Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Studiengang Industrie-4.0-Informatik

Bachelorarbeit

von

Amirhossein **Hassannezhad**

Entwurf und Implementierung eines personalisierbaren Dashboards zur webbasierten Visualisierung von Qualitätsdaten aus der Elektronikfertigung

Design and Implementation of a Customizable Dashboard for Web-based Visualization of Quality Data from Electronics Manufacturing

(Erstellt mit LaTeX)

Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Studiengang Industrie-4.0-Informatik

Bachelorarbeit

von

Amirhossein **Hassannezhad**

Entwurf und Implementierung eines personalisierbaren Dashboards zur webbasierten Visualisierung von Qualitätsdaten aus der Elektronikfertigung

Design and Implementation of a Customizable Dashboard for Web-based Visualization of Quality Data from Electronics Manufacturing

Bearbeitungszeitraum: von 02. Januar 2024
bis 31. Mai 2024

1. Prüfer: Prof. Dr.-Ing. Christoph P. Neumann

2. Prüfer: Prof. Dr.-Ing. Josef Pösl

Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Ostbayerische Technische Hochschule
Amberg-Weiden

Selbstständigkeitserklärung

Name und Vorname des Studenten: **Hassannezhad, Amirhossein**
Studiengang: **Industrie-4.0-Informatik**

Ich bestätige, dass ich die Bachelorarbeit mit dem Titel:

Entwurf und Implementierung eines personalisierbaren Dashboards zur webbasierten Visualisierung von Qualitätsdaten aus der Elektronikfertigung

selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Datum: 31. Mai 2024

Unterschrift:

Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Ostbayerische Technische Hochschule
Amberg-Weiden

Bachelorarbeit Zusammenfassung

Student (Name, Vorname):	**Hassannezhad, Amirhossein**
Studiengang:	Industrie-4.0-Informatik
Aufgabensteller, Professor:	Prof. Dr.-Ing. Christoph P. Neumann
Durchgeführt in Firma:	Siemens AG, Amberg
Betreuer in Firma:	Daniel Jackschik
Ausgabedatum:	02. Januar 2024
Abgabedatum:	31. Mai 2024

Titel:

Entwurf und Implementierung eines personalisierbaren Dashboards zur webbasierten Visualisierung von Qualitätsdaten aus der Elektronikfertigung

---

Zusammenfassung:

Qualitätsexperten des Siemens Elektronikwerk Ambergs brauchen ein Dashboard für die Neuentwicklung ihrer hauseigenen Plattform zur Überwachung der Fertigungsqualität, das sich von Anwendern ohne IT-Fachkenntnissen einrichten und personalisieren lässt. Das Dashboard soll eine Übersicht aus für sie relevanten Qualitätsauswertungen anbieten und mit anderen Kollegen teilbar sein. Im Rahmen dieser Arbeit wird ein Konzept für die Umsezung des Dashboards entwickelt und Siemens AG eine Vorgehensweise zur Implementierung und Einbindung des Dashboards in ihre Systemlandschaft vorgeschlagen.

Schlüsselwörter: Dashboard, Webanwendung, UX, Responsive Web Design, Grid Layout, JavaScript, JSON, REST, Schnittstelle, Datenbank

Inhaltsverzeichnis

Danksagung **vii**

1 Einleitung **1**
1.1 Siemens Elektronikwerk Amberg 1
1.2 Abteilung für Fertigungsqualität 4
1.3 Motivation 5
1.4 Zielsetzung 5
1.5 Umfang 6

2 Theoretische Grundlagen **7**
2.1 Übersicht der Grundbegriffe 7
2.2 Hypertext Markup Language 8
2.3 Cascading Style Sheets 11
2.4 JavaScript 14
2.5 Document Object Model 14
2.6 JavaScript Object Notation 17
2.7 Representational State Transfer 17
2.8 Responsive Web Design 18
2.9 Responsive Grid Layout 18

3 Methodik **20**
3.1 Analyse 20
3.1.1 Stakeholder-Analyse 20
3.1.2 Anforderungsanalyse 21
3.1.3 Ist-Analyse 23
3.1.4 Stand der Technik 23
3.2 Konzepterstellung 24
3.3 Implementierungsmöglichkeiten 29
3.3.1 Muuri 30
3.3.2 Sortable 31
3.3.3 Gridstack 31
3.3.4 Fazit 31

4 Scout-NextGen Dashboard Konzept **32**
4.1 Architektur 32

4.2 Webanwendung 34
4.2.1 Entitäten 34
4.2.2 Anwendungslogik 36
4.3 Datenmodell 40
4.3.1 Strategie 40
4.3.2 Sicherheit 40
4.3.3 Datenmodell 41

5 Diskussion und Fazit **45**
5.1 Zusammenfassung 45
5.2 Evaluation 46
5.3 Einschränkungen 47
5.4 Ausblick 48
5.5 Fazit 48

6 Anhang: User Stories **49**

Hilfsmittel **68**

Quellcodeverzeichnis **70**

Abbildungsverzeichnis **71**

Tabellenverzeichnis **72**

Abkürzungsverzeichnis **73**

Glossar **76**

Literaturverzeichnis **79**

Danksagung

Dank sei meiner Familie... für alles! ♥

Ein großer Dank gebührt der Siemens AG und im weiteren Sinne meinen Arbeitskollegen[1], insbesondere meinem Ausbilder, meinen Führungskräften und meinem Betreuer, die mir immer Rückhalt gegeben und mich mit einer angenehmen Arbeitsatmosphäre beschenkt haben.

