University of Groningen

Making reading in a second language more enjoyable

Supervisor
Dr. Mircea Lungu
Second Supervisor
George Digkas

Jorrit Oosterhof
S2528312

Faculty of Mathematics and Natural Sciences

August 3, 2016

Contents

Chapter 1

Introduction

When learning a second language, one usually follows a learning program, which includes studying lists of words, reading standard texts and doing exercises. Learning a second language is much more enjoyable if one enjoys reading materials in the new language. Articles that are not too long, written about topics the student likes could be a good way to keep the reader motivated in reading more and thus learning the language.

However, if the student does not know what a word means he has to look it up in a dictionary, and this means losing time, time that could have been better spent reading some more second language materials. One of the hypotheses of this work is that it might be more productive if one could just tap a word and its translation would appear within the text. This would save the learner from having to search for the word in an external dictionary or manually having to type it into a service like Google Translate. Keeping track of all the translations a user needs opens many opportunities for developing automated solutions that improve language learning.

In this thesis we introduce the idea of an application which offers a platform where users can improve a language of their preference by reading articles they like in this language. The application allows users to read articles from their favorite news and blogs websites in the language they want to learn as opposed to static and inflexible texts. Users are able translate unknown words by the simple tapping of a word. The word is pronounced and its translation is inserted in the text. The idea is implemented in the form of the Zeeguu Reader for iOS. The Zeeguu Reader for iOS uses RSS feeds to deliver articles, however, the user should not need to know about RSS and therefore it should be sufficient to just enter the url of the website he wants to follow.

In addition to translating a single word, the application offers two other ways of translating words. Translating a word pair and translating sentences/longer pieces of text. These two options can help users to better understand what the text is saying.

Since machine translation is not yet perfect, it can happen that the first translation is not the correct one. Therefore, the Zeeguur Reader for iOS allows the reader to tap the translation which was inserted in the text and select an alternative translation or provide a new translation.

The Zeeguu Reader is part of the Zeeguu platform. The Zeeguu platform is a collection of apps and services to enhance language learning. To help the reader remember words better, the Zeeguu platform offers personalized exercises, based on the words that were translated earlier. The reader is presented with the English translation and the context in which the foreign word appeared and has to type the word that resulted in the given translation. To make these exercises easily accessible, the Zeeguu Reader allows users to make these exercises as if they were part of the app.

To validate whether the Zeeguu Reader for iOS is a good way to read text in a second language, a user study was conducted to answer the following research questions:

1. Do the learners like such an application that allows them to read texts on their iOS device?
    - Do users generally read on iPhone or iPad?
    - How long do they use the app?
2. Which of the features of the application are the most used by the users?
    - Do users like the interaction of tapping a word and having the translation inserted in the text?
    - Do users use the pronunciation feature?
    - Which way of translating words/text do users use the most?

1.1 Structure

This document describes the Zeeguu Reader, the user study and the results of the user study. The remainder of this document is structured as follows:

2. **Related work**
   This chapter briefly introduces work related to the problem and solution.
3. **System Design**
   This chapter describes the Zeeguu Reader for iOS, as a solution of the problem.

4. **User Study**
   This chapter describes the user study we conducted and the usage results we collected.

5. **Results**
   This chapter describes the results we found and answers the research questions that were introduced in the introduction.

6. **Conclusion and Future Work**
   This chapter gives our conclusions about the results of the user study and describes future work that can be done to improve the solution.

Chapter 2

Related work

The Zeeguu platform is not the only language learning service available. There are already several other services that can help you learn a second language. Two examples are Duolingo[1] and Babbel[2]. In contrast to the Zeeguu platform, these language learning services aim at learners who are starting to learn a second language, whereas the Zeeguu platform requires already some knowledge about the language, as the learner will read foreign texts.

There are also services available doing something similar to what the solution, being the Zeeguu Reader for iOS, is doing. One such service/device is the Kindle Reader.

2.1 Kindle Reader

The Kindle Reader offers users to tap/click a word to see a definition[3][4]. However, these methods are cumbersome and non-intuitive. In the first blog article (footnote 3), the author describes that tapping (or clicking on older devices) a word gives a definition of the word, usually in English. The author writes that one can use this feature to translate words by downloading an appropriate dictionary, for example English-Spanish, installing it on the device and setting it as the default dictionary. Tapping/Clicking a word then gives a definition from the newly installed dictionary and thus gives one or more translations.

The second blog article (footnote 4) describes a method of translating words

[1]https://www.duolingo.com

[2]https://www.babbel.com

[3]http://learnoutlive.com/how-to-use-your-kindle-to-study-a-foreign-language/

[4]http://ebookfriendly.com/translate-words-in-kindle-app/

from within the Kindle app. The Kindle app also offers the feature of 'defining' a word by highlighting it. The definition appears at the bottom of the screen. The author then describes that to translate a word, one taps the 'Google' link, to search the definition using Google, and adds 'to <language>' to the search term to get the translation.

Although these articles are a few years old, the Kindle app does not seem to offer a translation feature but rather a 'definition' feature[5]. Therefore, translating words using the Kindle reader or app does not offer the seamless translation experience that the Zeeguu Reader for iOS offers.

2.2 Lingua.ly

Another service that is similar to the Zeeguu Reader for iOS is Lingua.ly[6]. This service is very similar to the Zeeguu Reader for iOS and also similar to the Zeeguu platform. It incorporates the same ideas into its apps: reading texts from regular textbooks is boring. Users like reading texts about subjects they like. Users of Lingua.ly can tap a word and a translation pops up below the word. Lingua.ly uses three dictionaries to get the best translation and uses the translated words to build exercises to help the user learn the language.

A difference between the Zeeguu Reader for iOS and Lingua.ly is that the Zeeguu Reader for iOS inserts the translation into the text. Therefore, the user doesn't need to re-translate words as the translation is still there.

2.3 ALOE

Andrew Trusty and Khai N. Truong developed a FireFox extension that randomly translates words on webpages to the language that the user wants to learn. They did a user study to see whether the web can be used for vocabulary learning [1].

The extensions works as follows. The user just browses as usual, reads news from his favorites news site, uses social media, etcetera. All this is just in his native language. The ALOE extension randomly picks words within an active webpage and changes them to the corresponding words in his second language, for example French. As the user reads the webpage, he will encounter those French words a should be able to infer the context. If not, he can click the word and he is presented with a small quiz: guessing the original word, out of three possibilities. This way, the user will encounter French words and learn them eventually.

[5]https://www.amazon.com/gp/digital/fiona/kcp-landing-page/
[6]https://lingua.ly

2.4 WaitChatter

A second language learning application which has a different approach than the Zeeguu Reader for iOS is the WaitChatter extension [2]. It uses Google Chat as a basis. The idea is that when the other party in a chat is typing, you are just waiting for them to send the message. WaitChatter picks a word from the conversation and asks the user for the translation in several different ways, depending on how often the word has been presented to the user already.

Just like ALOE, WaitChatter is a passive form of learning a second language. It happens while you are doing your usual activities, while the Zeeguu Reader for iOS or services like Lingua.ly require the user to actively read something in their second language.

2.5 The Zeeguu Platform

The initial version of the Zeeguu platform was created by Simon Marti [3] as a bachelor's project at the University of Bern. Since then, the Zeeguu platform has grown and several applications were created as part of the Zeeguu platform, like the Chrome extension and several Android based applications.

2.5.1 Chrome extension

The Chrome extension allows the learner to visit any foreign webpage they like and click words they don't know. The translation appears right below the word and users can search several online dictionaries for other translations if the first one does not seem right. The reader can bookmark difficult words. His list of bookmarked words is used for the exercises, allowing the user to practice with the difficult words he bookmarked.

2.5.2 Zeeguu Quantifier

The Zeeguu Quantifier was created as part of the bachelor's project of Karan Sethi [4]. It gives the user a metric about their progress in order to quantify the user's knowledge of the language they are learning.

2.5.3 Zeeguu Translate

The Zeeguu Translate Android application is a translator that allows its users to add words to their Zeeguu word list while on the go. It also allows the user to show their current list of words and to do exercises. The app allows users to enter words by typing, pasting from the clipboard and speaking into the microphone. The words can be pronounced and bookmarked. The application was developed by Pascal Giehl [5].

2.5.4 Zeeguu Reader for Android

A Zeeguu Reader for Android was already being developed by Linus Schwab [6] as part of his bachelor's project. The Android version of the reader follows a different approach, as it is presented as an RSS reader to automatically provide the user with the newest articles from his favorite news sources, whereas the iOS version tries to hide the use of RSS from the user.

There are more differences between the Zeeguu Reader for Android and the Zeeguu Reader for iOS. The Android version only allows the reader to select a word and receive the translation, whereas The iOS version inserts the translation into the text, next to the translated word. The iOS version also allows the reader to select a word pair or a complete sentence.

Chapter 3

System Design

The Zeeguu Reader app communicates with the Zeeguu server to get translations, translation history, user profile, etcetera. This document will focus mainly on the iOS app, as the server existed before the app and is not within the scope of this project.

Within this document, the server is considered a black box that runs a REST API[1], which has a set of endpoints that the app can communicate with [7].

The iOS version of the Zeeguu Reader app consists of two major distinct parts. The user interacts with only one part: the app. Internally, the app has to communicate with the Zeeguu server, which is done via a reusable framework, which acts as a wrapper around the Zeeguu server.

This section describes the most important classes of both the Zeeguu API iOS framework and the Zeeguu Reader iOS app. Within these descriptions, other classes may be discussed. Please note that any class or protocol that starts with the prefix 'UI' or 'NS' are classes provided by iOS. Have a look at the UIKit Framework reference[2] for more information about these classes.

[1]The server is located at `https://www.zeeguu.unibe.ch/`. The server is open source, its code is located at `https://github.com/mircealungu/Zeeguu-API`.

[2]https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIKit_Framework/index.html

3.1 Zeeguu API iOS Framework

The iOS API framework[3] is a set of **`public`/`private`** classes. There is one major class that communicates with the API and a small set of helper classes that represent types of data from the server. This section describes a few of these classes. Please note that in this section, the programmer using the framework is referred to as 'the user'.

The Zeeguu API iOS framework consists of 5 public classes. Furthermore, the Zeeguu API for iOS comes with a suite set of test cases to test whether each endpoint is still functional.

3.1.1 The Communication Layer

The `ZeeguuAPI` class is the most important class of the framework. Its purpose is to abstract away the process of creating a request, waiting for its response and converting the response to something useful. It does this by offering a method for each endpoint that expects all needed data and a completion block (a block of code that can be passed around and used as if it was a function). The only thing the user has to do in order to use an endpoint is the following: The user has to gather all needed data (for example the user credentials in case of login) and call the appropiate method on the `ZeeguuAPI` instance. The completion block is executed as soon as a response is received. The completion block is called with the requested data. In case of login, the completion will receive a boolean stating if login succeeded (the REST API returns sessionID, which is hidden from the user) or in case of translating a word, it will be called with the translation as an argument.

In order to fulfill its purpose, the `ZeeguuAPI` class must remember if the user is logged in. Most of the API endpoints require a user to be logged in. In order to verify a user being logged in, the endpoints require a session ID, which is given once a user logs in. Therefore, the `ZeeguuAPI` class must remember that session ID and send it to the server along with each request. To remember the session ID, the `ZeeguuAPI` class is a singleton class, meaning that only one instance (with a session ID) can exist at any given time [8].

3.1.2 The Domain Model

To ease working with the data from the server, the Zeeguu API iOS Framework offers some classes to represent data collections from the server. The collections from the server that make most sense to give their own class representations are articles,

[3]The source code of the Zeeguu API iOS framework is available at `https://github.com/JorritO/Zeeguu-API-iOS`.

feeds and bookmarks. The `Article`, `Bookmark` and `Feed` classes represent their respective types of data collections on the server.

These classes are easier to use and understand than using the arrays and dictionaries returned by the REST API endpoints. They also offer some convenient methods which wrap around some regular `ZeeguuAPI` methods, allowing for a nicer and more readable way of working with these entities.

Article The `Article` class represents an article and contains its title, summary, url, etcetera. In addition to the data that is provided by the server, the `Article` class also provides some properties for local use, such as whether the article is read, starred, liked, how difficult the article was for the user and whether the article was completely read.

Furthermore, it contains a method for getting the contents of the article, as a convenience for the user, so he doesn't need to use the corresponding method on the `ZeeguuAPI` class. The REST API offers an endpoint that retrieves an array of articles for a given feed. The corresponding `ZeeguuAPI` method converts that array of dictionaries into an array of `Article` objects, which eases the work for the user, as he does not need to do that himself.

A similar method is offered for retrieving the difficulty of the article. The 'get contents' endpoint of the REST API already includes the difficulty of the article, but the `getDifficulty` method also offers the ability to select another difficulty computation algorithm, in which case the regular 'get difficulty' endpoint is used.

The `Article` class implements the `ZGSerializable` protocol as described in section 3.1.3.

Bookmark The `Bookmark` class represents a previously translated word or phrase and contains the word/phrase, translation, optionally context, etcetera. Even though the app is presenting the bookmarks as history, the server still calls it bookmarks. The endpoints that retrieve bookmarks actually return arrays of `Bookmark` objects.

The `Bookmark` class offers some convenient methods for deleting the bookmark, adding or removing translations and retrieving all translations for the bookmark. These methods are just wrappers around the corresponding endpoints in the `ZeeguuAPI` class, giving them the id of the `Bookmark` instance.

