardog:api:context:local> WHERE {
    ?poeticWork a pdc:PoeticWork .
    }
    LIMIT 1000000
    """
```

Fig. 44: Query counting number of poems in POSTDATA knowledge graph

The central entities (collection of corpora, corpus, poem, author)[277] are implemented as designated classes, which use instances of the above-mentioned "SPARQL Query Classes" to obtain results from the Stardog triple store. The code snippet below shows the implementation of the method `get_num_poems` that uses `sparql_num_poems`, an instance of the class `CountPoeticWorks`.

[275] The "simplify" method is implemented here: <149>.
[276] Specification cf. https://www.w3.org/TR/sparql11-results-json.
[277] Each general entity class is defined in a separate module ("corpora.py", "corpus.py", "poem.py", "author.py"). Of these classes specialized subclasses are created, that are tailored towards the data model employed in the POSTDATA system: the classes "PostdataCorpora", "PostdataCorpus", "PostdataPoem" and "PostdataAuthor". The classes can be found in the files, which filenames resemble the one of the modules containing the general classes, but prefixed by "pd_", e.g., "pd_corpus.py". The functionality to execute queries and transform the data are implemented with these classes. They can also be used as stand-alone modules, as is demonstrated with the corresponding Jupyter Notebooks, e.g. the notebook "test_corpus_class.py" demonstrates the stand-alone use of the class "PostdataCorpus" <150>.

```
def get_num_poems(self) -> int:
    """Count poems in corpus.

    Uses a SPARQL Query of class "CountPoeticWorks" of the module
"pd_stardog_queries".

    Returns:
        int: Number of poems.
    """
    if self.num_poems:
        return self.num_poems
    else:
        if self.database:
            # Use the SPARQL Query of class "CountPoeticWorks" (set as
attribute of this class)
            self.sparql_num_poems.execute(self.database)
            # normally, the result would be a list containing a single
string value
            # by supplying a mapping to the simplify method the value
bound to the variable "count"
            # can be cast to an integer
            mapping = {"count": {"datatype": "int"}}
            self.num_poems =
self.sparql_num_poems.results.simplify(mapping=mapping)[0]
            return self.num_poems
        else:
            raise Exception("Database Connection not available.")
```

Fig. 45: Implementation of the method `get_num_poems`

By calling the method `execute` in the line `self.sparql_num_poems.execute(self.database)` the SPARQL query is executed on the database object “database”. The returned results are then transformed by evoking the method `simplify` in the line `self.num_poems = self.sparql_num_poems.results.simplify(mapping=mapping)[0]`, which returns the number of poems in the Knowledge Graph cast to an integer. This is specified by the mapping that is passed to the `simplify` method. This way several attributes of a corpus can be fetched and then output, for example, in the API.

The API is implemented with the module “api.py” using the Python package *flask*. The documentation is generated with the package *apidoc*, that allows to generate an OpenAPI specification from the docstring annotations. For example, the endpoint

`/corpora/{corpusname}` is implemented with the following function (cf. Fig. 46). The URL of the endpoint is defined in the function decoration `@api.route [...]`, The docstring following the function declaration is used in the OpenAPI specification; the data is retrieved and transformed in the line that contains “`corpora.corpora[corpusname].get_metadata(include_metrics=True)`”.[278]

```
@api.route("/corpora/<path:corpusname>", methods=["GET"])
def get_corpus_metadata(corpusname:str):
    """Corpus Metadata

    Args:
        corpusname: ID/name of the corpus, e.g. "postdata".

    ---
    get:
        summary: Corpus Metadata
        description: Returns metadata on a corpus. Unlike the DraCor
API the response does not contain information
            on included corpus items (poems). Use the endpoint
``/corpora/{corpusname}/poems`` instead.
        operationId: get_corpus_metadata
        parameters:
            -   in: path
                name: corpusname
                description: Name/ID of the corpus.
                required: true
                example: postdata
                schema:
                    type: string
        responses:
            200:
                description: Corpus metadata.
                content:
                    application/json:
                        schema: CorpusMetadata
            404:
                description: No such corpus. Parameter ``corpusname``
is invalid. A list of valid values can be
                    retrieved via the ``/corpora`` endpoint.
                content:
```

[278] The `get_metadata` method is implemented as a method of the class `PostdataCorpus`, cf. <151>.

```
                   text/plain:
                       schema:
                           type: string
   """
   if corpusname in corpora.corpora:
       metadata =
corpora.corpora[corpusname].get_metadata(include_metrics=True)

       # Validate response with schema "CorpusMetadata"
       schema = CorpusMetadata()
       schema.load(metadata)

       return jsonify(schema.dump(metadata))

   else:
       return Response(f"No such corpus: {corpusname}", status=404,
                       mimetype="text/plain")
```

Fig. 46: Implementation of the endpoint `/corpora/{corpusname}`

The API and the underlying modules were developed in a rapid prototyping process, resulting in code that is somewhat unpolished and is less geared towards efficiency, but transparency. Efforts have been made to ensure the re-use of components outside the API (which might be a replacement for a designated API wrapper) and in documenting the internals, especially the SPARQL queries. To make them self-explanatory, the queries themselves feature an `explain` method that returns a description of the query. A first testing of the prototype revealed some performance issues, which occur due to fetching properties from the triple store in several turns. These must be resolved before a reliable version can be deployed on a server.