Ich bedanke mich herzlich bei meinen Professoren, denen ich fachlich und menschlich viel zu verdanken habe, insbesondere Prof. Neumann, Prof. Pösl, Prof. Aßmuth und auch allen Interessenten für die Zeit und Aufmerksamkeit, die Sie meiner Arbeit schenken.

[1] m/w/d

Kapitel 1

Einleitung

1.1 Siemens Elektronikwerk Amberg

Das Siemens Elektronikwerk Amberg (EWA) wurde 1989 gegründet [1]. Fertigung von speicher-programmierbaren Steuerungen (SPS), aber u.a. auch HMI-Geräten[1] gehören zu den Kernkompetenzen des EWA. Das EWA ist von der ehem. Bundeskanzlerin Angela Merkel besucht [2] und durch namhafte Foren ausgezeichnet worden, u.a. mit *Industrie-4.0-Award* in der Kategorie *Smart Factory* (dt. intelligente Fabrik) von der *Fachzeitschrift Produktion und ROI Management Consulting AG* [3] und als *digital lighthouse factory* (dt. digitale Leuchtturmfabrik) vom World Economy Forum (WEF) [4].

Den Auszeichnungen zugrunde liegen z.B. Fertigungstempo von etwa 1 Gerät pro Sekunde, und die kaum übertroffene Fertigungsqualität von 99,99885% [5], die sich mit 11,5 Defekte pro Million Fehlermöglichkeiten übersetzen lässt[2]. Um sich diese geringe Fehlerrate besser vorstellen zu können, hilft es einen Blick auf die Entwicklung der Fertigungsqualität im EWA werfen (Abbildung 1.1). Die Anzahl der Defekte ist seit der Gründung des Werkes von über 550 auf 11,5 pro Million gesenkt worden.

dpm–A

Defekte Pro Million – Anschlüsse (dpm–A) ist eine interne Bezeichnung für Defect Per Million Opportunities (DPMO). D.h., die Fehlerrate wird nicht im Bezug auf die Anzahl der gefertigten Geräte berechnet, sondern alle Fehlermöglichkeiten bei der Fertigung (1 Anschluss := 1 Fehlermöglichkeit beim Löten):

$$\text{dpm-A} = 10^6 \cdot \frac{\sum_i \text{Anzahl der defekten Anschlüsse im Gerät}_i}{\sum_i \text{Anzahl aller Anschlüsse im Gerät}_i}$$

[1] Human Machine Interface

[2] $100\% - 99,99885\% = 0,00115\% = 11,5 \cdot 10^{-6}$ Fehler pro Möglichkeiten