The `Bookmark` class also implements the `ZGSerializable` protocol, although this functionality is not used by the Zeeguu Reader yet.

Feed The `Feed` class represents an RSS feed. It contains the feeds title, url, language, imageURL, etcetera. It also contains a method that retrieves its image, using the imageURL that is present.

As the `Article` class includes an instance of `Feed` among its properties, the `Feed` class is required to implement the `ZGSerializable` protocol.

ArticleDifficulty The `String`-based `ArticleDifficulty` enum represents how easy it was for a reader to understand an article. The enum offers three choices: `Easy`, `Medium` and `Hard`. Additionally it can have the value `Unknown`, for situations where the difficulty is not (yet) known. The enum offers a `description` method which returns a localized version of its raw value, which is a `String` object, suitable for use within a user interface. It also offers a `color` method that returns a color based on which value the enum has. This color can be used in a graphical user interface for easy recognition of the article difficulty.

3.1.3 Serialization

The Zeeguu Reader for iOS saves all articles locally in order to remember if the reader read the article, liked it, etcetera. Saving the articles also allows for keeping a history of articles, as the server only delivers the last n articles. To allow the Zeeguu Reader to save articles, the `ZGSerializable` protocol and `ZGSerialize` helper class were created. Objects conforming to `ZGSerialize` can be converted to a dictionary and back. Dictionaries can easily be stored in plist[4] based files, such as the default preferences file that is used by `NSUserDefault`.

ZGSerializable The `ZGSerializable` protocol dictates an initializer and a method for converting an object to a dictionary and back. This dictionary is of type `[String: AnyObject]`. This protocol allows for easy conversion to and from a dictionary, which makes it easy (and fairly trivial) to store an object in `NSUserDefaults`. The Zeeguu Reader iOS app is using the availability of these methods to store lists of `Article`s in `NSUserDefaults`.

Listing 3.1: The methods required by `ZGSerializable`

```
    init?(dictionary dict: [String: AnyObject])
    func dictionaryRepresentation() -> [String: AnyObject
        ]
```

The initializer should be of the form shown in listing 3.1. This initializer should initialize an instance of the class implementing `ZGSerializable`, using the keys

[4]A ‘plist’ file is a ‘property list’ file that stores information in XML. For more information see also https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/PropertyLists/Introduction/Introduction.html

and values in `dict`. Note that this initializer is a ‘failable initializer’, meaning that the initializer may throw an exception or return nil.

The `dictionaryRepresentation` method must return a dictionary that contains all information about the object that can be used to reconstruct the object again using the initializer.

The `Article`, `Bookmark` and `Feed` classes all implement these methods by returning a dictionary with all properties stored with their keys being the names of the corresponding properties.

ZGSerialize The `ZGSerialize` class offers some static methods for ‘encoding’ and ‘decoding’ `ZGSerializable` objects. It offers methods for both single `ZGSerializable` objects and arrays of `ZGSerializable` objects.

3.2 The User Interface

The Zeeguu Reader app[5] consists of three main sections. The feeds/articles, word translation history and profile, as shown in Figure 3.1:

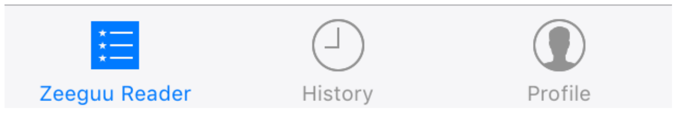


Figure 3.1: The tabbar showing the three sections of the app

The feeds/article section contains the feeds and tapping a feed leads to a list of its articles, which in turn gives access to the article's contents.

The history section contains a list, grouped by date, of previously translated words/phrases. By tapping a translation, also the original context of the translated word is shown.

The profile shows the user's name and email and allows the user to change their base language and the language they want to learn.

Note: Some of the about 29 classnames are shortened for readability. `TVC` stands for `TableViewController` and `VC` stands for `ViewController`.

AppDelegate The `AppDelegate` class is the main class of the Zeeguu Reader. This class is responsible for handling all kinds of events from the system. For example, if the application finished loading, the appropriate method of `AppDelegate` is called, which sets up the initial views and presents a login screen if necessary. This 'appropriate' method is, among others, prescribed by the `UIApplicationProtocol`, to which `AppDelegate` conforms.

Welcome screen The Zeeguu Reader for iOS features a very basic welcome screen, that uses a `UIScrollView` to display a few `UIImageView`s to display a few screenshots with added text to explain some of the features of the app. After the user scrolls past all images, he can dismiss the welcome screen. The scroll view has 'paging' enabled, meaning that if the user scrolls halfway an image and releases his finger, the scroll view bounces to the image that is more than 50% visible at that moment. To show the 'scroll progress', the welcome screen features a `UIPageControl` that shows as many circles as there are images and shows at which image the user is.

[5]The source code of the Zeeguu Reader iOS app is available at `https://github.com/JorritO/Zeeguu-Reader-iOS`.

3.2.1 Feeds

The Zeeguu Reader for iOS uses RSS feeds, managed by the Zeeguu REST API, to deliver articles to its users. In contrast to the Android version of the Zeeguu Reader (section 2.5.4), the iOS version tries to hide that fact from the user, by allowing the user to enter normal site urls and having the Zeeguu Rest API find the corresponding RSS feeds on the website.

FeedOverviewTVC The `FeedOverviewTVC` class is responsible for showing all feeds the logged in user subscribed to. This class is a subclass of `UITable-ViewController`, which displays a `UITableView` on the screen. The `Feed-OverviewTVC` acts as the delegate[6] and datasource of that table view and feeds the tableview an entry for each feed. Each feed entry is displayed using an instance of the `FeedTableViewCell` class.

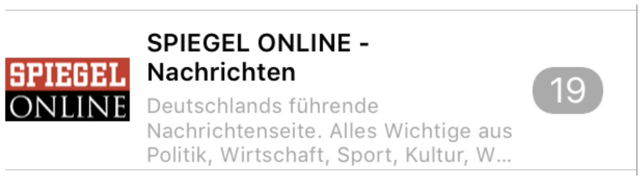

SPIEGEL ONLINE
SPIEGEL ONLINE - Nachrichten
Deutschlands führende Nachrichtenseite. Alles Wichtige aus Politik, Wirtschaft, Sport, Kultur, W...
19

Figure 3.2: A feed cell as displayed within the list of feeds

The `FeedTableViewCell` (Figure 3.2) displays the title and description of the feed. Additionally, as most feeds feature a 'feed image', usually the logo of the corresponding website, the cell also displays that image. To show the user how many articles of the feed are unread, the articles are retrieved and each feed cell shows how many articles the user hasn't read yet.

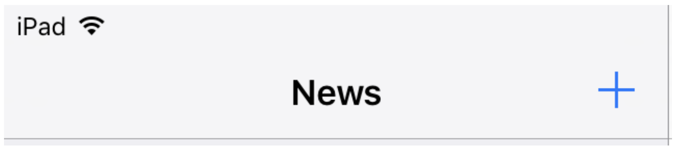


Figure 3.3: The navigation bar featuring a title and an 'Add' button

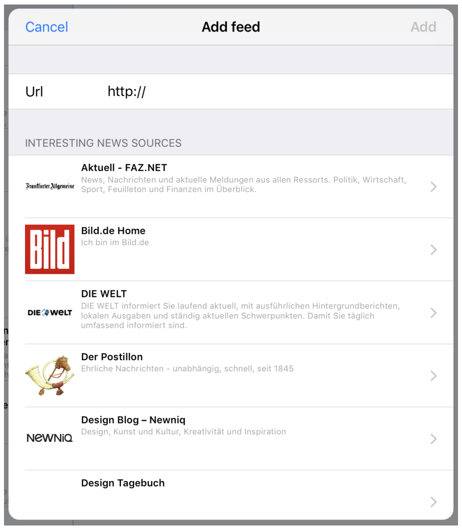
The `FeedOverviewTVC` is displayed within a `UINavigationController` , to enable easy navigation between view controllers[7] and the navigation bar, which

[6]A delegate is an object that can be used to delegate operation to. For example, a user can tap a row in a table view. The table view is a generic object, representing a table view element on screen and doesn't know what to do when a row is tapped, apart from selecting it. That is were the delegate comes in. The table view informs the delegate about which row was tapped and delegates the decision of what has to be done to the delegate object. The delegate conforms to a protocol (in this case `UITableViewDelegate`), that prescribed methods that can be delegated.

[7]The `UI-Navigation-Controller` contains a view controller stack. The navigation be-

is shown on top of the tableview (Figure 3.3). The `FeedOverviewTVC` is also responsible for populating the navigation bar with a title and buttons. The `FeedOverviewTVC` class adds an add button to the navigation bar that allows the user to add one or more feeds.

AddFeedTVC, SelectFeedsTVC The `AddFeedTVC` (Figure 3.4a) and `SelectFeedsTVC` (Figure 3.4b) allow the user to subscribe to extra news feeds. The `AddFeedTVC` allows the user to enter a url of their favorite website. After adding the url, the `SelectFeedsTVC` is presented, which will display all RSS feeds that were found on the entered website[8]. The `SelectFeedsTVC` allows the user to select one or more to add to their feed list.



Url http://

INTERESTING NEWS SOURCES

Aktuell - FAZ.NET
News, Nachrichten und aktuelle Meldungen aus allen Ressorts. Politik, Wirtschaft, Sport, Feuilleton und Finanzen im Überblick.

Bild.de Home
Ich bin im Bild.de

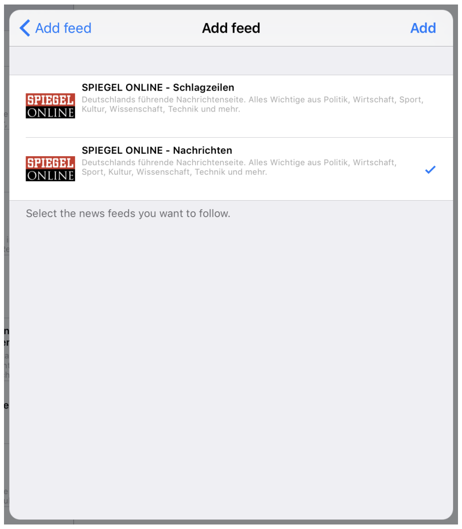
DIE WELT
DIE WELT informiert Sie laufend aktuell, mit ausführlichen Hintergrundberichten, lokalen Ausgaben und ständig aktuellen Schwerpunkten. Damit Sie täglich umfassend informiert sind.

Der Postillon
Ehrliche Nachrichten - unabhängig, schnell, seit 1845

Design Blog – Newniq
Design, Kunst und Kultur, Kreativität und Inspiration

Design Tagebuch

(a) The 'Add feed' screen



SPIEGEL ONLINE - Schlagzeilen
Deutschlands führende Nachrichtenseite. Alles Wichtige aus Politik, Wirtschaft, Sport, Kultur, Wissenschaft, Technik und mehr.

SPIEGEL ONLINE - Nachrichten ✓
Deutschlands führende Nachrichtenseite. Alles Wichtige aus Politik, Wirtschaft, Sport, Kultur, Wissenschaft, Technik und mehr.

Select the news feeds you want to follow.

(b) 'Select feed' screen for http://spiegel.de

Figure 3.4: The 'Add feed' and 'Select feed' screens

The `AddFeedTVC` features an 'interesting news sources' section. This section, generated by the Zeeguu server using its endpoint, only shows a list of news feeds written in the language the user is learning.

Both the `AddFeedTVC` and `SelectFeeddTVC` use the `FeedTableViewCell` to show the interesting news sources and available feeds respectively.

tween view controllers, which can be seen as screens of the app, is like the navigation between screens within the settings app on iPhone. When a new view, managed by its view controller, is pushed, it slides in from the right. Similarly, if the top view is popped, slides off screen to the right, revealing the previous view.

[8]Some website mention multiple feeds in the `head` section of the page. For example, they offer a general feed, with all articles and offer some specialized feeds with only sports articles, foreign affairs, etcetera.

3.2.2 Articles

ArticleListVC The `ArticleListVC` class is responsible for showing the articles that belong to a given feed (or in one case for all subscribed feeds). This controller also displays a table view that displays all articles by using instances of the `Article TableViewCell` class for displaying the article titles and descriptions nicely.

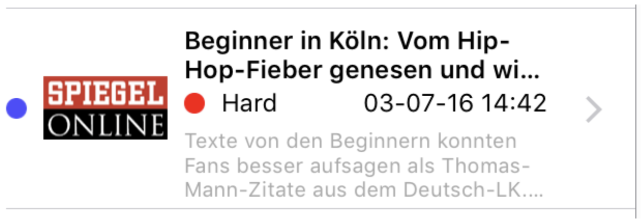

Beginner in Köln: Vom Hip-Hop-Fieber genesen und wi...
Hard 03-07-16 14:42
Texte von den Beginnern konnten Fans besser aufsagen als Thomas-Mann-Zitate aus dem Deutsch-LK....