We are considering evolving the prototype and including a second source, i.e., the "German Poetry Corpus"[279] (Haider, Egger 2019; Haider 2021), which will probably have to include a designated data store that is not RDF based. The prototype is a solid foundation for such an endeavor, because we introduced general and implementation specific classes for the core entities (corpus, poem, author). Such an abstraction layer should make it possible to integrate systems that rely on different technology stacks altogether and thus realize a joint Programmable Poetry Corpora System.

[279] Repository on GitHub: https://github.com/tnhaider/DLK

7. Some Lessons Learned

This report covers the initial development cycles in prototyping on the Programmable Corpora concept. In the course of the CLS INFRA project, further development cycles will follow, which will be documented and reflected in a final report (Deliverable D7.4) in 2025. At this point, we would like to conclude by noting four learnings that will also be of great importance for further development.

- The DraCor prototype we have reported on has already strikingly demonstrated the potentials that the style of a "network-based software architecture" offers for a future infrastructural ecosystem for CLS. Making the distributed resources and applications of CLS interconnectable via APIs proves to be a promising approach suitable for beginners in CLS as well as expert users. In this way, corpora that follow the concept of Programmable Corpora can be, for example, provided with beginner-friendly front-ends as well as be addressed directly from common programming languages (possibly via appropriate API wrappers). The transferability and combinability of resources and applications in a distributed system following the style of a "network-based software architecture", as we have showcased in section 7, seems particularly attractive to us. Further research and development in Work Package 7 will focus on these aspects and explore how other CLS tools can be connected to the DraCor prototype and how the approach of the DraCor prototype can be transferred to other systems. In this way, the first building blocks of an evolving infrastructural ecosystem for CLS will emerge.
- APIs, as has been shown, are at the core of the research that is delivered in Work Package 7. In doing so, we have followed the ecological approach of a "natural growth" of APIs in an environment of specific demands, as outlined by Aaron Swartz. In this sense, the endpoints of the DraCor API have "grown" first from the structure of the corpus documents ("document-based growth"), second from the needs of concrete research projects ("research-driven growth"), third from the requirements of the front-end ("front-end-oriented growth"). This development approach, which could be called "building a tactical infrastructure", has some major advantages: User needs can be met quickly, and no resources are wasted on unused functions. At the same time, as noted in section 6, such a development approach lacks systematicity and genericity. To counteract this and to push the development in the direction of a long-term infrastructure, the alignment with a systematic, ontology-based domain model for CLS is necessary.
- Documentation is crucial to ensure the connectivity of the components in a network-based software architecture. Standards for documentation, such as OpenAPI, should be as widespread as possible. At the same time, such documentation usually covers only the technical aspects. To make APIs semantic (and thus conceptually transparent), again, the

alignment with an ontology-based domain model for CLS via Linked Open Data is required. Programmable Corpora and Semantic Web need to unite.

- While the development of the DraCor prototype has been supported for some time by funding from the European Commission under CLS INFRA, central to the development of this infrastructure prototype is the involvement of a vibrant community of researchers who integrate their corpora into DraCor and use DraCor in their research. We would like to point out the crucial importance of this community-based prototyping approach, because this approach leads to the fact that the “natural growth” Aaron Swartz was talking about can actually take place in a real-life environment of scientific practice. – Last but not least, we would like to thank this community.

References

Almas, Bridget, Hugh Cayless, Thibault Clérice, Vincent Jolivet, Pietro Maria Liuzzo, Jonathan Robie, Matteo Romanello, and Ian Scott. “Distributed Text Services (DTS): A Community-Built API to Publish and Consume Text Collections as Linked Data.” *Journal of the Text Encoding Initiative* (2023). https://doi.org/10.4000/jtei.4352.

Bastian, Mathieu, Sebastien Heymann, and Mathieu Jacomy. “Gephi: An Open Source Software for Exploring and Manipulating Networks.” *International AAAI Conference on Weblogs and Social Media*. 2009. https://gephi.org/publications/gephi-bastian-feb09.pdf.

Börner, Ingo, Peer Trilcke, Carsten Milling, Frank Fischer, and Henny Sluyter-Gäthje. “Dockerizing DraCor. A Container-Based Approach to Reproducibility in Computational Literary Studies.” In *DH2023. Book of Abstracts*. Graz, [submitted].

Budde, Reinhard, Karlheinz Kautz, Karin Kuhlenkamp, and Heinz Züllighoven. “What Is Prototyping?” *Information Technology & People* 6.2/3 (1992): 89–95. https://doi.org/10.1108/EUM0000000003546.

Burnard, Lou, Christof Schöch, and Carolin Odebrecht. “In Search of Comity: TEI for Distant Reading.” *Journal of the Text Encoding Initiative* 14 (2021). https://doi.org/10.4000/jtei.3500.

Cayless, Hugh, and Raffaele Viglianti. “CETEIcean: TEI in the Browser.” *Proceedings of Balisage: The Markup Conference 2018*. Washington, DC, 2018. https://doi.org/10.4242/BalisageVol21.Cayless01.