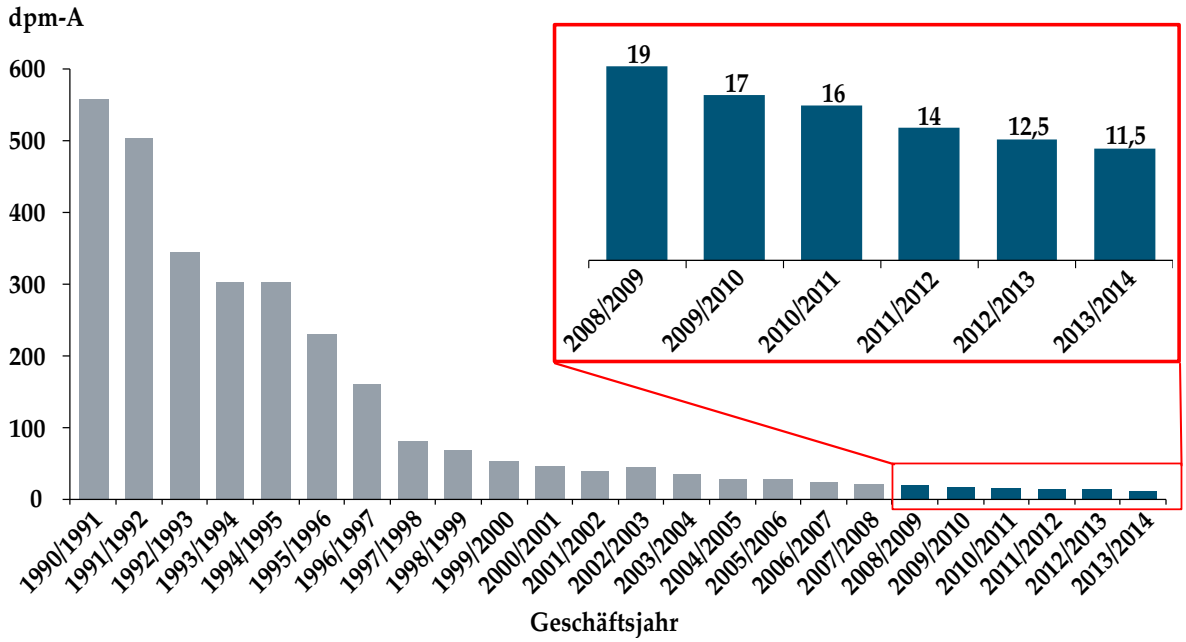

Abbildung 1.1: Zeitliche Entwicklung der Fertigungsqualität seit der Gründung des EWA bis Geschäftsjahr 2013/2014 (Quelle: [5, S. 4, grafisch bearbeitet])

Dies erspart dem EWA Zeit, Aufwand und Mehrkosten, die sonst in weitere Prüfungen und Reparaturen einfließen würden. Doch die Qualität bleibt nicht nur im Werk. Die erhöhte Fertigungsqualität sorgt für langlebigere, robustere Produkte und im weiteren Sinne für sogar weniger Ausfallrate und trägt somit zur erhöhten Kundenzufriedenheit bei, was wiederum durch Reduzierung der Reklamationen und Entlastung des Kundendiensts ressourcenschonend wirkt. Die verschwendungsarme Fertigung erhöht gewissermaßen auch die Fertigungskapazität, indem die zurückgewonnenen Aufwände und Ressourcen wieder in die Fertigung zurückfließen. Dieses Konzept wird in der Industrie-Welt als die schlanke Fertigung bezeichnet.

ℹ Schlanke Fertigung

(Engl. lean production/manufacturing) Die aus Japan stammende effizienzorientierte Fertigungsphilosophie, die nach Beseitigung von Verschwendung in vielerlei Hinsicht (zeitlich, logistisch etc.) **strebt**. [6, vgl. S. 10 u. 40]

Dank der schlanken Fertigung konnte das EWA seit der Gründung das Fertigungsvolumen bei fast derselben Kopfzahl und ohne Ausweitung der Werksfläche verachtfachen [1]. Bei der Umsetzung der schlanken Fertigung kam dem EWA die Industrie-4.0 sehr gelegen.

Die Industrie hat bis heute 4 große Reformen erlebt, die aufgrund ihrer immensen Bedeutsamkeit und Reichweite als die „Stufen der industriellen Revolution“ bezeichnet werden [7]:

1784: Industrie-1.0: **Mechanisierung**
1870: Industrie-2.0: **Elektrifizierung**
1969: Industrie-3.0: **Automatisierung**
heute: Industrie-4.0: **Vernetzung** von cyber-physischen Systemen (CPS)

Der vierten industriellen Revolution liegt nicht nur die Vernetzung von Anlagen zugrunde. Die Art von Anlagen, die vernetzt werden, spielt dabei eine wesentliche Rolle, nämlich die CPS.

Cyber-physische Systeme

Informationstechnische vernetzte Systeme, die durch Sensorik bzw. Informationsaustausch mit anderen cyber-physischen Systemen die Umwelt wahrnehmen und durch Aktorik entsprechend mit der Umwelt interagieren.

In diesem Kontext sind mit CPS daten-getriebene Fertigungsanlagen gemeint. Sie erfassen entweder selbst Daten von der Umwelt – bspw. durch Messung, Scannen eines Barcodes oder einer RFID an einem Produkt/Bauteil – oder bekommen sie Daten von ihresgleichen mitgeteilt bekommen (:= Vernetzung). Im Vergleich zu den Anlagen der letzten Generation, die durch Automatisierung gesteuert werden, greifen die CPS in den Regelkreis des Fertigungsprozesses ein und übernehmen teilweise die Steuerung. Das ermöglicht der Industrie eine neue Stufe der Autonomie und der damit verbundenen Flexibilität, weswegen CPS in EWAs Vision einer vollautomatisierten Fertigung eine zentrale Rolle spielen und im EWA umfänglich zu Einsatz kommen.

Die Anlagen können je nach Daten unterschiedliche Fertigungsschritte vornehmen. D.h., eine Linie bestehend aus CPS-Anlagen kann möglicherweise durch Bereitstellung der nötigen Daten als Fertigungslinie für beliebig viele Produkte dienen. Sie ersparen Herstellern mehrere produktspezifische Fertigungslinien, sind robuster gegen Ausfälle der einzelnen Anlagen, da jede gleichartige Maschine mit denselben Daten die Aufgabe (:= den Fertigungsschritt) übernehmen bzw. fortsetzen kann. Eine Redundanz der Maschinen gleicher Art kann ausreichend zeitlichen Puffer zur Wartung bzw. Ersetzung schaffen und somit eine praktisch ununterbrochene Fertigung gewährleisten.