Figure 3.5: An article cell as displayed within the list of articles

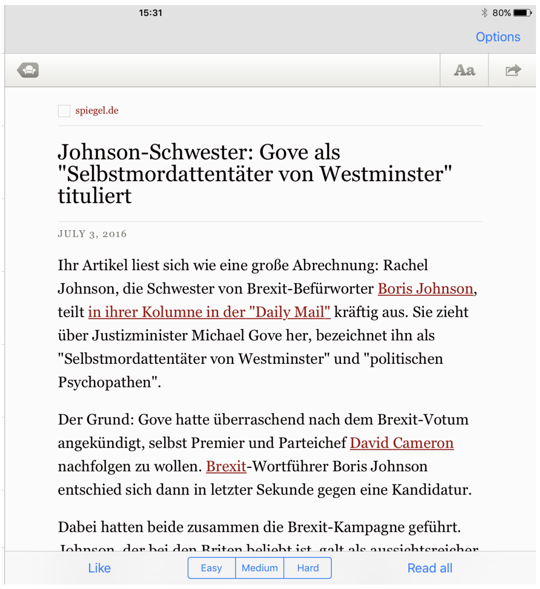
Each cell (Figure 3.5) shows the title, description and feed image. Additionally, the difficulty is shown, using both color (green, orange or red) and text (easy, medium or hard). Of course, the cell does not lack the date and time the article was published. To indicate whether an article is unread, a blue circle is shown at the left side of the cell and it disappears when the reader opens the article.

ArticleVC The `ArticleVC` is responsible for showing the article's contents. In a first version, it used an instance of the `ArticleView` class to display the article's title and contents. However, due to the Zeeguu REST API relying on the beautiful-soup[9] library which was not correctly extracting the contents of some articles, the `ArticleView` was replaced by a web view. Take for example the german news website 'Der Spiegel': the contents extracted by the server did not start with the small description given by article entry within the feed. In this case, the intro of the article was not positioned in the same element as the rest of the article, therefore, the intro was missing in the extracted content and therefore the extracted content did not correspond to the small description given by the feed.

To solve this problem, the second iteration of the application uses a web view to display the articles. To achieve a consistent look, the url to the article online is given to Readability[10] and the resulting readability page is displayed in the web view (Figure 3.6a).

While reading the article, there are a few things to customize (Figure 3.6b). It if possible to change the font size of the article, to improve readability of the text. The reader can disable links, allowing the reader to translate words inside hyperlinks, without the article disappearing, because the web view followed the link.

[9] `https://www.crummy.com/software/BeautifulSoup/`
[10] `https://www.readability.com`



spiegel.de

Johnson-Schwester: Gove als "Selbstmordattentäter von Westminster" tituliert

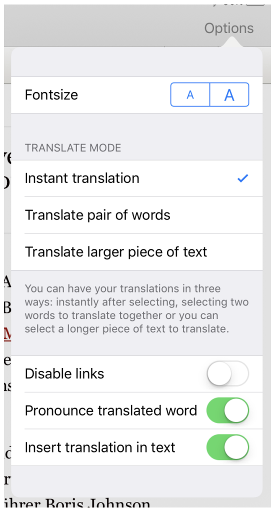
JULY 3, 2016

Ihr Artikel liest sich wie eine große Abrechnung: Rachel Johnson, die Schwester von Brexit-Befürworter Boris Johnson, teilt in ihrer Kolumne in der "Daily Mail" kräftig aus. Sie zieht über Justizminister Michael Gove her, bezeichnet ihn als "Selbstmordattentäter von Westminster" und "politischen Psychopathen".

Der Grund: Gove hatte überraschend nach dem Brexit-Votum angekündigt, selbst Premier und Parteichef David Cameron nachfolgen zu wollen. Brexit-Wortführer Boris Johnson entschied sich dann in letzter Sekunde gegen eine Kandidatur.

Dabei hatten beide zusammen die Brexit-Kampagne geführt. Johnson, der bei den Briten beliebt ist, galt als aussichtsreicher

(a) The article screen



Fontsize

TRANSLATE MODE

Instant translation ✓

Translate pair of words

Translate larger piece of text

You can have your translations in three ways: instantly after selecting, selecting two words to translate together or you can select a longer piece of text to translate.

Disable links: off

Pronounce translated word: on

Insert translation in text: on

(b) Article options

Figure 3.6: Article screen with options

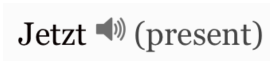
By default, if a word is tapped, it is pronounced. While the pronunciation is played, the translation is retrieved from the server. The reader can choose to disable the pronunciation, but can have it pronounced later on by tapping a speaker icon (Figure 3.7a) next to the translation.

After the translation is retrieved, the translation is inserted into the text. If the reader doesn't like that, the reader can disable this behavior from the options menu. When a translation is retrieved, the translation is briefly displayed at the bottom of the article view instead.

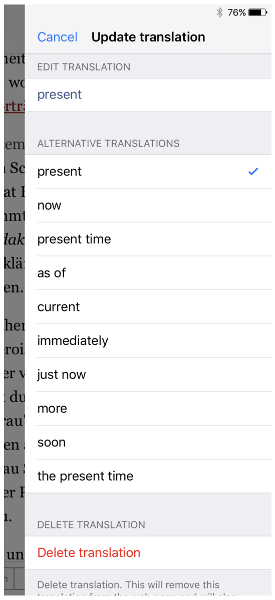
If the given translation seems incorrect, tapping the translation (Figure 3.7a) shows a view where the reader can select an alternative translation, give their own translation or delete the existing one (Figure 3.7b). Deleting the translation removes it both from the text and the history. If another translation is selected or given, using the 'Edit translation' text field, the translation is updated both within the text and the history.

The first iteration of the application offered two methods of translation. Selection of a single word or a whole sentence/large piece of text. With the new approach, using the web view, three methods are offered (Figure 3.6b).

1. Instant translation. With this method active, if the reader taps a word, the app immediately translates the word.


Jetzt (present)

(a) Translating 'Jetzt'


Cancel Update translation

EDIT TRANSLATION

present

ALTERNATIVE TRANSLATIONS

present ✓
now
present time
as of
current
immediately
just now
more
soon
the present time

DELETE TRANSLATION

Delete translation

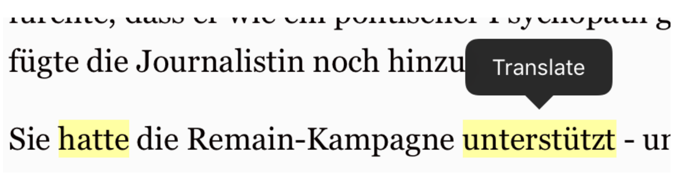
Delete translation. This will remove this

(b) Update translation screen

Figure 3.7: Updating translation of 'Jetzt'

2. Translating a word pair (Figure 3.8). With this method, the reader has to select two words. After having selected the second word, a small pop-up appears with a 'Translate' button. Tapping the translate button fuses the two words together, separated by a space, in the order they appeared in the text and translates the combination.

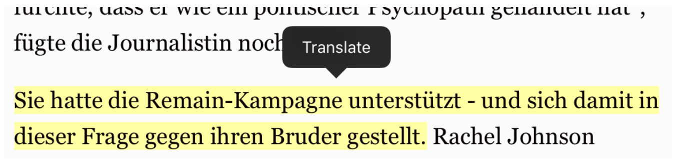
3. Translating a sentence (Figure 3.9). Using this method, the reader has to select two words again, but this time all text in between these two words is also translated.


fügte die Journalistin noch hinzu

Sie hatte die Remain-Kampagne unterstützt - ur

Figure 3.8: Selecting 'hatte unterstützt'

The article view also features a toolbar at the bottom of the screen (Figure 3.6a). This contains a like/dislike button, a 'segmented' control for selecting the difficulty and a 'read all' button. The interactions with these buttons are only stored locally, so liked articles and difficulty assessments are not synchronized between multiple



fügte die Journalistin noch

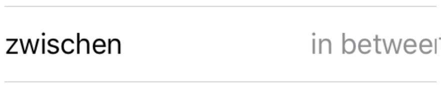
Sie hatte die Remain-Kampagne unterstützt - und sich damit in dieser Frage gegen ihren Bruder gestellt. Rachel Johnson

Figure 3.9: Selecting 'Sie hatte die ... ihren Bruder gestellt.'

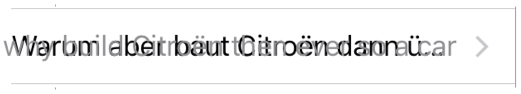
devices. Therefore, if the reader likes an article and assesses the article as hard, that information is stored locally and persists after having restarted the app, updating the buttons accordingly.

3.2.3 History

HistoryTVC The `HistoryTVC` is responsible for displaying the list of previously translated words and phrases. The controller receives all `Bookmark`[11] objects from the `ZeeguuAPI`, grouped by date, and displays them using regular `UITable-ViewCell` objects. The `UITableViewCell` offers a few layouts for displaying information within a cell. First, this controller used the style that is displayed in Figure 3.10. For just a single word (Figure 3.10b), this style is fine, however, the reader also supports translating sentences or even complete paragraphs and in those cases, this style fails in such a way that both the word and the translation get mixed up in each other (Figure 3.10b).

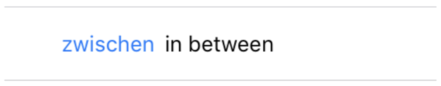


(a) Single word

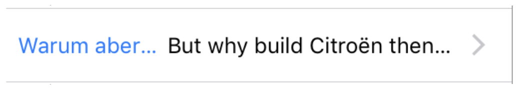

Warum aber baut Citroën dann ü...
Why build Citroën then a car

(b) Sentence

Figure 3.10: First History table view cell style



(a) Single word

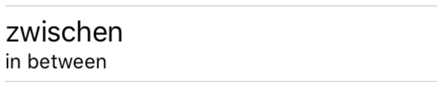


(b) Sentence

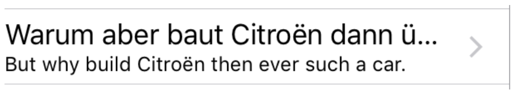
Figure 3.11: Alternative History table view cell style

To solve this problem, it was possible to use one of the other available styles (figures 3.11 and 3.12). After comparing Figure 3.11b with Figure 3.12b, it turns out that the style from Figure 3.12 performs best with longer pieces of text. As a

[11]Though these aren't really bookmarks, the `ZeeguuAPI` still refers to them as 'bookmark', because the endpoints on the server still do.


zwischen
in between

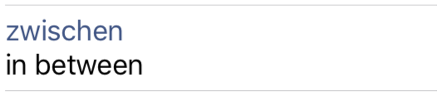
(a) Single word


Warum aber baut Citroën dann ü...
But why build Citroën then ever such a car.

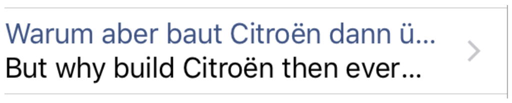
(b) Sentence

Figure 3.12: Second alternative History table view cell style

result, the style from Figure 3.12 was chosen, but with a slightly customized layout (Figure 3.13).


zwischen
in between

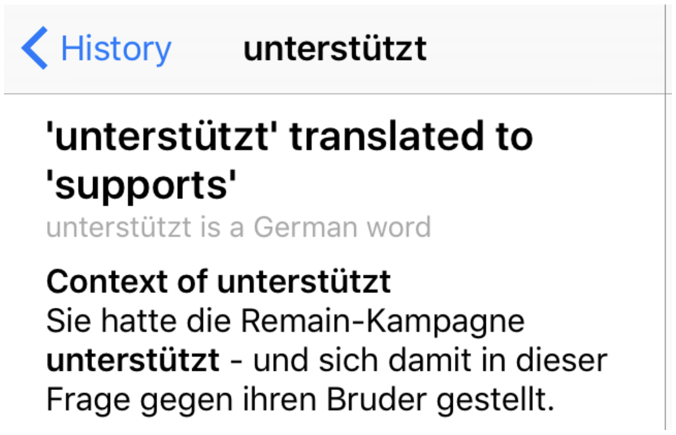
(a) Single word


Warum aber baut Citroën dann ü...
But why build Citroën then ever...

(b) Sentence

Figure 3.13: Final History table view cell style

HistoryItemVC The `HistoryItemVC` class is responsible for displaying the original word, the translation and the context in which the word was translated. This controller is presented after a user tapped an entry in the `HistoryTVC`. The `HistoryItemVC` just displays an instance of the `HistoryItemView` class that is responsible for displaying the data correctly.



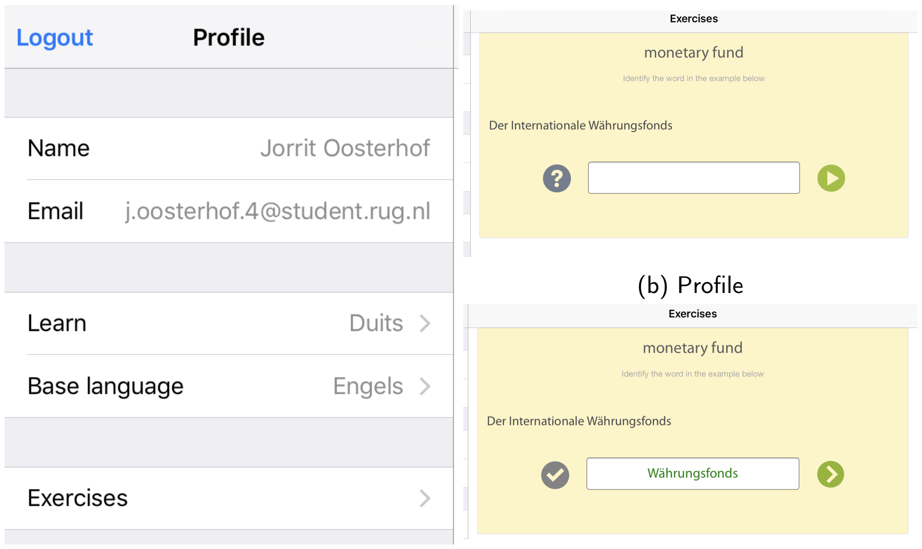
'unterstützt' translated to 'supports'
unterstützt is a German word
Context of unterstützt
Sie hatte die Remain-Kampagne **unterstützt** - und sich damit in dieser Frage gegen ihren Bruder gestellt.