Chiarcos, Christian, John McCrae, Philipp Cimiano, and Christiane Fellbaum. “Towards Open Data for Linguistics: Linguistic Linked Data.” *New Trends of Research in Ontologies and Lexical Resources: Ideas, Projects, Systems*, ed. by Alessandro Oltramari, Piek Vossen, Lu Qin, and Eduard Hovy. Berlin, Heidelberg: Springer, 2013: 7–25. https://doi.org/10.1007/978-3-642-31782-8_2.

Ďurčo, Matej, Vera Maria Charvat, Ingo Börner, Michał Mrugalski, and Carolin Odebrecht. “CLS INFRA D6.1: Inventory of Existing Data Sources and Formats.” Zenodo, 2022. https://doi.org/10.5281/zenodo.7520287.

Fielding, Roy Thomas. *Architectural Styles and the Design of Network-Based Software Architectures*. University of California, 2000. https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf.

Fischer, Frank, Ingo Börner, Mathias Göbel, Angelika Hechtl, Christopher Kittel, Carsten Milling, and Peer Trilcke. “Programmable Corpora: Introducing DraCor, an Infrastructure for the Research on European Drama.” In *DH2019. Book of Abstracts*. Utrecht, 2019. https://doi.org/10.5281/ZENODO.4284002.

Fischer, Frank, Anika Schultz, Christopher Kittel, Elisa Beshero-Bondar, Steffen Martus, Peer Trilcke, Jana Wolf, et al. “Brecht Beats Shakespeare! A Card-Game Intervention Revolving Around the Network Analysis of European Drama.” In *DH2018. Book of Abstracts*. Mexico City, 2018: 595–96.

Floyd, Christiane. “A Systematic Look at Prototyping.” *Approaches to Prototyping*, ed. by Reinhard Budde and Karin Kuhlenkamp. Berlin: Springer, 1984: 1–18.

Gavin, Michael. *Literary Mathematics: Quantitative Theory for Textual Studies*. Stanford, California: Stanford University Press, 2023.

Giovannini, Luca, Daniil Skorinkin, Peer Trilcke, Ingo Börner, Frank Fischer, Julia Dudar, Carsten Milling, and Petr Pořízka. "Distributed Corpus Building in Literary Studies: The DraCor Example." In *DH2023. Book of Abstracts*. Graz, [submitted].

Hagberg, Aric A., Daniel A. Schult, and Pieter J. Swart. "Exploring Network Structure, Dynamics, and Function Using NetworkX." *Proceedings of the 7th Python in Science Conference (SciPy2008)*, ed. by Gäel Varoquaux, Travis Vaught, and Jarrod Millman. Pasadena, CA, 2008: 11–15

Haider, Thomas. "Metrical Tagging in the Wild: Building and Annotating Poetry Corpora with Rhythmic Features." *Proceedings of the European Association for Computational Linguistics*. ArXiv. 2021. https://doi.org/10.48550/arXiv.2102.08858.

Haider, Thomas, and Steffen Eger. "Semantic Change and Emerging Tropes In a Large Corpus of New High German Poetry." *Proceedings of the 1st International Workshop on Computational Approaches to Historical Language Change*. ArXiv. 2019 https://doi.org/10.48550/arXiv.1909.12136

Marcus, Solomon. *Poetica Matematică*. Bucureşti: Ed. Acad. R. S. România, 1970.

———. *Mathematische Poetik*. Frankfurt (Main): Athenäum, 1973.

Massé, Mark. *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. Beijing, Köln: O'Reilly, 2012.

Montani, Ines, Matthew Honnibal, Sofie Van Landeghem, Adriane Boyd, Henning Peters, Paul O'Leary McCann, et al. "Explosion/SpaCy: V3.5.0: New CLI Commands, Language Updates, Bug Fixes and Much More." Zenodo. 2023. https://doi.org/10.5281/zenodo.7553910.

Moretti, Franco. *Distant Reading*. London, New York: Verso, 2013.

———. *Network Theory, Plot Analysis*. Stanford Literary Lab. 2011. https://litlab.stanford.edu/LiteraryLabPamphlet2.pdf

Mrugalski, Michał, Carolin Odebrecht, Vera Charvat, Ingo Börner, and Matej Durco. "CLS INFRA D5.1. Review of the Data Landscape." Zenodo. 2022. https://doi.org/10.5281/zenodo.6861022.

Nadareishvili, Irakli, Ronnie Mitra, Matt McLarty, and Michael Amundsen. *Microservice Architecture: Aligning Principles, Practices, and Culture*. Beijing: O´Reilly, 2016.

Naumann, Justus D., and A. Milton Jenkins. "Prototyping: The New Paradigm for Systems Development." *MIS Quarterly* 6.3 (1982): 29–44. https://doi.org/10.2307/248654.

Niles, Rebecca, and Michael Poston. "Re-Modeling the Edition: Creating the Corpus of Folger Digital Texts." *Early Modern Studies after the Digital Turn*, ed. by Laura Estill, Diane Jakacki, and Michael Ullyot. Tempe,Toronto: Iter and the Arizona Center for Medieval and Renaissance Studies, 2016: 119–46.

Redick, Stacey J. L, and Eric M. Johnson. "Mediating the Shakespeare User's Digital Experience." *The Routledge Handbook of Shakespeare and Interface*, ed. by Clifford Werier and Paul Budra. New York: Routledge, 2022: 268–82.