Die Wichtigkeit der Daten bei der Industrie-4.0 verstärkt deren Rolle nicht nur zum Guten. Die Verlässlichkeit der Daten und ein ununterbrochener Datenfluss werden dabei zu einer unerlässlichen Voraussetzung für die Integrität der Fertigung. Somit wird die Qualität der Daten mit der Qualität der Produkte quasi synonym und im weiteren Sinne werden Daten selbst zu Produkten. Es gibt unterschiedliche Arten von Daten, die in einem Fertigungsprozess zustande kommen. Besonders interessant sind für diese Arbeit die Qualitätsdaten.

Qualitätsdaten (Definition nach Geiger)

„Daten über die Qualität von Einheiten, über die bei der Ermittlung der Qualität angewendeten Qualitätsprüfungen und über die dabei herrschenden Randbedingungen sowie ggf. über die jeweiligen Einzelforderungen an die Qualitätsmerkmale.“[8, S. 12]

Im Kontext dieser Arbeit versteht man unter Qualitätsdaten Prozessdaten, die bei der Fertigung von den CPS erfasst und ausgetauscht werden und wichtige Hinweise auf Ursache der Mängel in Fertigungsqualität liefern können. Aufgrund des hohen Produktionsvolumens von 15 Mio. Geräten pro Jahr [1] spielt die Qualität eine wichtige Rolle, weswegen sich eine große Abteilung im EWA damit beschäftigt.

1.2 Abteilung für Fertigungsqualität

Die Abteilung DI FA MF QM[3] besteht allein im EWA aus insgesamt 6 Unterabteilungen, die zusammen die durchgängige Fertigungsqualität verantworten:

Component Engineering (Elektronik/Mechanik) Die Qualität beginnt bei der Wahl der Bauteile und der engen Kommunikation mit den Lieferanten, Entwicklungs- und Einkaufskollegen durch die Component Engineers (CEs), die sich jeweils auf elektrische oder mechanische Bauteile spezialisieren.

Incoming Quality Die Gewährleistung der Qualität wird bei der Warenannahme mit der *Unikatisierung*, der Stichprobenprüfung und der fachgerechten Lagerung fortgesetzt. Das Letztere spielt insbesondere bei Waren mit einer begrenzten *Haltbarkeit* bzw. dem Bedarf an einer speziellen (z.B. klimatisierten) Lagerungsmethode eine wichtige Rolle.

Haltbarkeit

Haltbarkeit betrifft meist Surface Mounted Device (SMD)/Surface Mounted Technology (SMT) Bauteile. Z.B. mit Lötpaste gelieferte Leiterplatten dürfen nur zeitlich begrenzt ohne Qualitätsverlust zum Lötvorgang konfektioniert werden. Ähnliches gilt auch für die Kondensatoren, da diese bei Feuchtigkeitsaufnahme im Lötofen platzen würden.

Quality Engineering Spätestens jetzt beginnen die Qualitätsansprüche auf die eigene Fertigung. Die Daten über die *aufgetretenen Fehler* werden von den Quality Engineers (QEs) in Echtzeit überwacht, die durch enge Zusammenarbeit mit Fertigungskollegen zur kontinuierlichen Verbesserung der Fertigungsprozesse (KVP) beitragen, um ähnlichen Fehlern vorzubeugen.

[3]Digital Industries, Factory Automation, Manufacturing, Quality Management

Field Quality & Customer Support Sollten trotz größter Sorgfalt die Mängel nicht erst im Werk entdeckt werden, nehmen sich die hardware-affinen Experten Zeit für eine gründliche Untersuchung der Rückwaren. Die dabei *festgestellten Fehlerursachen* dienen als Feinschliff der Fertigungsqualität.

Quality Data Management Die unterstrichenen Begriffe haben etwas gemeinsam: Sie sind Qualitätsdaten und unabdingbar für die Qualitätsgewährleistung einer hochgetakteten Produktion. Immerhin sind die rohen Qualitätsdaten nicht ausdrucksstark für Qualitätsexperten. Sie müssen bereinigt, ausgewertet und visualisiert werden, Aufgaben, die informationstechnisches Fachwissen erfordern. Somit ist die Unterabteilung Quality Data Management (QDM) als Schnittstelle zwischen der Fertigung und den Qualitätsexperten für die Qualität und die Aufbereitung der Qualitätsdaten zuständig. Die o. g. Dienste werden durch das hauseigene Qualitätsdatenmanagementsystem *Scout* bereitgestellt.

1.3 Motivation

Im Rahmen der kontinuierlichen Verbesserung erfährt die Plattform *Scout* die Neuentwicklung *Scout-NextGen*. Die stark divergierenden Kompetenzen und Arbeitsroutinen lassen ein zentrales Cockpit wünschen, das sich Arbeitsroutinen und Anwenderbedürfnissen noch besser anpassen lässt.

Der Analyseaufwand besteht bei komplexen Qualitätsauswertungen, je nach Aufgabenstellung, in der Aggregation einer Vielzahl an unterschiedlichen Auswertungen, die ad-hoc generiert werden müssen, was viel Zeit kostet und im weiteren Sinne die Abteilung QDM mit unterstützenden Sonderauswertungen belastet.