Figure 3.14: An example of a history item

The `HistoryItemView` (Figure 3.14) shows the original word and its translation. It also shows how the word appeared in its context and from which language the word is.

3.2.4 Profile

ProfileTVC The `ProfileTVC` (Figure 3.15a) is responsible for displaying the user information. It shows the name, email, learn language and base language of the user similar to how the `HistoryTVC` shows words and translations. It also allows the user to change the learned and base language: it presents an instance of the `LanguagesTVC` class, which displays all available languages for a given mode (learned language or base language). Once the user chose a new language, the `ProfileTVC` is notified, as it listed itself as a delegate of the `LanguagesTVC` instance. Once the `ProfileTVC` is notified, it updates the language preference using the `ZeeguuAPI` class and updates its view to reflect the changes.



Logout Profile

Name Jorrit Oosterhof

Email j.oosterhof.4@student.rug.nl

Learn Duits

Base language Engels

Exercises

Exercises

monetary fund

Identify the word in the example below

Der Internationale Währungsfonds

Exercises

monetary fund

Identify the word in the example below

Der Internationale Währungsfonds

Währungsfonds

(a) Profile (b) Profile (c) Exercise correctly answered

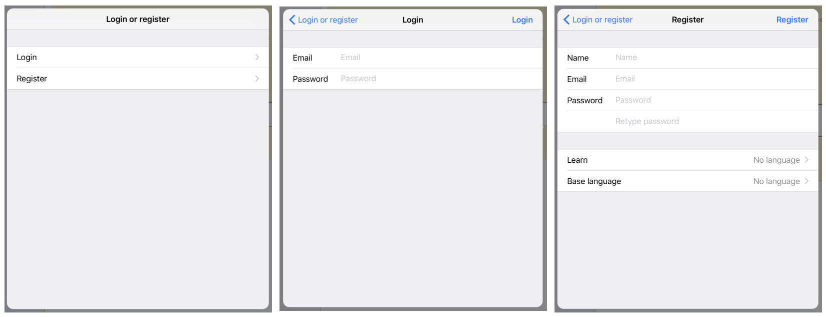
Figure 3.15: Profile and exercises

Additionally, the `ProfileTVC` offers a button to open the exercises (Figure 3.15b). For example, an exercise gives the translation of a word. It is then the job of the user to identify the original word in the given context. The translation in Figure 3.15b is 'monetary fund'. The user now has to identify which German word resulted in this translation. In this example, the user should identify 'Währungsfonds' as the original word (Figure 3.15c).

Login-, Register- and LoginRegisterTVC The `LoginTVC` (Figure 3.16b) allows the user to login, the user enters their email and password and the `LoginTVC` logs the user in or displays a failure message.

The `RegisterTVC` (Figure 3.16c) allows the user to create an account. The user has to enter their name, email and choose a password. The user also has to choose the language they want to learn and their base language, which are also chosen using the `LanguagesTVC`.

The `LoginRegisterTVC` (Figure 3.16a) simply offers the choice to login or to register.



(a) Login or register (b) Login (c) Register

Figure 3.16: The login/register screens

3.3 Web View integration

This section describes the challenges and difficulties in displaying the article's contents and interacting with the words to translate them.

3.3.1 First Iteration

In the first iteration of the application, the `ArticleVC` was responsible for showing the article's contents. It used an instance of the `ArticleView` class to display the article's title and contents. The `ArticleView` used a subclass of `UITextView`, called `ZGTextView`, that intercepted text selection events and asked the `ZeeguuAPI` for a translation. The `ZGTextView` also inserted the received translations in the article text, with a grey font color.

The `ArticleVC` was also responsible for toggling the translate mode that the `ArticleView` used. There were two distinct translation modes. The first mode intercepted the text selection and immediately began to retrieve a translation. The second mode allowed the user to select more than one word, for example a complete sentence, and showed a small popup with a 'Translate' button, like the system default cut/copy/paste selection popup. The 'Translate' popup was also a responsibility of the `ArticleVC`. The `ArticleVC` added the translate button to the general `UIMenuController` instance that actually implements the text selection popup.

Because the automatically extracted article contents delivered by beautifulsoup were not satisfactory, the decision was made to use a web view instead.

3.3.2 Second Iteration

Using a web view introduces several challenges, because the approach that worked for the text view in the first iteration did not work for the web view.

Intercepting selection events For example, detecting selection and intercepting selection events like before was not possible, as there was no way to achieve that without some 'hacks'.

The next approach was to detect a tap on the web view and then find out which word was tapped. Also this approach was not viable, because it is impossible to get from coordinates within the web view to an element from the HTML page, let alone find out which word was tapped.

The final approach is to loop over all text elements of the HTML page. Each text element is enclosed within `<zeeguuParagraph></zeeguuParagraph>` tags. To detect which word is tapped, simply all words within the text elements are enclosed within `<zeeguuWord></zeeguuWord>` tags. After that, simply attaching a click handler to each `zeeguuWord` tag is enough to detect taps on words.

Extracting context However, knowing the tapped word is not enough. To have exercises, the context of the word is also important. To find the context, the period (`.`), exclamation mark (`!`) and question mark (`?`) are enclosed in `<zeeguuPeriod ></zeeguuPeriod>` tags. Now, to find the context, the algorithm loops through the neighboring elements until it finds the `zeeguuPeriod` element. This procedure is executed twice, one time to find the beginning of the context and one time to find the end of the context. If there is no `zeeguuPeriod` element is found, it is considered done.

Take for example the word *Finanzausschusses* (line 13) in listing E.1 (section E.1). To find its context, the algorithm will walk over all `zeeguuWord` elements (towards the first one), until it is at the beginning of the `zeeguuParagraph` element at the word *Schneider* (line 3). Then it will walk over all `zeeguuWord` elements (towards the last one) until it finds a `zeeguuPeriod` element (line 34). The resulting context is then: *Schneider wies darauf hin, dass es in der Anhörung des*

Finanzausschusses *zahlreiche Bedenken gegeben habe, ob das Ziel der Förderung insbesondere des sozialen Wohnungsbaus mit dem vorgelegten Gesetzentwurf erreicht werden könne.*

With this approach, there was still one challenge to solve. Like said before, each text element was enclosed within `<zeeguuParagraph></zeeguuParagraph>` tags. However, if there is some link (listing 3.2) in between the tapped word and one of the context ends, the algorithm would stop at the link. So, for example, the context of "*word n-1*" (listing 3.3, line 22) would become "*word y+1 word y+2 ...* ***word n-1*** *word n*." Therefore, if the algorithm cannot go any further, it will check if the neighbour of the current `zeeguuParagraph` is a link (or bold text using `<b></b>`, etcetera), it will continue into the link and will also leave the link again into the next neighboring `zeeguuParagraph`, until it finds the correct begin or end of the context.

Using this modification, the context of "*word n-1*" becomes "*word 1 word 2 ... word x-1 word x word x+1 word x+2 ... word y-1 word y word y+1 word y+2 ...* ***word n-1*** *word n*."

Listing F.1 in appendix F shows the complete 'get context' algorithm.

Listing 3.2: Text elements with a link

```
<parent>
    #text  <!-- A text element with an arbitrary
         amount of text -->
    <a href="http://www.some.web/page.html">
        #text
    </a>
    #text
</parent>
```

Listing 3.3: Text elements with a link, after being processed

```
1  <parent>
2      <zeeguuparagraph>
3          <zeeguuword>word 1</zeeguuword>
4          <zeeguuword>word 2</zeeguuword>
5          <zeeguuword>...</zeeguuword>
6          <zeeguuword>word x-1</zeeguuword>
7          <zeeguuword>word x</zeeguuword>
8      </zeeguuparagraph>
9      <a href="http://www.some.web/page.html">
10         <zeeguuparagraph>
11             <zeeguuword>word x+1</zeeguuword>
12             <zeeguuword>word x+2</zeeguuword>
13             <zeeguuword>...</zeeguuword>
14             <zeeguuword>word y-1</zeeguuword>
```

```
            <zeeguuword>word y</zeeguuword>
        </zeeguuparagraph>
    </a>
    <zeeguuparagraph>
        <zeeguuword>word y+1</zeeguuword>
        <zeeguuword>word y+2</zeeguuword>
        <zeeguuword>...</zeeguuword>
        <zeeguuword>word n-1</zeeguuword>
        <zeeguuword>word n</zeeguuword>
        <zeeguuperiod id="zeeguuPeriod1">.</zeeguuperiod>
    </zeeguuparagraph>
</parent>
```

3.3.3 Interaction between website and app

There are two interactions possible between a webpage and the app:

- The app executes code within the webpage.
- The webpage executes code within the app

To enable the app executing code within the webpage, the `WKWebView` class offers a method that allows the programmer to execute JavaScript code within the webpage. It even allows for injection of entire JavaScript files at each page load. Consequently, the Zeeguu Reader has a few JavaScript files with all functionality concerning the article text, which are being injected at each page load. Later on, if something needs to be executed, the Zeeguu Reader executes a little bit of JavaScript, usually a function call or setting the value of some global property.

To make this a bit easier, the `ZGJavaScriptAction` enum was introduced. Each **`case`** may contain some arguments, in some cases a dictionary with all information. The enum offers a method that returns a string with the corresponding JavaScript code. Some actions hold a dictionary and need the possibility to add a translation to the dictionary or update one. There are methods available to handle this, which do nothing if the enum has another value. Have a look at listing G.1 in appendix G for the `ZGJavaScriptAction` enum.

To allow the webpage to execute code within the app, the `WKWebView` exposes a `webkit.messageHandlers.zeeguu.postMessage` function, where the `zeeguu` part is custom and defined by the app. By sending a dictionary with the action information and any other necessary information, the app is able to determine what it is supposed to do and executes the corresponding procedure. Listing 3.4 and 3.5 show how the webpage can have the app execute a specific action.

Listing 3.4: An example of how to send an action to the app from JavaScript

```
function zeeguuPostMessage(message) {
    window.webkit.messageHandlers.zeeguu.postMessage(
        message);
}

var word = "Wörterbuch";
var message = {action: "pronounce", word: word};
zeeguuPostMessage(message);
```

Listing 3.5: An example of how to receive an action from JavaScript

```
func userContentController(userContentController:
    WKUserContentController, didReceiveScriptMessage
    message: WKScriptMessage) {
    guard let body = message.body as? Dictionary<String,
        AnyObject> else {
        return
    }
    var dict = Dictionary<String, String>()

    let action = ZGJavaScriptAction.parseMessage(dict)

    switch action {
    case .Pronounce(_):
        self.pronounceWord(action)
        break
    default:
        break
    }
}
```

Chapter 4

User Study

After doing multiple iterations and testing the application within the team, we decided that it was time to validate it with external users.

We organized a user study to validate whether our assumptions about the usability and usefulness of the application were correct.

We conducted the user study in a similar way to how several other ways of learning a second language were evaluated. [1, 9]

To conduct a user study, we first needed participants. We invited participants to our study, by asking a professor the language center of the University of Groningen to ask his students to participate. We also posted an announcement on Facebook.

Each participant was required to fill in a pre-study questionnaire before they could use the Zeeguu Reader iOS app. The questionnaire gathers some information about the participant, like name, email, age and further background information. After the participants filled in the questionnaire, they received an email from TestFlight[1] with instructions on how to install the app.

The pre-study questionnaire was filled in by 24 potential participants. The pre-study questionnaire asked whether the future participant was interested in reading texts on their iPhone or iPad, Android device or the Google Chrome browser, because that is interesting to know for the Zeeguu Team (appendix B). This study only picked the participants who were interested in reading texts on their iPhone or iPad, as this study is about the Zeeguu Reader for iOS app. According to this criterion, 16 participants remained.

[1]Testflight is a service offered by Apple that lets developers test their applications without needing to have the app in the App Store or fiddling with unique device IDs. Visit `https://developer.apple.com/testflight/` for more information.

Of these 16 participants, 13 participants actually installed the app, and 7 used it on a daily basis for about 5 to 15 minutes. The app sends certain events to the server while the app is used. Table 4.1 shows the events that are sent to the server. The 'App use' event is measured in on-screen time, during which the app is not necessarily used. These events are used to determine which features are used by the users and also allows for determining which features are popular or never used.

Event	**Arguments**
userUsedAppInSeconds	Time in seconds
userOpensArticle	-
userTranslatedUsingInstantTranslation	Translation response, JavaScript action information
userOpensExercises	-
userPronouncesWord	-
userSwitchesToInstantTranslation	-
userSwitchesToWordPairTranslation	-
userSwitchesToSentenceTranslation	-
userEnablesLinks	-
userDisablesLinks	-
userLikesArticle	1 if like, 0 if dislike, article url
userSaysArticleDifficultyEasy	Article url
userSaysArticleDifficultyMedium	Article url
userSaysArticleDifficultyHard	Article url
userReadArticleCompletely	Article url
userTranslatedUsingWordPairTranslation	Translation response, JavaScript action information
userTranslatedUsingSentenceTranslation	Translation response, JavaScript action information
userEditedTranslation	JavaScript action information
userOpensHistoryItem	-
userDeletedTranslation	JavaScript action information

Table 4.1: The events that are sent to the server

To conclude the user study, the participants were asked to fill in a final questionnaire, which gathers information about how the users experienced the app. The post-study questionnaire was filled in by 6 participants, who were generally positive about the app (appendix D).