Revzina, Olga G., and Isaak I. Revzin. “Nekotorye Matematicheskie Metody Analiza Dramaturgicheskogo Postroeniia.” *Tochnye Metody v Issledovaniakh Kul’tury i Iskusstva (Materialy k Simpoziumy)*, ed. by Nauchnyi sovet po kibernetike AN SSSR, Vserossiiskoe teatral’noe obshchestvo, and NII Kul’tury Minesterstva RSFSR. Moscow, 1971. Vol. 2: 291–300.

Ruecker, Stan. “A Brief Taxonomy of Prototypes for the Digital Humanities.” *Scholarly and Research Communication* 6.2 (2015). https://doi.org/10.22230/src.2015v6n2a222.

Sapogov, Vyacheslav Alexandrovich. “Nekotorye harakteristiki dramaturgičeskogo postroeniâ komedii A. N. Ostrovskogo »Les« [Some Characteristics of the Dramatic Construction of A. N. Ostrovsky’s Comedy »The Forest«].” *A. N. Ostrovskij i russkaâ literatura [A. N. Ostrovsky and Russian Literature]*, ed. by Vyacheslav Alexandrovich Sapogov. Kostroma: Âroslavskij pedagogičeskij institut [Yaroslavl State Pedagogical Institute], 1974: 60–69.

Schöch, Christof, Frédéric Döhl, Achim Rettinger, Evelyn Gius, Peer Trilcke, Peter Leinen, Fotis Jannidis, Maria Hinzmann, and Jörg Röpke. “Abgeleitete Textformate: Text und Data Mining mit urheberrechtlich geschützten Textbeständen.” *Zeitschrift für digitale Geisteswissenschaften* (2020). https://doi.org/10.17175/2020_006.

Schöch, Christof, Evgeniia Fileva, and Julia Dudar. “CLS INFRA D3.1 Baseline Methodological User Needs Analysis.” Zenodo. 2022. https://doi.org/10.5281/zenodo.6389333.

Schöch, Christof, Roxana Patras, Tomaž Erjavec, and Diana Santos. “Creating the European Literary Text Collection (ELTeC): Challenges and Perspectives.” *Modern Languages Open* 1 1 (2021). https://doi.org/10.3828/mlo.v0i0.364.

Swartz, Aaron. “Aaron Swartz’s A Programmable Web: An Unfinished Work.” *Synthesis Lectures on the Semantic Web: Theory and Technology* 3, no. 2 (2013): 1–64. https://doi.org/10.1007/978-3-031-79444-5.

Tavolato, Paul, and Karl Vincena. “A Prototyping Methodology and Its Tool.” *Approaches to Prototyping*, ed. by Reinhard Budde, Karin Kuhlenkamp, Lars Mathiassen, and Heinz Züllighoven. Berlin, Heidelberg: Springer, 1984: 434–46. https://doi.org/10.1007/978-3-642-69796-8_38.

Trilcke, Peer. “Social Network Analysis (SNA) als Methode einer textempirischen Literaturwissenschaft.” *Empirie in Der Literaturwissenschaft*, ed. by Philip Ajouri, Katja Mellmann, and Christoph Rauen. Münster: Mentis, 2013: 201–47.

Trilcke, Peer, Frank Fischer, Mathias Göbel, Dario Kampkaspar, and Christoph Kittel. “Network Dynamics, Plot Analysis: Approaching the Progressive Structuration of Literary Texts.” *DH2017. Book of Abstracts*. Montreal, 2017: 437–41. https://dh2017.adho.org/abstracts/071/071.pdf.

Trilcke, Peer, Christopher Kittel, Nils Reiter, Daria Maximova, and Frank Fischer. “Opening the Stage – A Quantitative Look at Stage Directions in German Drama.” In *DH2020. Book of Abstracts*, ed. by Laura Estill and Jennifer Guiliano, 2020. https://dh2020.adho.org/wp-content/uploads/2020/07/337_OpeningtheStageAQuantitativeLookatStageDirectionsinGermanDrama.html.

Trilcke, Peer, Evgeniya Ustinova, Ingo Börner, Frank Fischer, and Carsten Milling. “Detecting Small Worlds in a Corpus of Thousands of Theatre Plays. A DraCor Study in Comparative

Literary Network Analysis." *Computational Drama Analysis: Achievements and Opportunities*, ed. by Melanie Andresen, Nils Reiter, Benjamin Krautter, and Janis Pagel, [27 pages]. Berlin: De Gruyter, [in press]. Preprint [Conference Version, Cologne 2022]: https://github.com/dracor-org/small-world-paper/raw/conference-version/Detecting_Small_World_Networks__in_a_Huge_Multilingual_Corpus_of_Theater_Plays.pdf.

Wendell, Inna. *A Statistical Analysis of Genre Dynamics: Evolution of the Russian Five-Act Comedy in Verse in the Eighteenth and Nineteenth Centuries*. UCLA, 2021. https://escholarship.org/uc/item/9rr5k9p7.

Wiedmer, Nathalie, Janis Pagel, and Nils Reiter. "Romeo, Freund Des Mercutio: Semi-Automatische Extraktion von Beziehungen zwischen Dramatischen Figuren." *DHd2020. Book of Abstracts*. Paderborn, 2020: 194–200. https://doi.org/10.5281/zenodo.4621778.