Die Verzahnung zwischen Abteilungen, die sich für Fehler und deren Folgen wie z.B. Mehrkosten interessieren (die oben aufgeführten QM[4] Abteilungen, Einkauf, Finanz usw.), macht das Fehlen eines zentralen Hubs für die Zusammenarbeit bzw. einer gemeinsamen Ansicht des Aufgabenfortschritts spürbar. Somit wird der Autor damit beauftragt, ein Dashboard für den *Scout*-Nachfolger zu konzipieren und Vorschläge zur Implementierungsmöglichkeiten zu liefern.

Dashboard

(EDV) „Computerprogramm, das **relevante** Informationen **zusammenfasst** und **übersichtlich** darstellt.“ DUDEN

1.4 Zielsetzung

Kurzgefasst soll das Dashboard dem Anwender ermöglichen, selber eine oder mehrere **übersichtliche** Sammlungen aus für sie **relevanten** Auswertungen zusammenzustellen.

[4] Quality Management

heute Im heutigen *Scout* werden Auswertung bei Bedarf ausgesucht, parametriert und beim häufigen Gebrauch als Lesezeichen im Browser angelegt. Das hat zur Folge, das die Auswertungen als Lesezeichen organisiert und evtl. bei Verwendung eines neuen Browsers exportiert und wieder importiert werden müssen. Außerdem werden eine Vielzahl an Auswertungen in mehreren Browser-Fenstern geöffnet, was die Identifizierung der gerade gesuchten Auswertung bei generischen Tab-Überschriften und kleiner werdenden Tab-Namen nicht leicht macht.

morgen Das Dashboard soll den Anwendern des *Scout-NextGen* ermöglichen, mehrere Dashboards für das jeweilige Aufgabengebiet anzulegen. Hat der Anwender eine Auswertung einmal auf *Scout-NextGen* ausgesucht und parametriert, soll es ihm möglich sein, diese zu einem von ihm angelegten Dashboard hinzuzufügen. Das Dashboard soll von ihm ohne informationstechnische Fachkenntnisse und Rechte – außer ein authentifizierter Anwender zu sein – leicht umgestaltet werden, also die Auswertungen sollen umgeordnet und gelöscht werden können. Jedes Dashboard soll unter einem Link abrufbar und mit anderen Nutzern teilbar sein, sodass die Nutzer ihre Ansicht unter sich teilen können. Ein bewährtes Dashboard soll als Vorlage zur Erstellung von neuen Dashboards verwendet werden, damit man bei ähnlichen Aufgabengebieten nicht alles neu einrichten muss.

1.5 Umfang

Diese Arbeit wird von 2 teilweise divergierenden Interessen vorangetrieben:

- **Dem akademischen Interesse**: Wie geht man an die Aufgabe von einer höheren Abstraktionsebene heran? Welche „Fragestellungen“ liegen der Konzeption zugrunde, und das nicht nur bei einem Dashboard.
- **Dem geschäftlichen Interesse**: Wie passt das Konzept in die interne Randbedingungen der Siemens AG?

Das Letztere bietet nicht unbedingt einen akademischen Mehrwert, weil den internen Randbedingungen z. B. strategische Entscheidungen oder die bereits bestehende Systemlandschaft zugrunde liegen. Außerdem wird die Implementierung des Dashboards als Teil des Gesamtprojektes (Scout-NextGen) von weiteren Entscheidungen (z.B. der Wahl des Frameworks) betroffen sein. Aus dem Grund wird der Lieferumfang der Forschungsarbeit auf das akademische Interesse beschränkt.

Als Ergebnis dieser Arbeit zu erwarten sind ein Konzept bestehend aus Datenmodell und Datenschnittstelle, die Anwendungslogik, das Verhalten der Benutzeroberfläche (UX) und Vorschläge für Implementierungsmöglichkeiten. Wenn der zeitliche Rahmen es erlaubt, wird ein Prototyp aus hoch-prioren Grundanforderungen umgesetzt.

Die finale Implementierungmethode, Erscheinungsbild der Oberfläche (UI) oder in das Gesamtprojekt integrierbare Implementierung sind allerdings als Ergebnis dieser Arbeit nicht zu erwarten.

Kapitel 2

Theoretische Grundlagen

2.1 Übersicht der Grundbegriffe

In diesem Kapitel werden die relevanten Grundbegriffe erklärt. Der Zusammenhang der Begriffe wird in einem Übersichtsdiagramm veranschaulicht (Abbildung 2.1). Die wichtigsten Begriffe werden womöglich in der chronologischen Reihenfolge erklärt, damit die Motiven hinter der Erscheinung der Konzepte und Technologien bzw. Probleme, die durch sie gelöst werden sollen, nachvollziehbar sind.