The users The 7 users who were using the app for multiple days, are mostly male and between 35 and 44 years old. Most of them have graduated from a university and two of them are doing or have finished their PhD. 2 out of the 7 users were still Bachelor students. 3 participants were interested in learning German and 3 others were interested in learning French. The last one wanted to learn Dutch.

One participant had a language level of A1, three participants had a language level of B1, two participants had a language level of B2 and one participant had a language level of C1. Therefore, we can assume that most users can at least understand the main points of clear speech and can narrate an event, experience or dream.

4.1 Usage results

As mentioned above, 7 used the app actively for multiple days. Some users used the app more than others, but they all did some actual reading. Table 4.2 shows the language level of each user.

User	Language level
1	A1
2	B1
3	B1
4	B2
5	B2
6	B1
7	C1

Table 4.2: The language levels of each user

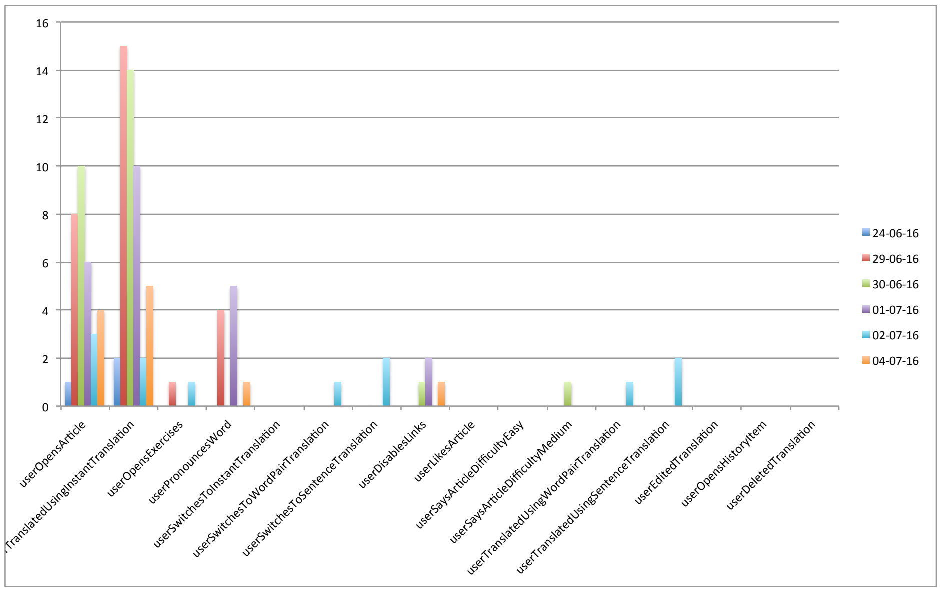


Figure 4.1: Usage data of user #1

Figure 4.1 shows the usage data of user #1. This user read many articles and also translated many words. He has turned off the pronounce words option, as visible by the fact that less words have been pronounced than words were translated. This user has also switched to word pair and sentence translation and used these translation modes to do some translations.

Figure 4.2 shows the usage data of user #2, who also translated many some

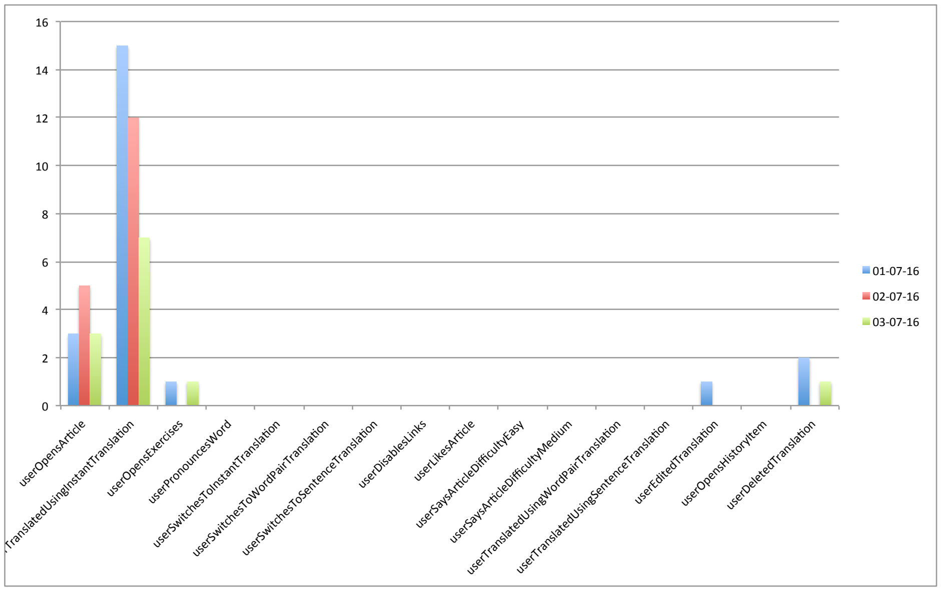


Figure 4.2: Usage data of user #2

words and read less articles, however, this user only used the instant translation option. Furthermore, user #2 is one of two users who discovered the existence of the feature to edit and delete translations. User #2 had the pronounce word option disabled from the beginning, because his graph doesn't show any word having been pronounced.

Figure 4.3 shows the usage data of user #3, who read few articles, but translated more words than user #1 as well as user #2. This user left the option to automatically pronounce words on, as Figure 4.3 shows exactly as many pronunciations as there are translations.

Figure 4.4 shows the usage data of user #4. He read many articles and didn't need to translate many words. This user probably turned of the automatic pronunciation feature after hearing the first pronounced word. He also switched to both other translation modes, but has not used these translation modes to do any translation.

Figure 4.5 shows the usage data of user #5. This user also read many articles, but didn't translate too many words. This user has not used any other translation mode, besides the default one and also turned off the pronunciation feature quite quickly. This user has a B2 language level, so it's likely he understood what the articles were saying and could infer most unknown words.

Figure 4.6 shows the usage data of user #6. This user has read few articles and

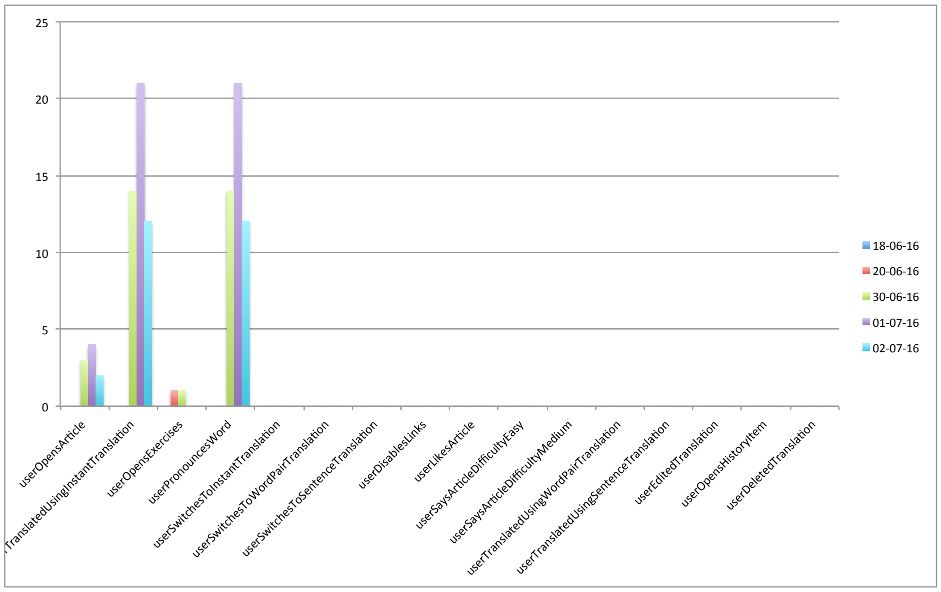


Figure 4.3: Usage data of user #3

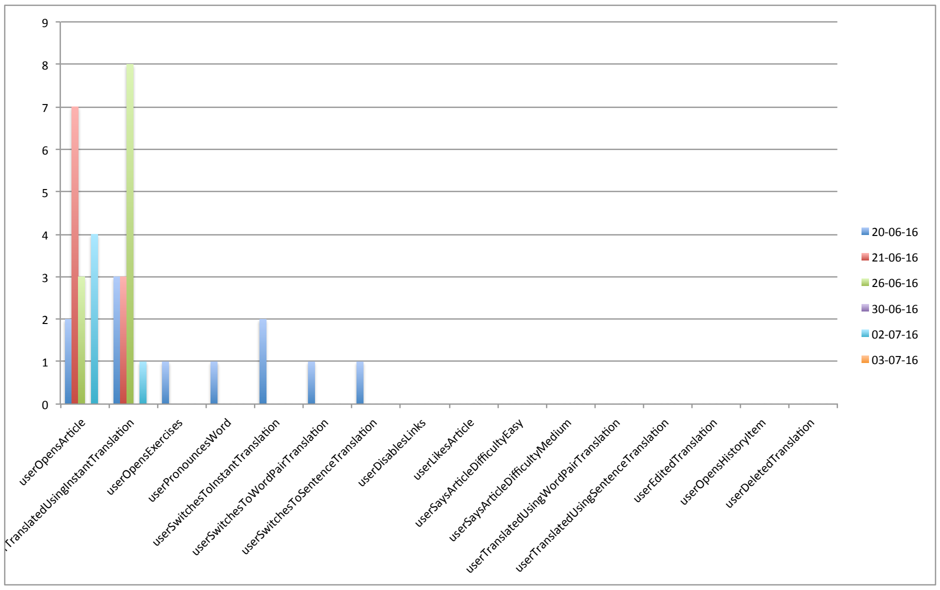


Figure 4.4: Usage data of user #4

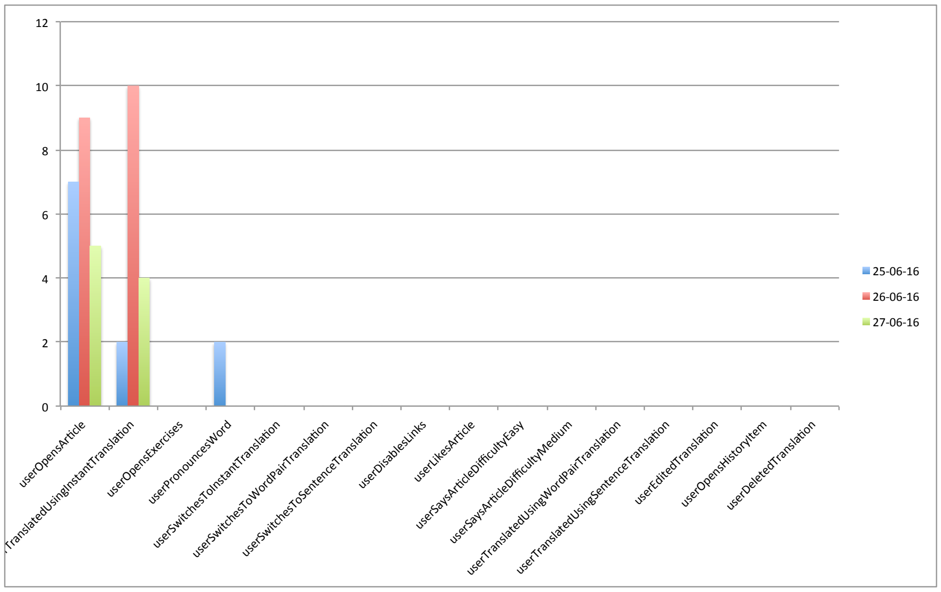


Figure 4.5: Usage data of user #5

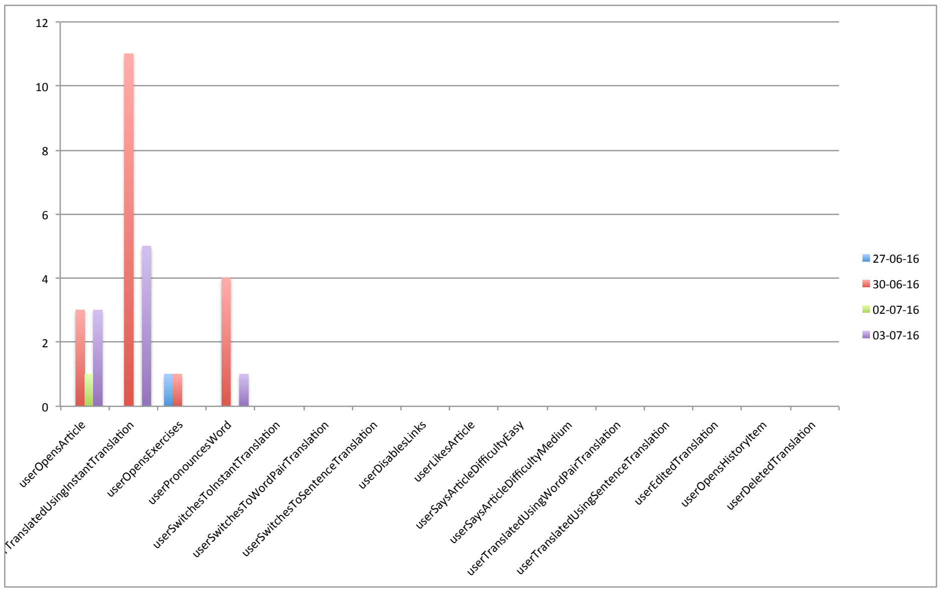


Figure 4.6: Usage data of user #6

translated not so many words. This user also turned off the pronunciation feature sometimes.