Yarkho, Boris I. "Speech Distribution in Five-Act Tragedies (A Question of Classicism and Romanticism)." *Journal of Literary Theory* 13.1 (2019): 13–76. https://doi.org/10.1515/jlt-2019-0002.

Code References

<1> https://github.com/dracor-org/gerdracor/blob/3dc874101e2d10d687510aeb5ff8a907331843c1/tei/lessing-emilia-galotti.xml#L11

<2> https://github.com/dracor-org/gerdracor/blob/3dc874101e2d10d687510aeb5ff8a907331843c1/tei/lessing-emilia-galotti.xml#L122

<3>https://github.com/dracor-org/romdracor/blob/e80db098a89e842174cf76dd0ffb56b5449d351d/tei/ad_lat.xml#L173-L175

<4> https://github.com/dracor-org/dracor-api/blob/ac5a6816ce68efc845aef19a80558f6f956b3cc8/modules/metrics.xqm#L49-L118

<5> https://github.com/dracor-org/dracor-api/blob/ac5a6816ce68efc845aef19a80558f6f956b3cc8/modules/util.xqm#L263-L286

<6> https://github.com/dracor-org/dracor-api/blob/ac5a6816ce68efc845aef19a80558f6f956b3cc8/modules/util.xqm#L103-L116

<7> https://github.com/dracor-org/dracor-api/tree/645ed31d091f4ea57b5513be2b88a9340102b0f6/modules

<8> https://github.com/dracor-org/dracor-api/blob/e03b629bc74cfb10299213fb17abfabfd063a666/modules/api.xqm

<9> https://github.com/dracor-org/dracor-api/blob/e03b629bc74cfb10299213fb17abfabfd063a666/modules/api.xqm#L845-L873

<10> https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/api.xqm#L65-L87

<11> https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/api.xqm#L163-L190

<12 https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/api.xqm#L143-L161

<13 https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/util.xqm#L401-L417

<14 https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/util.xqm#L436-L475

<15> https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/api.xqm#L197

<16> https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/api.xqm#L202

<17 https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/api.xqm#L192-L220

<18> https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/api.xqm#L243-L245

<19> https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/api.xqm#L213-L217

<20> https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/api.xqm#L663

<21> function `dutil:get-corpus-metadata` https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/util.xqm#L516-L626

<22> function `api:get-corpus-metadata` https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/api.xqm#L667-L685

<23> https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/util.xqm#L595-L598

<24> function api:corpus-meta-data-csv https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/api.xqm#L687-L701

<25> function `api:get-corpus-meta-data-csv` https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/api.xqm#L667-L685

<26> https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/api.xqm#L22-L63

<27> function `api:play-info` https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/api.xqm#L764-L786

<28> function `dutil:get-play-info` https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/util.xqm#L897-L1014

<29> function `api:play-metrics` https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/api.xqm#L821-L843

<30> function `dutil:get-play-metrics` https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/util.xqm#L1016-L1072

<31> function `api:play-rdf` https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/api.xqm#L940-L961

<32> function `api:play-tei` https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/api.xqm#L845-L873

<33> function `dutil:get-doc` https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/util.xqm#L77-L91

<34> function `dutil:get-segments` https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/util.xqm#L263-L286

<35> function `api:cast-info` https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/api.xqm#L1399-L1421

<36> function `dutil:dutil:cast-info` https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/util.xqm#L1074-L1136

<37> function `api:cast-info-csv-ext` https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/api.xqm#L1457-L1464

<38> function `api:cast-info-csv` https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/api.xqm#L1423-L1455

<39> https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/api.xqm#L1433

<40> function `api:spoken-text` https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/api.xqm#L1565-L1611

<41> function `dutil:get-speech-filtered` https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/util.xqm#L161-L235

<42> function `dutil:get-speech` https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/util.xqm#L118-L136

<43> function `api:get-spoken-text-by-character` https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/api.xqm#L1638-L1647