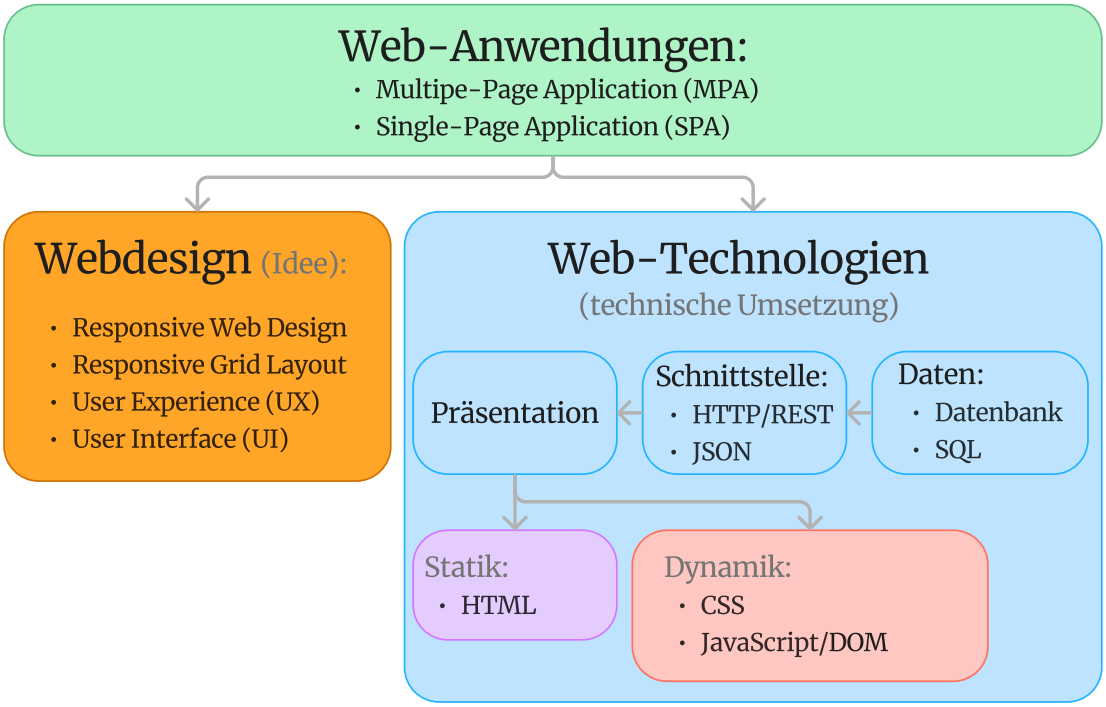

Abbildung 2.1: Übersicht der in dieser Arbeit behandelten Grundbegriffe (Quelle: eigene Darstellung).

2.2 Hypertext Markup Language

Geschichte Hypertext Markup Language (HTML) ist eine Sprache zur Beschreibung der statischen Struktur einer Webseite durch Markierungen (engl. markup/tag) mit Unterstützung für navigierbare Inahalte (engl. hypertext), die mit anderen Webseiten verlinkt sind (engl. hyperlink). Die erste HTML-Webseite (Abbildung 2.2) wurde 1989–1990 von Timothy Berners-Lee in CERN[1] – der europäischen Organisation für Kernforschung – in Genf in der Schweiz entwickelt, um eine weltweite Zusammenarbeit zwischen Wissenschaftlern durch Verknüpfung ihrer Erkenntnisse in **netzartig verlinkten** Dokumenten über das Internet zu ermöglichen, was dem Konzept den Namen World Wide Web (dt. weltweites **Netz**) eingebracht hat [9].