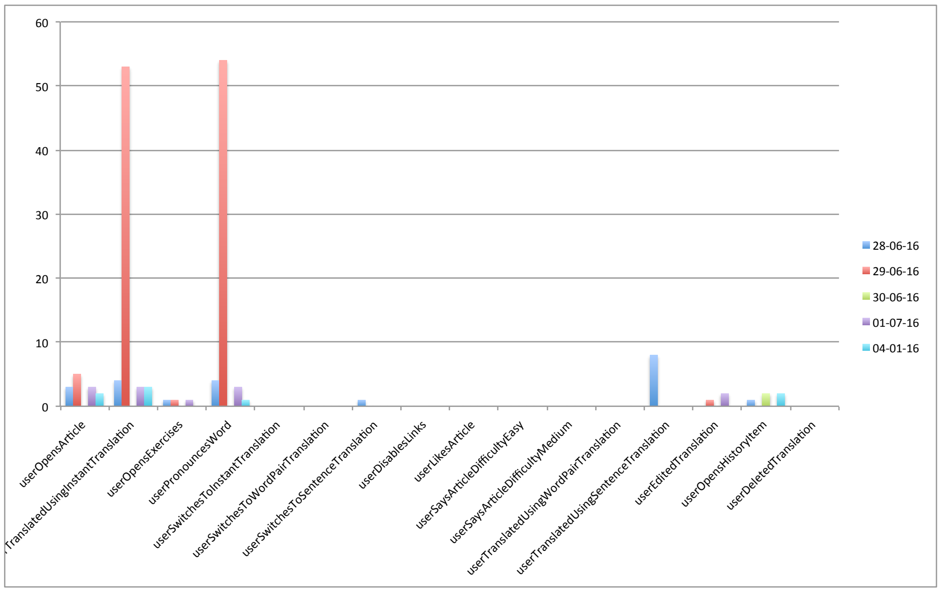


Figure 4.7: Usage data of user #7

Figure 4.7 shows the usage data of user #7. This user read many articles, but not too many per day. this user translated a huge amount of words. This user has a language level of C1, so it is possible that he stumbled on an extremely hard article or he was playing with the translation feature. Another possibility is that he did not have the translations inserted in the text and had to retranslate a lot of words. He also left the automatic pronunciation option turned on and even had the app pronounce some words again. This user also found the sentence translation mode and had it translate a few pieces of text. Furthermore, this user is the only user having looked at the history of his translations and is the other of two users to discover the existence of the feature to edit translations.

Chapter 5

Results

The collected user data revealed some interesting things. For example, only two users discovered the possibility to tap a translation, which shows a screen where the user can edit or delete a translation (Figure 3.7b, page 23). In hindsight, this makes sense, as the users needs to tap the translation to get to this screen. If that is not known to the user, he has to tap the translation by accident. The possibility to edit the translation by tapping it is also not documented in the app, nor in the welcome screen.

User	Level	Words translated	Articles opened	Average	Median
1	A1	48	32	1.50	7.5
2	B1	34	11	3.09	12
3	B1	47	9	5.22	12
4	B2	15	16	0.94	2
5	B2	16	21	0.76	4
6	B1	16	7	2.29	2.5
7	C1	63	13	4.85	3

Table 5.1: The average and median amount of words each user translated per article

Table 5.1 shows that users translate only a few words per article, with the exception of user #7. The median of user #7 is lower than his average, which means that he has translated a lot of words within one article. Users #2 and #3 have a median way higher than their average, which means that they have read articles without translating words. The users translating only a few words per article can only mean that all users understood what they were reading and that they didn't need that many words translated. As every user has translated a word at least once, it cannot be the case that they didn't know about the possibility of translating, especially because a descriptive popup about the 'tap-to-translate' feature is shown when an article is opened. The reason that user #7 translated a lot of words can mean two things. Either this user just had to translate a lot of words due to the article being hard or this user disabled having the translations inserted in the text, giving him the need to retranslate words he did not remember immediately. Sadly, the app does not send an event for this particular setting.

Another thing that stands out is the fact that almost all users only use the 'instant' translation feature. Only a few users switched to the other translation mode, but they did not translate a lot using these modes. This can have two causes. Either the users did not need to translated word pairs or sentences or they did not known about the existence of these translation modes. However, some of the users who did not use the other translation modes, did turn off the automatic pronunciation option, indicating that they must have seen all translation modes. Therefore, it is most likely the case that users were satisfied with the 'instant'/'single-word' translations or did not realize it if translations were inaccurate.

The post-study questionnaire delivered some insights as well. For example, the users who filled it in did like the fact that translations where inserted in the text. Most participants liked it very much. There was one person who said that he didn't mind it and that he would have liked it to have the option of a popup box above the clicked word. Essentially, this option is available, but currently, if this option is selected, the translation is briefly shown at the bottom of the screen, rather than as a popup above the tapped word.

Users were also quite happy about the given translations. This also indicates that the users didn't particularly feel the need to edit the given translations and that they didn't mind it if the translations were not accurate or did not realize the translations were wrong. Having the possibility to edit the translation was not given as a hint, so the users who didn't knew about the feature probably didn't miss it. There was also a user who said that he used on of the other translation modes to get the correct translation for some word, however this user didn't state whether he was happy about the given translation, but he did say he got "a lot of weird translations" (appendix D, answer 3d) for verbs, for which word pair or sentence translation worked better.

An issue that arose during the study was the fact that the Zeeguu server was not able to find RSS feeds for every URL that was entered by users. This was the reason to introduce a section with interesting feeds, in order to provide the user with alternative content sources, if his own preferences did not work right away.

5.1 Research questions

In the introduction (section 1, page 5), some questions were posed which should be answered by the user study:

1. **Do the learners like such an application that allows them to read texts on their iOS device?**
   The users are generally positive about the app. They liked the interaction of translating words by tapping them and seeing the translation within the text. However, they did point out improvements that can be made to make an even better app. There are even users still using the app after the user study ended, which indicates that these users like the app to read texts in a second language on their iOS device.

   - **Do users generally read on iPhone or iPad?** 6 out of 7 users have installed the the app on their iPhone, whereas 2 out of 7 users installed the app on their iPad. It can be concluded that a majority of these users read on their iPhone.
   - **How long do they use the app?**
     The users who did some reading have all used the app for a few days. Per day, the users were using the app ranging from 5 minutes up to half an hour. It should be noted that the app measures on-screen time, therefore it is not guaranteed that the users were actually reading for half an hour, however, the usage time is a sum of the whole day. Some users have read somewhere in the morning and than later in the afternoon they did some reading again.

2. **Which of the features of the application are the most used by the users?** As seen in section 5, users mainly read articles and translate words. All other features of the app are not used very much. Some users did use or try some of the other features, but most users stuck to reading and translating single words. However, most users did have a look at the exercises and some users visited the exercises two or three times. Because the exercises are based on earlier translations, they was not enough data yet for exercises as most users created an account to use the app, which means that they did not have a lot of translations to create exercises from.

   - **Do users like the interaction of tapping a word and having the translation inserted in the text?**
     Users do like the translations being inserted in the text. The post-study questionnaire received mostly positive reactions about this feature. One user did have a side note about having a popup above the tapped word, which could be the replacement of the current optional feature which shows translations briefly at the bottom of the screen instead of within the text.
   - **Do users use the pronunciation feature?**
     The pronunciation feature was an idea of one of the linguistics students, while we discussed the application with them. Some users seem to like this feature and others don't. As the graphs in section 4.1 show, some users had this feature turned off and others had it turned on. It is good to note that the feature is turned on by default. During this study, only

two participants left the feature turned on. As a conclusion, one could argue that users don't like this feature, however, turning it off can have a few reasons, not relating to whether users like it or not. If one uses the Zeeguu Reader mainly in public, say while traveling by bus or train, it is only logical to turn this feature off, as well as when playing music. Having it turned off once, users may forget to turn it on once they are reading in private or using headphones.

- **Which way of translating words/text do users use the most?**
  The 'instant' translation option, which translates only a single word is definitely the most used form of translation. It is likely that users also like this form of translation the most, as it involves just tapping a word and the translation appears, which is far more easier, compared to the other translation modes. It could also be the case that the other two translation modes were unknown to the user, however that does not seem likely, as most users have seen the options popover (Figure 3.6b, page 22).

Chapter 6

Conclusions and Future Work

The Zeeguu Reader for iOS is not finished yet. Issues arose both in functionality and in usability. Some features where not clearly indicated and others had issues.

The study revealed that most users probably didn't know about the possibility to edit translations and therefore the app should make it clear that the translation can be tapped, either by telling the user specifically about the feature or giving the translation an indication that it can be tapped, like underlining, as if it were a link. Of course, implementing both options is also a possibility.

A big issue was adding feeds. Some users had trouble adding their feeds. As explained in section 3.2.1, the Zeeguu Reader for iOS tries to hide the fact that it uses RSS to get the articles. However, because not every website announces its RSS feeds in the `head` section of the HTML source, the Zeeguu server did not find these RSS feeds and it is hard to find these feeds automatically. Therefore, it would be better to still try to hide the RSS, but when no feeds are found, to give an option to the user to manually enter the RSS feed(s) he wants to follow.

Despite some issues, the users generally liked the app and some even still use the app. The users liked the interaction of tapping a word and getting the translation, whereas immediately hearing the pronunciation was not that popular.

By conclusion, the user study with the Zeeguu Reader for iOS indicates that this concept of reading texts in a foreign language and inserting translations within the text is a promising way of encouraging second language learners to read articles written in the language they are learning. However a more longitudinal study is needed to find out if this way of reading and translating words helps with learning a second language.

6.1 Future Work

During the development and user study of the Zeeguu Reader for iOS, many ideas for future improvements and features arose.

Because the Zeeguu Reader for iOS uses RSS feeds to collect articles, it was quite easy to just name the news sources feeds. The goal was to hide RSS, but the concept of feeds was not really hidden because the app mentions feeds. An improvement would be to rename the feeds within the app to, for example, ‘News sources’.

The results of the user study showed that not all users used the options to translate word pairs or sentences. To encourage users to switch between translation modes, a toolbar could be added, allowing users to switch translation modes without them having to open the ‘Options’ menu. Of course, this toolbar should not be limited to only switching translation modes, but other popular or useful settings could be added too.

The Zeeguu Reader for iOS uses Readability to show the articles in a clean way to the user. To make the app future proof, the use of Readability could be optional. The alternative would be to show the original webpage with the article. This ensures that the app is not dependent on Readability and that users can continue using the app if Readability does not work any longer.

There was a user (appendix D, answer 1a) who mentioned that translations took a long time. An improvement would be to make the translations faster by caching or prefetching the most likely words to be translated.

This user also mentioned the toolbar with the like button and easy/medium/hard buttons. He didn’t know what the easy/medium/hard buttons are for. The purpose of these buttons should be indicated somewhere. Also, at the moment, the buttons do nothing except sending an event to the server. In the future, these buttons could be used to improve the algorithm that calculates how difficult a text is going to be for a user.

Additionally, the user behind answer 1c mentions that the app is limited to RSS feeds. This is indeed true. To resolve this problem, the app could feature a web browser, allowing the user to browse any website they like, which would be similar to the experience with the Chrome extension (chapter 2).

The answers of question 2 from the post-study questionnaire (appendix D) mentioned that users would have used the app more if they would have received notifications saying it is time to read or when a new article is published. A system of reminders, using push notifications for example, could increase engagement with the application.

Appendix A

Requirements

This chapter describes the original requirements of the Zeeguu Reader iOS app by the help of a user story.

A.1 Persona

To scope the requirements of the application, the news reader for iOS should have in mind the following persona:

- Jenny, a British young woman who lives in the Netherlands, studying psychology at the University of Groningen. She owns an iPad mini and the latest version of the iPhone.
- She has taken several beginner level courses in Dutch, and she is now at the level where she can speak the basics of the language, she can read an easy book or an easy news article, but still there are many words that she encounters and are new for her. Thus reading a paper based book or a newspaper is too difficult since she would have to constantly look up words in the dictionary.
- Moreover, she would like to improve her vocabulary in Dutch or at least learn some of the most frequently encountered words.
- She reads news in English on her iPad mini
- She does not want to read language textbooks in Dutch anymore. They are boring. She wants to read interesting news in Dutch. Maybe general news, or even news about specific subjects of her interest, but these texts should not be too hard.

- She wants to be able to have her word list synced across devices, such that she can rehearse when she is waiting in the bus station or traveling by train.

A.2 Factors

To address this persona a news reader should take into account the following factors:

- It would probably not be the main news reader for Jenny, so article management is not the most important feature. The graphics of how the articles are presented for example is not important. A simple list with the articles and first few lines should be sufficient.
- It should be able to remember the current article when the app is restarted.
- It should make it really easy to translate a word that the user does not understand.
- The reader should start with a welcome screen. After that the user should either log in or create an account.
- A setting page should allow the user to login, logout.
- The words that have been looked up in the past should be stored in a database, such that if the user wants to review them it should be possible.
- The app requires a connection to the internet for the translation functionality to be available.
- The user should not be aware of the existence of RSS. The user should be able to insert a link to their news site/blog of choice (in the target language of course) and the app should find the RSS information itself.