<44> function `local:get-text-by-character` https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/api.xqm#L1613-L1636

<45> function `api:spoken-text-by-character-csv` https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/api.xqm#L1685-L1716

<46> function `api:spoken-text-by-character-json` https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/api.xqm#L1666-L1683

<47> https://github.com/dracor-org/gerdracor/blob/bfadf6b5844d4e05ea0501898a23c21f71c10cb3/tei/lessing-emilia-galotti.xml#L159-L176

<48> function `api:stage-directions` https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/api.xqm#L1718-L1741

<49> https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/api.xqm#L1738-L1739

<50> function `api:stage-directions-with-speakers` https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/api.xqm#L1743-L1769

<51> function `api:play-metrics` https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/api.xqm#L821-L843

<52> function `dutil:get-play-metrics` https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/util.xqm#L1016-L1072

<53> function `api:networkdata-csv` https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/api.xqm#L963-L1013

<54> function `api:networkdata-gexf` https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/api.xqm#L1037-L1110

<55> function `api:networkdata-graphml` https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/api.xqm#L1112-L1194

<56> https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/api.xqm#L985-L997

<57> https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/api.xqm#L1058-L1070

<58> https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/api.xqm#L1133-L1145

<59> https://github.com/dracor-org/dracor-api/blob/b78724b3ab2009901b52818ad9af741e95976042/modules/metrics.xqm#L62-L72

<60> https://github.com/dracor-org/gerdracor/blob/bfadf6b5844d4e05ea0501898a23c21f71c10cb3/tei/lessing-emilia-galotti.xml#L90-L94

<61> function `api:relations-csv` https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/api.xqm#L1197-L1245

<62> function `dutil:get-relations` https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/util.xqm#L1144-L1180

<63> https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/api.xqm#L1247-L1318

<64> https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/util.xqm#L951

<65> https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/api.xqm#L1320-L1397

<66> https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/api.xqm#L1771-L1792

<67> function `dutil:get-plays-with-character` https://github.com/dracor-org/dracor-api/blob/3215fde0c34cc51a51386728775cbb3c058ae06f/modules/util.xqm#L1182-L1206

<68> https://github.com/dracor-org/gerdracor/blob/d23a93d9fa0e4eb53a580904ac5d01c8b8f8037c/tei/goethe-faust-eine-tragoedie.xml#L75-L77

<69> function `metrics:get-network-metrics` https://github.com/dracor-org/dracor-api/blob/b78724b3ab2009901b52818ad9af741e95976042/modules/metrics.xqm#L49-L118

<70> https://github.com/dracor-org/dracor-metrics/blob/fc3d3d3f6e1185c447678dc8113dd6c4a0a58141/main.py#L18-L36

<71> https://github.com/dracor-org/dracor-metrics/blob/fc3d3d3f6e1185c447678dc8113dd6c4a0a58141/main.py#L38

<72> https://github.com/dracor-org/dracor-metrics/blob/fc3d3d3f6e1185c447678dc8113dd6c4a0a58141/main.py#L71

<73> https://github.com/dracor-org/dracor-metrics/blob/fc3d3d3f6e1185c447678dc8113dd6c4a0a58141/main.py#L72-L73

<74> https://github.com/dracor-org/dracor-metrics/blob/fc3d3d3f6e1185c447678dc8113dd6c4a0a58141/main.py#L74

<75> https://github.com/dracor-org/dracor-metrics/blob/fc3d3d3f6e1185c447678dc8113dd6c4a0a58141/main.py#L75

<76> https://github.com/dracor-org/dracor-metrics/blob/fc3d3d3f6e1185c447678dc8113dd6c4a0a58141/main.py#L39

<77> https://github.com/dracor-org/dracor-metrics/blob/fc3d3d3f6e1185c447678dc8113dd6c4a0a58141/main.py#L40

<78> React component “TopNav” https://github.com/dracor-org/dracor-frontend/blob/0db19a0b59f8253bccf688dcec70539d64099752/src/components/TopNav.js

<79> React component “Footer”

https://github.com/dracor-org/dracor-frontend/blob/0db19a0b59f8253bccf688dcec70539d64099752/src/components/Footer.js

<80> function `fetchInfo` in App.tsx https://github.com/dracor-org/dracor-frontend/blob/7af2edc988813fab99f4d1fdcb7c15a290558760/src/App.tsx#L27-L43

<81> https://github.com/dracor-org/dracor-frontend/blob/ea545e97e5eb654b3730a45925703f32f1648212/src/types.ts#L1-L6

<82> https://github.com/dracor-org/dracor-frontend/blob/0db19a0b59f8253bccf688dcec70539d64099752/src/components/Footer.js#L74-L96

<83> https://github.com/dracor-org/dracor-frontend/blob/0db19a0b59f8253bccf688dcec70539d64099752/src/components/Home.js