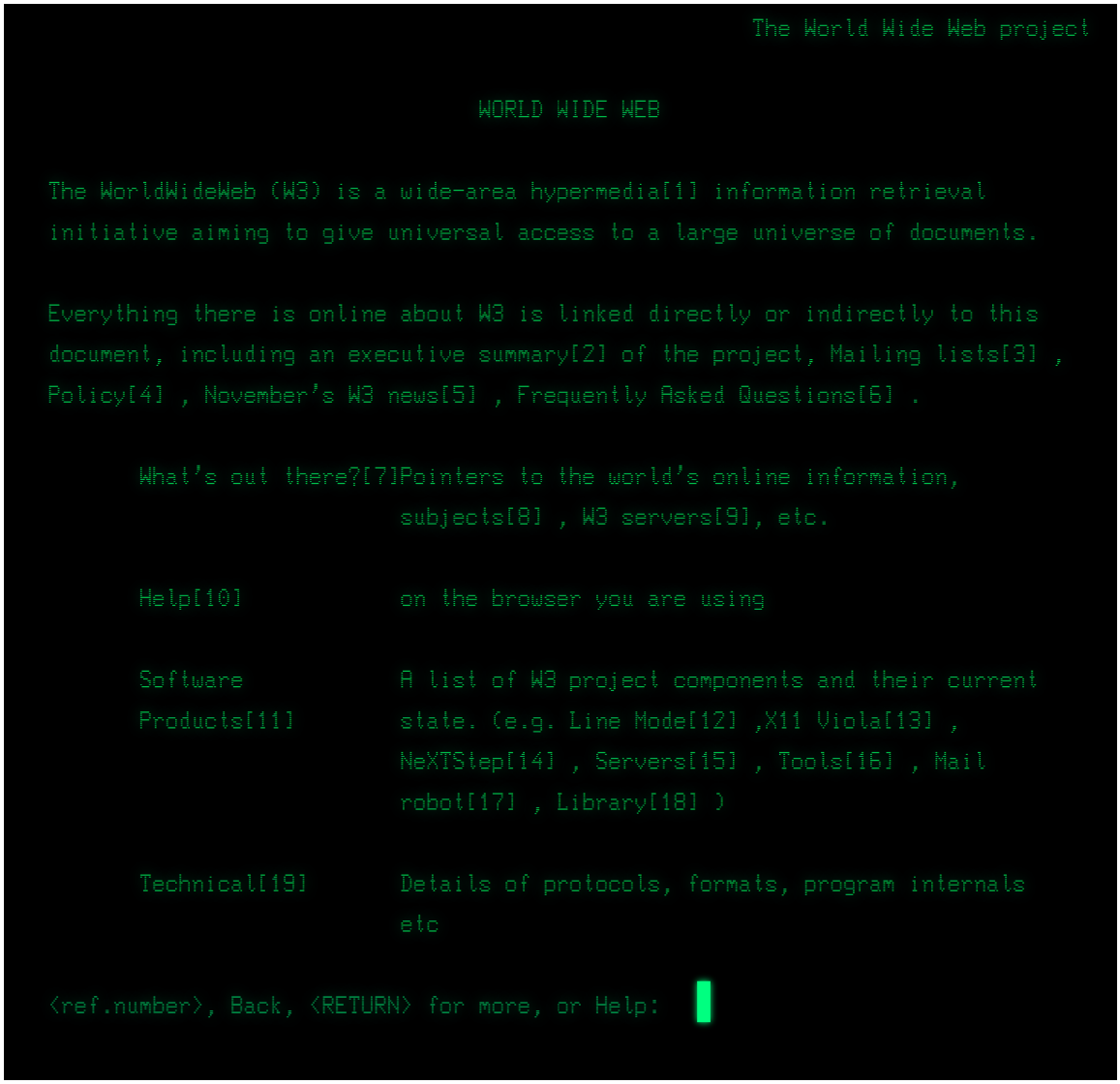

Abbildung 2.2: Die Simulation der ersten HTML-Webseite der Welt, die Berners-Lee auf seinem Rechner in CERN 1989 entwickelt hat. Diese kann von der CERN-Webseite abgerufen und per Tastatur bedient werden. (Quelle: [10])

[1] Conseil Européen pour la Recherche Nucléaire (franz.)

Zu Spezifikationen und Normen

Folgende wichtigen Normungsorganisationen werden im Rahmen dieser Arbeit erwähnt: Deutsches Institut für Normung e. V. (DIN) für deuteschlandweite Normen, International Oraganization for Standardization (ISO) für weltweite Normen, International Electrotechnical Commission (IEC) für weltweite elektrotechnische Normen. Für Web-Technologien besonders wichtig sind World Wide Web Consortium (W3C), Web Hypertext Applications Technology Working Group (WHATWG) und European Computer Manufacturers Association (ECMA).

Request For Comments (RFC) – auf Deutsch „Bitte um Stellungnahme/-Kommentare" – ist eine online Plattform zur Einreichung technischer Entwürfe zu Internet-Technologien (engl. drafts) zwecks Begutachtung durch andere Experten. Die Entwürfe werden evtl. nach mehreren Überarbeitungen als Spezifikationen zugelassen. RFC wird von Elitegruppen wie Internet Engineering Task Force (IETF) geführt.

Berners-Lee hat 1994 das W3C gegeründet, das sich der Spezifikation und Weiterentwicklung von Web-Technologien widmet. Ein Jahr später hat er die zweite Version vom HTML unter RFC 1866 offiziell veröffentlicht [11] und 2000 wurde die vierte Version unter ISO/IEC 15445 weltweit normiert [12].

2004 wurde aufgrund eines Interessenkonflikts zwischen W3C und 3 prominenten Web-Browser Herstellern (Apple, Mozilla und Opera) die Gruppe WHATWG zur Weiterentwicklung von HTML ins Leben gerufen. Nach Uneinigkeiten zwischen W3C und WHATWG, die Veröffentlichung divergierender Spezifikationen für HTML zur Folge hatte, erklärte W3C im Jahr 2019 die von WHATWG veröffentlichte HTML-Spezifikation als die einzig gültige Weiterentwicklung, die alle vorherigen Versionen ablöst [13][14][15][16]. Von WHATWG stammt die heute allgegenwärtige Version von HTML (ehem. als HTML5 bezeichnet), die sich gewissermaßen von vorherigen Versionen unterscheidet [17]. U. a. hält sich die jüngste Version nicht mehr an das Standard Generalized Markup Language (SGML), obwohl sie syntaktisch sehr ähnlich sind [18, § 13.2].