A.3 Screens

The app should have the following screens:

- **Welcome screen**
  Short description of the project.
- **Login / Create account page**
  Users should be able to login using their existing account or create an account if they don't have one yet.

- **Article list**
  List with articles, add news source, delete news source should be available, maybe also the possibility to only see articles from one news source or all.
- **Article reader**
  Single page. Font size should be changeable. A readable font should be default, if well chosen no need to provide an option of changing it.
- **Account/Settings screen**
  A screen where the user can change their base language and the language they learn, as well as log out.
- **History page**
  A list of previously translated words, sorted by date.

Appendix B

Pre-Study Questionnaire

The pre-study questionnaire asked the following questions to potential participants:

1. **Name**
2. **Email**
3. **Gender**
   - Male
   - Female
4. **What is your approximate age?**
   - Below 18
   - 18 - 24
   - 25 - 34
   - 35 - 44
   - 45 - 54
   - 55 and over
5. **What is your current educational status?**
   - Doing my Bachelors
   - Doing my Masters
   - Doing my PhD
   - I've graduated from the university
   - Other

6. **What is your native language?**

7. **Which language are you learning?**
   If you are currently learning multiple languages, please select the one you want to improve using Zeeguu.

   - Dutch
   - French
   - Italian
   - German
   - Spanish

8. **What is your current language level in this language that you are learning?**

   - A1 - Beginner. Can recognize and use simple phrases
   - A2 - Elementary. Using simple words, can describe his or her surroundings and communicate immediate needs
   - B1 - Intermediate. Can understand the main points of clear standard speech. Can narrate an event, an experience or a dream.
   - B2 - Upper Intermediate. Can speak in a clear, detailed way on a number of subjects; express an opinion on current affairs, giving the advantages and disadvantages of the various options.
   - C1 - Advanced. Can use the language effectively and fluently in a social, professional or academic context.
   - C2 - Master. Can express him or herself precisely in a spontaneous, fluent way, conveying finer shades of meaning precisely

9. **For how many years have you been studying this language?**

   - Lass than 1 year
   - 1 - 2 years
   - 2 - 3 years
   - More than 3 years

10. **Are you living in a country where people speak the language you are learning?**

    - Yes
    - No

11. **What way of learning the language are you currently using?** (multiple choice)

    - I'm not learning it yet, but I will

- Learning texts in the other language
- Using a textbook for the other language
- Talking/chatting with other people speaking the foreign language
- Other

12. **How do you translate the words you don't understand in texts written in the language you are currently learning?**
    Do you use a dictionary application / website? If so, which? Do you use a browser plugin? If so, which?

13. **What is your current motivation for trying out the Zeeguu apps?**
    - Improving my current language skill
    - I am curious about the software
    - Maintaining my current level of language
    - Testing my current level of language knowledge
    - I just want to provide feedback to the app creators
    - Other

14. **What is your usual web browser?**
    - Google Chrome
    - Firefox
    - Internet Explorer
    - Safari
    - Other

15. **Do you have a website that you visit frequently from your computer, even if you feel that you should be visiting it less?** (multiple choice)
    This info is optional even if it is very relevant for a later study. Answering it now would be cool!
    - Facebook
    - Twitter
    - Instagram
    - Other

16. **Which of the following are interesting for you as a means for reading new texts in your learning language?** (multiple choice)
    In the current study we are interested in people who want to read on their iPhone, iPad or in their Chrome Browser. In the future we will run another study for those who use other browsers, or other types of mobile devices.
    - I would like to read texts on my iPhone or iPad

- I would like to read texts in my Chrome browser
- I would like to read texts on my Android phone or tablet
- Other

17. **Would you be willing to spend at least 15 minutes every day studying your second language with Zeeguu?**

    - Yes for one week
    - Yes for two weeks
    - No way! I don't have that much time!
    - Other

18. **How did you find out about Zeeguu?**

Appendix C

Post-Study Questionnaire

The post-study questionnaire asked the following questions to participants:

1. **Name**
2. **Email**
3. **What did or didn't you like about the app?**
4. **What would have made you use the app more?**
5. **Were you happy about the given translations?**
6. **Did you like that the translations were inserted in the text?**
7. **If not, why not?**
8. **Did you like the exercises?**
9. **If not, what could have been better?**
10. **Did you miss any features?**
11. **Was the app usable on your device?**
    Like, were features there where you'd expect them, buttons/text large enough on iPhone etc.
12. **Would you recommend the Zeeguu Reader to your friends?**
    - Yes
    - No
13. **Wat score would you give the Zeeguu Reader for iOS?**

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10

14. **Other remarks, tips or feedback**

Appendix D

Post-Study Questionnaire Answers

The post-study questionnaire was answered by 6 participants (designated with letters from a to f) as listed below. The answers of one participant are discarded and not shown, because this participant did open the app only once and did not generate useful data (only 3 events, excluding some usage events). His answers on the post-study questionnaire came across as trolling and thus this appendix contains only the answers of the 5 serious participants.

1. **What did or didn't you like about the app?**

   (a) Did like: overview of news items of different sites. Didn't like: the sometimes long time is takes to translate a word. When I clicked several words it didn't translate anything in the article. In the news site Die Bild some articles start with only links and no text to read, even when I switch off links. It would come in handy to see switching off is applicable the whole time instead of switching it off in every article. In the bottom there are bottoms for easy, medium and hard, but I don't know where I should use them for.

   (b) I had to reset my password which was complicated, which dissuaded me from using the app (already emailed with team about this). Once I used it, I really liked the words both showing in context in the story in translation + the pronunciation. I used the feeds that were pre-populated in the app, which worked well for me, but the NOS feed wouldn't work consistently, and that's actually the main Dutch news source I read.

   (c) The layout of the documents wasn't as good as with some other RSS readers. Limited to RSS feeds. The inline translation option was great.

   (d) When clicking on articles, more articles were loaded. Instead of the article selected.
   (e) More features:
       - I like that i can edit the word or add my own translation
       - translate pair of words/more words is cool
       - the history of words

2. **What would have made you use the app more?**
   (a) Popup: "time to read" at an indicated time. Or notify the user when there is an article about a favourite subject. And maybe highlight the words that are already added, in new articles
   (b) A better user interface. I'd like to see the images from the stories... Maybe the option to set a daily notification to read. Also, I think it would be great to have a Quizlet-style review of the words I translated and/or allow me to easily export the list to Quizlet so I can review with the randomization to make sure I actually remember the words!
   (c) Browsing other pages. Better layout. Blendle support.
   (d) Nothing really specific
   (e) - Really trying to learn a language at the moment:) which isn't the case
       - Having a version for English, with thesaurus

3. **Were you happy about the given translations?**
   (a) Yes!
   (b) as far as I can tell, yes! :)
   (c) Yes, much better than expected.
   (d) I learned later that you could also translate multiple words or a sentence, this works especially better for verbs since I got a lot of weird translations for that.
   (e) Yes, most of the time it was the right translation

4. **Did you like that the translations were inserted in the text? If not, why not?**
   (a) I didn't mind it, but it would be nice to have an option to choose for an popup box above the clicked word.
   (b) Yes, very much
   (c) Yes. Worked well (if you only need to translate a couple of words)
   (d) See previous answer, for verbs usually you have to select more words to get a better translation. Other than that translations are OK.
   (e) Yes, it's a brilliant idea. It's cool to visualise it and be able to come back, or see how many words you didn't know from the text when finished with an article. if forgetting the meaning, you don't have to click again on the word. it's cool.

5. **Did you like the exercises? If not, what could have been better?**
   (a) I didn't do the exercises, because I was interested in reading. Maybe it would be nice to leave out words (that are already selected) or show the translation in new articles so you are learning while reading.
   (b) Umm... what exercises? :) Was this something I could access on the iOS app? I would say that I couldn't find the exercises, so, no, I wasn't happy with them on iOS. I like to study with "gamification" on my iPhone, so if it's web only, I don't necessarily spend the time. My comptuer feels like work. Studying Dutch is both education + leisure.
   (c) n/a
   (d) Exercises are OK, but doing them on the phone is not convenient for me.
   (e) i haven't tried the exercises :/
6. **Did you miss any features?**
   (a) See the answer above
   (b) not sure
   (c) see above
   (d) Maybe to save an article you read, so you can later come back to it.
   (e) There could be some other features
7. **Was the app usable on your device?**
   Like, were features there where you'd expect them, buttons/text large enough on iPhone etc.
   (a) Yes!
   (b) It was OK, but a very basic design.
   (c) yes, on a ipad
   (d) Everything looked fine on a iphone 4s.
   (e) Yes, did not have problems
8. **Would you recommend the Zeeguu Reader to your friends?**
   (a) Yes
   (b) Yes
   (c) No
   (d) Yes
   (e) Yes
9. **Wat score would you give the Zeeguu Reader for iOS?**
   (a) 6

   (b) 6
   (c) 7
   (d) 8
   (e) 8

10. **Other remarks, tips or feedback**
   (a) See the answers above :)
   (b) Good start!
   (c) Great idea. Well executed for a first project. With additional development, this could be a killer-app
   (d) The articles problem mentioned earlier on is huge. When I click on article I want to read that article and not get other suggestions.
   (e) Some suggestions:
   - would be useful to have the already read articles in the left column marked as read, as well as current article=> it happened to get out of the article and then needed some time to find it again in the list!
   - when editing translation, it feels like should be a button 'Update' rather than only enter (Update Translation should do that, doesn't work)
   - would be nice to be able to tick the words from the History, when I think I know them, have a visual representation of them (eg. green tick) (and maybe reorder them on the column:))
   - be able to delete words from history

Appendix E

Web View Interaction

E.1 Example German paragraph

Listing E.1 shows an example paragraph in HTML. The original paragraph text from the website[1] without HTML tags is listed below:

Schneider wies darauf hin, dass es in der Anhörung des Finanzausschusses zahlreiche Bedenken gegeben habe, ob das Ziel der Förderung insbesondere des sozialen Wohnungsbaus mit dem vorgelegten Gesetzentwurf erreicht werden könne. ...

Listing E.1: Example German paragraph in HTML

```
<p class="MsoNormal">
    <zeeguuparagraph>
        <zeeguuword>Schneider</zeeguuword>
        <zeeguuword>wies</zeeguuword>
        <zeeguuword>darauf</zeeguuword>
        <zeeguuword>hin</zeeguuword>,
        <zeeguuword>dass</zeeguuword>
        <zeeguuword>es</zeeguuword>
        <zeeguuword>in</zeeguuword>
        <zeeguuword>der</zeeguuword>
        <zeeguuword>Anhörung</zeeguuword>
        <zeeguuword>des</zeeguuword>
        <zeeguuword>Finanzausschusses</zeeguuword>
        <zeeguuword>zahlreiche</zeeguuword>
```

[1]http://www.readability.com/m?url=http://www.handelsblatt.com/politik/deutschland/mietwohnungsbau-spd-brueskiert-eigene-ministerin/13832808.html

```
        <zeeguuword>Bedenken</zeeguuword>
        <zeeguuword>gegeben</zeeguuword>
        <zeeguuword>habe</zeeguuword>,
        <zeeguuword>ob</zeeguuword>
        <zeeguuword>das</zeeguuword>
        <zeeguuword>Ziel</zeeguuword>
        <zeeguuword>der</zeeguuword>
        <zeeguuword>Förderung</zeeguuword>
        <zeeguuword>insbesondere</zeeguuword>
        <zeeguuword>des</zeeguuword>
        <zeeguuword>sozialen</zeeguuword>
        <zeeguuword>Wohnungsbaus</zeeguuword>
        <zeeguuword>mit</zeeguuword>
        <zeeguuword>dem</zeeguuword>
        <zeeguuword>vorgelegten</zeeguuword>
        <zeeguuword>Gesetzentwurf</zeeguuword>
        <zeeguuword>erreicht</zeeguuword>
        <zeeguuword>werden</zeeguuword>
        <zeeguuword>könne</zeeguuword>
        <zeeguuperiod id="zeeguuPeriod16">.</zeeguuperiod
           >
        ...
    </zeeguuparagraph>
</p>
```

Appendix F

Get context algorithm

Listing F.1 shows the algorithm that is used to get to context of a tapped word. Note that each word is enclosed in `<zeeguuword></zeeguuword>` tags and that walking over siblings involves walking over `zeeguuword` elements.