<84> https://github.com/dracor-org/dracor-frontend/blob/e4377a561728fc961c8a9412a136dd46f692840d/src/components/Corpora.js

<85> https://github.com/dracor-org/dracor-frontend/blob/e4377a561728fc961c8a9412a136dd46f692840d/src/components/Corpora.js#L79-L89

<86> https://github.com/dracor-org/dracor-frontend/blob/e4377a561728fc961c8a9412a136dd46f692840d/src/components/CorpusCard.js

<87> https://github.com/dracor-org/dracor-frontend/blob/10498ca94530452de1e1b8306301577226c0c361/src/components/Corpus.js#L25-L45

<88> https://github.com/dracor-org/dracor-frontend/blob/10498ca94530452de1e1b8306301577226c0c361/src/components/Corpus.js

<89> https://github.com/dracor-org/dracor-frontend/blob/7af2edc988813fab99f4d1fdcb7c15a290558760/src/components/CorpusIndex.js

<90> https://github.com/dracor-org/dracor-frontend/blob/7af2edc988813fab99f4d1fdcb7c15a290558760/src/components/CorpusIndex.js#L104

<91> https://github.com/dracor-org/dracor-frontend/blob/7af2edc988813fab99f4d1fdcb7c15a290558760/src/components/CorpusIndex.js#L115-L122

<92> https://github.com/dracor-org/dracor-frontend/blob/a553f004df3ede7de1cc26be9dd50b8942fe794d/src/components/Play.js

<93> https://github.com/dracor-org/dracor-frontend/blob/a553f004df3ede7de1cc26be9dd50b8942fe794d/src/components/Play.js#L51-L70

<94> https://github.com/dracor-org/dracor-frontend/blob/a553f004df3ede7de1cc26be9dd50b8942fe794d/src/components/Play.js#L114-L166

<95> https://github.com/dracor-org/dracor-frontend/blob/a553f004df3ede7de1cc26be9dd50b8942fe794d/src/components/Play.js#L97

<96> https://github.com/dracor-org/dracor-frontend/blob/e4377a561728fc961c8a9412a136dd46f692840d/src/components/PlayDetailsHeader.js

<97> https://github.com/dracor-org/dracor-frontend/blob/dab5c4c6e13b79bf06a176d32ba07b7ee4b080dc/src/components/AuthorInfo.js

<98> https://github.com/dracor-org/dracor-frontend/blob/dab5c4c6e13b79bf06a176d32ba07b7ee4b080dc/src/components/AuthorInfo.js#L19

<99> https://github.com/dracor-org/dracor-frontend/blob/dab5c4c6e13b79bf06a176d32ba07b7ee4b080dc/src/components/AuthorInfo.js#L18-L54

<100> https://github.com/dracor-org/dracor-frontend/blob/0db19a0b59f8253bccf688dcec70539d64099752/src/components/NetworkGraph.js

<101> https://github.com/dracor-org/dracor-frontend/blob/0db19a0b59f8253bccf688dcec70539d64099752/src/components/NetworkGraph.js#L29-L39

<102> https://github.com/dracor-org/dracor-frontend/blob/9933527844f6187e01a20ad06b8497141445f43c/src/network.js#L3-L36

<103> https://github.com/dracor-org/dracor-frontend/blob/9933527844f6187e01a20ad06b8497141445f43c/src/network.js#L38-L59

<104> https://github.com/dracor-org/dracor-frontend/blob/27c7fa7edfbabe9512c1232f6d9d33ee0b20fa3e/src/components/CastList.js

<105> https://github.com/dracor-org/dracor-frontend/blob/0db19a0b59f8253bccf688dcec70539d64099752/src/components/RelationsGraph.js

<106> https://github.com/dracor-org/dracor-frontend/blob/0db19a0b59f8253bccf688dcec70539d64099752/src/components/RelationsGraph.js#L37-L63

<107> https://github.com/dracor-org/dracor-frontend/blob/0db19a0b59f8253bccf688dcec70539d64099752/src/components/RelationsGraph.js#L14-L21

<108> https://github.com/dracor-org/dracor-frontend/tree/0db19a0b59f8253bccf688dcec70539d64099752/src/components/SpeechDistribution

<109> https://github.com/dracor-org/dracor-frontend/blob/0db19a0b59f8253bccf688dcec70539d64099752/src/components/SpeechDistribution/Sapogov.js

<110> https://github.com/dracor-org/dracor-frontend/blob/0db19a0b59f8253bccf688dcec70539d64099752/src/components/SpeechDistribution/Sapogov.js#L74-L88

<111> https://github.com/dracor-org/dracor-frontend/blob/0db19a0b59f8253bccf688dcec70539d64099752/src/components/SpeechDistribution/Yarkho.js

<112> https://github.com/dracor-org/dracor-frontend/blob/0db19a0b59f8253bccf688dcec70539d64099752/src/components/SpeechDistribution/Yarkho.js#L86-L132

<113> https://github.com/innawendell/European_Comedy/blob/a0ffe348031579278990cb29a14c799985adbdfb/Russian_Comedies/Russian_Comedies.tsv

<114> https://github.com/innawendell/European_Comedy/blob/466c2d2bed597ee7dd850b61445554d8fc173c30/TAGS_EXPLANATION.md#2-russian-tei-files

<115> https://github.com/innawendell/player/blob/85a4173ea41146f0ab852cf7b328358575e0280a/player/russian_tei_functions.py

<116> https://github.com/innawendell/European_Comedy/blob/7b96d1e43d31acab85c8431a915039082813c126/Russian_Comedies/TEI_files/R_1.xml#L1036-L1047

<117> https://github.com/lehkost/dramavis/pull/16/commits/db960b36d305ee3a3275335209d10d57afdf6e60

<118> https://github.com/dracor-org/dracor-frontend/blob/0db19a0b59f8253bccf688dcec70539d64099752/src/components/SpeechDistribution/TrilckeFischer.js

<119> https://github.com/dracor-org/dracor-frontend/blob/0db19a0b59f8253bccf688dcec70539d64099752/src/components/SpeechDistribution/TrilckeFischer.js#L31-L40

<120> https://github.com/dracor-org/dracor-frontend/blob/0db19a0b59f8253bccf688dcec70539d64099752/src/components/SpeechDistribution/TrilckeFischer.js#L14-L29

<121> https://github.com/dracor-org/dracor-frontend/blob/0db19a0b59f8253bccf688dcec70539d64099752/src/components/SpeechDistribution/TrilckeFischer.js#L53-L100

<122> https://github.com/dracor-org/dracor-frontend/blob/a553f004df3ede7de1cc26be9dd50b8942fe794d/src/components/Play.js#L102

<123> https://github.com/dracor-org/dracor-frontend/blob/a553f004df3ede7de1cc26be9dd50b8942fe794d/src/components/Play.js#L138

<124> https://github.com/dracor-org/dracor-frontend/blob/1a108a52f43536bdab7269850d9cfbcb8e7ce64f/src/components/TEIPanel.js

<125> https://github.com/dracor-org/dracor-frontend/blob/0db19a0b59f8253bccf688dcec70539d64099752/src/components/SourceInfo.js

<126> https://github.com/dracor-org/dracor-frontend/blob/a553f004df3ede7de1cc26be9dd50b8942fe794d/src/components/Play.js#L140

<127> https://github.com/dracor-org/dracor-frontend/blob/27c7fa7edfbabe9512c1232f6d9d33ee0b20fa3e/src/components/Segments.tsx

<128> https://github.com/dracor-org/dracor-frontend/blob/6cff1feb7c2ceab9df48cfe70a994fd8d63bf785/src/components/DownloadLinks.js

<129> https://github.com/dracor-org/dracor-frontend/blob/6cff1feb7c2ceab9df48cfe70a994fd8d63bf785/src/components/DownloadLinks.js#L18-L29

<130> https://github.com/dracor-org/dracor-frontend/blob/6cff1feb7c2ceab9df48cfe70a994fd8d63bf785/src/components/DownloadLinks.js#L134

<131> https://github.com/ingoboerner/folger-shakespeare-openapi/blob/a1bb3b82c777d4dc5350bc09362b86f8eb444c83/folger-shakespeare-api-doc.ipynb

<132> https://github.com/ingoboerner/folger-shakespeare-openapi/blob/47800fb3a6ccc0de0f8d9281d3ac4559da7caa99/openapi.yaml

<133> https://github.com/linhd-postdata/knowledge-graph-queries/blob/e742c9b34dab10cbccd0da502ac69afe7e382c9a/knowledge_graph_queries/openapi/openapi.yml

<134> https://github.com/linhd-postdata/knowledge-graph-queries/blob/102d9a7a70e092340e6df30ee7db9e9306478153/knowledge_graph_queries/app.py

<135> https://github.com/linhd-postdata/knowledge-graph-queries/blob/e742c9b34dab10cbccd0da502ac69afe7e382c9a/knowledge_graph_queries/core.py

<136> https://github.com/linhd-postdata/knowledge-graph-queries/blob/e742c9b34dab10cbccd0da502ac69afe7e382c9a/knowledge_graph_queries/core.py#L10-L21
<137>https://github.com/linhd-postdata/knowledge-graph-queries/blob/e742c9b34dab10cbccd0da502ac69afe7e382c9a/knowledge_graph_queries/core.py#L211-L219

<138> https://github.com/linhd-postdata/knowledge-graph-queries/blob/e742c9b34dab10cbccd0da502ac69afe7e382c9a/knowledge_graph_queries/core.py#L188-L208

<139> https://github.com/linhd-postdata/knowledge-graph-queries/blob/e742c9b34dab10cbccd0da502ac69afe7e382c9a/knowledge_graph_queries/queries.py#L2-L77

<140> https://github.com/linhd-postdata/knowledge-graph-queries/blob/e742c9b34dab10cbccd0da502ac69afe7e382c9a/knowledge_graph_queries/queries.py

<141> https://github.com/linhd-postdata/knowledge-graph-queries/blob/e742c9b34dab10cbccd0da502ac69afe7e382c9a/knowledge_graph_queries/queries.py#L2-L30

<142> https://github.com/linhd-postdata/knowledge-graph-queries/blob/2d1805c48b44e2cce38dd948e166030103dad571/docker-compose.yml

<143> https://github.com/linhd-postdata/postdata-stardog/blob/bafa3a8d42814400ae44d326bd43c71852ac58ac/Dockerfile

<144> https://github.com/dh-network/postdata-2-dracor-api/blob/e614a71b79572b372afa5f70582287a48af2ca80/query_tryouts_poetry_lab.ipynb

<145> https://github.com/dh-network/postdata-2-dracor-api/blob/ddf8e1af61cae6f7c6692092723c9d1f19350ddb/openapi.yaml

<146> https://github.com/linhd-postdata/knowledge-graph-queries/blob/e742c9b34dab10cbccd0da502ac69afe7e382c9a/knowledge_graph_queries/openapi/openapi.yml#L53

<147> https://github.com/dh-network/postdata-2-dracor-api/blob/d7b4c3bcbb4abe0355d5c8169788ca9e31c20f34/pd_stardog_queries.py

<148> https://github.com/dh-network/postdata-2-dracor-api/blob/1999ba51cfa2b35b1e35d24609466168d106bd38/sparql.py

<149> https://github.com/dh-network/postdata-2-dracor-api/blob/1999ba51cfa2b35b1e35d24609466168d106bd38/sparql.py#L528-L600

<150> https://github.com/dh-network/postdata-2-dracor-api/commit/ddf8e1af61cae6f7c6692092723c9d1f19350ddb

<151> https://github.com/dh-network/postdata-2-dracor-api/blob/5e41840e45b127d201e8a9580d4daa0e0ec81399/pd_corpus.py#L339-L374

<152> https://github.com/dracor-org/dracor-frontend/blob/0db19a0b59f8253bccf688dcec70539d64099752/src/components/PlayMetrics.js