Listing F.1: The get context algorithm

```
zgjq = jQuery.noConflict(true);
var zeeguuInlineTextElementsToWalkThrough = ["a", "b", "i
    ", "u"];
var zeeguuParagraphTagName = "zeeguuParagraph";
var zeeguuTranslatedWordTagName = "zeeguuTranslatedWord";
var zeeguuPronounceTagName = "zeeguuPronounce";
var zeeguuPeriodTagName = "zeeguuPeriod";

function isWalkThroughElement(el) {
    return zeeguuInlineTextElementsToWalkThrough.indexOf(
        el.tagName.toLowerCase()) != -1;
}

function enterParagraphOutSideCurrent(el,
    directionIsPrevious) {

    var siblingProperty = directionIsPrevious ? "
        previousSibling" : "nextSibling";
    var firstLastChildOfParagraph = directionIsPrevious ?
         "firstChild" : "lastChild";
    var firstLastChildOfLink = directionIsPrevious ? "
        lastChild" : "firstChild";

    var parentSibling = null;
    var isInside = false;
    if (isWalkThroughElement(el.parentNode.parentNode)) { // The parent of el (zeeguuParagraph) has a parent that is in the walkthrough list (such as 'a')
        isInside = true;
        parentSibling = el.parentNode.parentNode[siblingProperty];
    } else {
        parentSibling = el.parentNode[siblingProperty];
    }

    if (parentSibling == null) {
        return null;
    }

    if (el == el.parentNode[firstLastChildOfParagraph] && parentSibling.nodeType != 3 /* is not a text node */ && isWalkThroughElement(parentSibling)) {

        // There is a link (or bold, etc.) next to the parent
        // Assume that each 'a' element has a zeeguu paragraph as first child

        var zeeguuParagraph = el.parentNode[siblingProperty].firstChild;
        return zeeguuParagraph[firstLastChildOfLink];
    } else
        // We are in a link (or bold, etc.) and want to continue in the adjoining zeeguuParagraph
        if (isInside &&
            el == el.parentNode[firstLastChildOfParagraph] &&
            parentSibling.nodeType != 3 /* is not a text node */ &&
            parentSibling.tagName.toLowerCase() == zeeguuParagraphTagName.toLowerCase()) {



        return parentSibling[firstLastChildOfLink];
    }
    return null;
}
```

```

function walkElementsStartingWith(element,
   directionIsPrevious, callback) {
    var siblingProperty = directionIsPrevious ? "
       previousSibling" : "nextSibling";

    var text = "";
    var siblingElement = element[siblingProperty];
    while (siblingElement != null) {
        var currentElement = siblingElement;
        siblingElement = siblingElement[siblingProperty];

        if (callback != null) {
            var str = callback(currentElement,
               directionIsPrevious);
            if (str === "continue") continue;
            if (str === "break") break;
        }

        if (siblingElement == null) {
            siblingElement = enterParagraphOutSideCurrent
               (currentElement, directionIsPrevious);
        }
    }
    return text;
}

function elementIsPeriod(el) {
    return el.tagName && el.tagName.toLowerCase() ==
       zeeguuPeriodTagName.toLowerCase();
}

function elementIsTranslation(el) {
    return el.tagName && el.tagName.toLowerCase() ==
       zeeguuTranslatedWordTagName.toLowerCase();
}

function elementIsPronounceIcon(el) {
    return el.tagName && el.tagName.toLowerCase() ==
       zeeguuPronounceTagName.toLowerCase();
}

function getContextNextTo(element, directionIsPrevious) {
    var text = "";

    walkElementsStartingWith(element, directionIsPrevious
        , function (currentElement, directionIsPrevious) {
        if (elementIsTranslation(currentElement) ||
            elementIsPronounceIcon(currentElement)) {
            return "continue";
        }

        if (!directionIsPrevious) {
            text = text + zgjq(currentElement).text();
        }

        if (elementIsPeriod(currentElement)) {
            return "break";
        }

        if (directionIsPrevious) {
            text = zgjq(currentElement).text() + text;
        }
    });

    return text;
}

function getContextOfClickedWord(wordID) {
    var el = document.getElementById(wordID);

    var text = zgjq(el).text();

    text = getContextNextTo(el, true) + text;
    text = text + getContextNextTo(el, false);

    return text.trim();
}
```

Appendix G

JavaScript action

Listing G.1 shows the `ZGJavaScriptAction` enum. In Swift, enums can have some variables attached to them. Swift enums also support methods. For example, the `ZGJavaScriptAction` enum provides a method to get the JavaScript code that corresponds to the action.

Listing G.1: The `ZGJavaScriptAction` enum

```
/**
Holds a JavaScript action to be executed.
*/
enum ZGJavaScriptAction {
    /// No action
    case None
    /// The translate action.
    ///
    /// Use the `ZGJavaScriptAction.
       getJavaScriptExpression` method to retrieve a
       JavaScript expression that will insert the
       translation behind the original word.
    ///
    /// **Important**: Before using `ZGJavaScriptAction.
       getJavaScriptExpression`, use `ZGJavaScriptAction.
       setTranslation` to set a translation, to make sure
        the JavaScript expression can be created!
    case Translate(Dictionary<String, String>)
    /// The edit translation action.
    ///
    /// Use the `ZGJavaScriptAction.
       getJavaScriptExpression` method to retrieve a
       JavaScript expression that will update the
       translation behind the original word.
    ///
    /// **Important**: Before using `ZGJavaScriptAction.getJavaScriptExpression`, use `ZGJavaScriptAction.setTranslation` to set a new translation, to make sure the JavaScript expression can be created!
    case EditTranslation(Dictionary<String, String>)
    /// The delete translation action. The value is the JavaScript element id of the HTML element that displays the translation.
    case DeleteTranslation(String)
    /// The change font size action. The value indicates the factor of change (1 = +10%, -1 = -10%, ...)
    case ChangeFontSize(Int)
    /// The change translate mode action. The value indicates the translation mode.
    case ChangeTranslationMode(ArticleViewTranslationMode)
    /// The enable/disable links action. The value indicates whether links should be disabled or not.
    case DisableLinks(Bool)
    /// The remove selection highlights action. This action will remove selections of word groups that were selected for translation.
    case RemoveSelectionHighlights
    /// The selection incomplete action. If this action was parsed, it means a selection between two words is incomplete and that the user tapped a second word outside the paragraph of the first word. This is not supported yet.
    case SelectionIncomplete
    /// The pronounce action. If this action was parsed, the given word is pronounced by iOS. The string is the word/phrase to pronounce.
    case Pronounce(Dictionary<String, String>)
    /// The set inserts translation action. Sets whether the translation will be inserted or not. If the translation is not inserted, it is possible to translate a word multiple times.
    case SetInsertsTranslation(Bool)
    /// The insert loading icon action. The string contains the id of the element after which to put the loading icon.
    case InsertLoadingIcon(String)
    /// Send a post request with a URL, Method and POST
```

```
        parameters. Use this if you want to load a POST
        request with parameters using WKWebview. WKWebView
         ignores the HTTPBody of an NSURLRequest by
        default.
    case SendPOSTRequest(String, String, String)
    /// Get the page as an HTML string
    case GetPageHTML
    /// Get the page text as a string
    case GetPageText

    static func parseMessage(dict: Dictionary<String,
        String>) -> ZGJavaScriptAction {
        var dict = dict
        guard let action = dict.removeValueForKey("action
            ")  else {
            return .None
        }
        if action == "translate" {
            if let _ = dict["word"] {
                return .Translate(dict)
            }
        } else if action == "editTranslation" {
            if let _ = dict["oldTranslation"], _ = dict["
                originalWord"] {
                return .EditTranslation(dict)
            }
        } else if action == "selectionIncomplete" {
            return .SelectionIncomplete
        } else if action == "pronounce" {
            if let _ = dict["word"] {
                return .Pronounce(dict)
            }
        }
        return .None
    }

    mutating func setTranslation(newWord: String) {
        switch self {
        case var .Translate(dict):
            dict["translation"] = newWord
            self = .Translate(dict)
        case var .EditTranslation(dict):
            dict["newTranslation"] = newWord
            self = .EditTranslation(dict)
        default:
            break // do nothing
```

```swift
        }
    }

    mutating func setOtherTranslations(ot: String) {
        switch self {
        case var .EditTranslation(dict):
            dict["otherTranslations"] = ot
            self = .EditTranslation(dict)
        default:
            break // do nothing
        }
    }

    mutating func setBookmarkID(id: String) {
        switch self {
        case var .Translate(dict):
            dict["bookmarkID"] = id
            self = .Translate(dict)
        default:
            break // do nothing
        }
    }

    mutating func setPronounceID(id: String) {
        switch self {
        case var .Translate(dict):
            dict["pronounceID"] = id
            self = .Translate(dict)
        default:
            break // do nothing
        }
    }

    func getActionInformation() -> Dictionary<String,
       String>? {
        switch self {
        case let .Translate(dict):
            return dict
        case let .EditTranslation(dict):
            return dict
        case let .Pronounce(dict):
            return dict
        default:
            return nil
        }
    }

    func getJavaScriptExpression() -> String {
        switch self {
        case let .Translate(dict):
            guard let translation = dict["translation"],
                word = dict["word"], context = dict["
                context"], id = dict["id"], bid = dict["
                bookmarkID"], pid = dict["pronounceID"]
                else {
                fatalError("The ZGJavaScriptAction.
                    Translate(_) dictionary is in an
                    incorrect state!")
            }
            let t = translation.stringByJSEscaping()
            let c = context.stringByJSEscaping()
            let w = word.stringByJSEscaping()

            return "insertTranslationForID(\"\(t)\", \"\(
                w)\", \"\(c)\", \"\(id)\", \"\(bid)\",
                \"\(pid)\")"
        case let .EditTranslation(dict):
            guard let word = dict["newTranslation"], id =
                 dict["id"] else {
                fatalError("The ZGJavaScriptAction.
                    EditTranslation(_) dictionary is in an
                     incorrect state!")
            }
            let w = word.stringByJSEscaping()
            if let ot = dict["otherTranslations"] {
                let str = ot.stringByJSEscaping()

                return "updateTranslationForID(\"\(w)\",
                    \"\(id)\", \"\(str)\")"
            } else {
                return "updateTranslationForID(\"\(w)\",
                    \"\(id)\", null)"
            }
        case let .DeleteTranslation(id):
            return "deleteTranslationWithID(\"\(id)\")"
        case let .ChangeFontSize(factor):
            return "document.getElementsByTagName('body')
                [0].style.webkitTextSizeAdjust='\(100 +
                factor * 10)%'"
        case let .ChangeTranslationMode(mode):
            return "setTranslationMode(\(mode.rawValue));
                "
        case let .DisableLinks(disable):
            return "zeeguuLinksAreDisabled = \(disable ? "true" : "false"); zeeguuUpdateLinkState();"
        case .RemoveSelectionHighlights():
            return "removeSelectionHighlights();"
        case let .SetInsertsTranslation(inserts):
            return "setInsertsTranslation(\(inserts ? "true" : "false"));"
        case let .InsertLoadingIcon(id):
            return "insertIconAfterID(\"\(id)\");"
        case let .SendPOSTRequest(url, method, params):

            var json = "{ "
            let pairs = params.characters.split(",").map(String.init)
            for pair in pairs {
                let kv = pair.characters.split("=").map(String.init)
                let key = kv[0]
                let value = kv[1]
                json += "\"\(key)\": \"\(value)\","
            }
            json = String(json.characters.dropLast()) + "}"

            return ["function post(path, params, method) {\n",
                    "    method = method || \"post\"; // Set method to post by default if not specified.\n",
                    "    \n",
                    "    // The rest of this code assumes you are not using a library.\n",
                    "    // It can be made less wordy if you use one.\n",
                    "    var form = document.createElement(\"form\");\n",
                    "    form.setAttribute(\"method\", method);\n",
                    "    form.setAttribute(\"action\", path);\n",
                    "    \n",
                    "    for(var key in params) {\n",
                    "        if(params.hasOwnProperty(key)) {\n",
                    "            var hiddenField = document.createElement(\"input\");\n",
                    "            hiddenField.setAttribute(\"type\", \"hidden\");\n",
                    "            hiddenField.setAttribute(\"name\", key);\n",
                    "            hiddenField.setAttribute(\"value\", params[key]);\n",
                    "            \n",
                    "            form.appendChild(hiddenField);\n",
                    "        }\n",
                    "    }\n",
                    "    \n",
                    "    document.body.appendChild(form);\n",
                    "    form.submit();\n",
                    "}\n",
                    "post(\"\(url)\", \(json), \"\(method)\");"].reduce("", combine: +)
        case .GetPageHTML:
            return "document.documentElement.outerHTML.toString()"
        case .GetPageText:
            return "document.documentElement.outerText.toString()"
        default:
            return ""
        }
    }
}
```

Appendix H

List of Figures

Appendix I

List of Tables

Appendix J

List of Listings

Appendix K

Bibliography

[1] A. Trusty and K. N. Truong, "Augmenting the web for second language vocabulary learning," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, (New York, NY, USA), pp. 3179–3188, ACM, 2011.

[2] C. J. Cai, P. J. Guo, J. R. Glass, and R. C. Miller, "Wait-learning: Leveraging wait time for second language education," in *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, (New York, NY, USA), pp. 3701–3710, ACM, 2015.

[3] S. Marti, "A platform for second language acquisition through free reading and repetition," bachelor's thesis, University of Bern, Aug. 2013.

[4] K. Sethi, "Modelling the acquisition of natural language," bachelor's thesis, University of Bern, Aug. 2015.

[5] P. Giehl, "Zeeguu translate application — extending the Zeeguu platform to the Android device," bachelor's thesis, University of Bern, Aug. 2015.

[6] L. Schwab, "Using rss feeds to support second language acquisition," bachelor's thesis, University of Bern, June 2016.

[7] M. Lungu, K. Sethi, S. Marti, and L. Schwab, "The Zeeguu API - Modeling Learner Progress to Accelerate Vocabulary Acquisition," July 2016. doi:10.5281/zenodo.58569.

[8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

[9] D. Dearman and K. Truong, "Evaluating the implicit acquisition of second language vocabulary using a live wallpaper," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, (New York, NY, USA), pp. 1391–1400, ACM, 2012.