SGML

Das SGML ist eine 1986 über ISO 8879 normierte Metasprache, mit der das Extensive Markup Language (XML) und die älteren Versionen von HTML geschrieben worden sind. [19][20]

Metasprachen sind Sprachen, die zur syntaktischen und semantischen Beschreibung bzw. Interpretation anderer Sprachen verwendet werden. DUDEN Sie liegen der Übersetzung und Fehlererkennung bei Compilern und Interpretern zu Grunde.

Aufbau Code 2.1 zeigt den Aufbau eines einfachen HTML-Dokuments. Der Code beginnt mit `<!DOCTYPE html>` in der 1. Zeile. Dies ist der sog. Document Type Descriptor (DTD) und ist eigentlich kein HTML-Tag, sondern sagt dem Code-Interpreter, in welcher Sprache das Dokument geschrieben wurde (vgl. Metasprache) [19][20]. Erst in der 2. Zeile beginnt das `<html>`-Dokument und in der 12. Zeile wird es mit `</html>` beendet. Ein Schrägstrich vor dem Tag-Namen beendet die Beschreibung des gleichnamigen Elements.

Tag vs. Element

Es muss zwischen Element und Tag unterschieden werden: Ein Element ist ein abstraktes Objekt und ein Tag ist ein Symbol zur Beschreibung dieses Objektes. Z. B. das `head`-Element ist immer in einem HTML-Dokument vorhanden, selbst wenn `<head>...</head>`-Tags nicht vorhanden sind [20].

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <meta charset="UTF-8">
5          <title>Überschrift</title>
6      </head>
7      <body>
8          <header>Kopfzeile</header>
9          <p>Inhalt</p>
10         <footer>Fußzeile</footer>
11     </body>
12 </html>
```

Code 2.1: Beispiel: HTML-Code

Metadaten Der Bereich `<head>...</head>` beinhaltet Metadaten, die auf dem Dokument nicht sichtbar sind, aber für die Darstellung des Dokumentes eine Rolle spielen. Z. B. die Zeichenkodierung wird in der 4. Zeile durch das `charset`-Attribut eines `<meta>`-Tags festgelegt, damit bspw. die deutschen Umlaute richtig dargestellt werden. Die in `<title>...</title>` eingeschlossene Überschrift wird nicht auf dem Dokument selbst, sondern auf dem Browser-Fenster angezeigt.

Körper Der Bereich `<body>...</body>` umfasst der sichtbare Hauptteil des Dokumentes, der in diesem Fall aus einer Kopfzeile namens „Kopfzeile“ in `<header>...</header>`, einem Paragraph „Inhalt“ in `<p>...</p>` und einer Fußzeile namens „Fußzeile“ in `<footer>...</footer>` besteht.

Darstellung Der Code wird in einer Textdatei mit der Endung `.html` abgespeichert und mit einem Internet-Browser geöffnet. Abbildung 2.3 zeigt die Darstellung der HTML-Seite im Browser.

Abbildung 2.3: Die Darstellung der HTML-Seite von Code 2.1 in einem Internet-Browser

2.3 Cascading Style Sheets

Motivation Wurde man die Struktur des Dokumentes in Abbildung 2.3 auch dann erkennen, wenn nicht jeder Teil mit dem eigenen Namen versehen worden wäre? Code 2.1 würde in dem Fall aufgrund der Hierarchie und der Markierungen mehr Informationen über die Dokumentstruktur liefern als die Abbildung. Wie kann man den Stil des Dokumentes anpassen?

Geschichte Die Frage beschäftigte auch den Håkon Wium Lie, als er 1994 bei CERN gearbeitet hat. Der Erfinder von Web – Berners-Lee – hatte zwar die Stilanpassung von Web-Dokumenten im Sinn, hat jedoch die Wahl über den Stil den Webbrowser-Herstellern überlassen. Es wurden mehrere Spezifikationen für die Stilanpassung von Webseiten vorgeschlagen. Die Spezifikation von Wium Lie – Cascading Style Sheets (CSS) – hatte entscheidende Vorteile: Er war der Überzeugung, dass die Anwenders und Urhebers Wünsche sowie die Eigenschaften des Browsers und des Darstellungsmediums hierarchisch zusammenfließen bzw. „kaskadiert“ werden sollen. Zudem hat sich die einfache Syntax, die damals für die Kritik sorgte, die CSS würden der wachsenden Komplexität der Aufgabe nicht gerecht werden, in der Praxis als eine große Stärke erwiesen. [21].

Konzept Die CSS sind austauschbare Formatierungsschablonen und beschreiben nicht nur die statische Formatierung von Dokumenten und Elementen, sondern auch die dynamische Formatierung. Beispiel hierfür wäre die Darstellung auf Rechnern vs. mobilen Endgeräten (vgl. Abschnitt 2.8 Responsive Web Design). Da die CSS i. d. R. nicht den Inhalt des Dokumentes beschreiben, zählen sie zu Metadaten und werden im Kopf des Dokuments entweder unmittelbar definiert oder auch als eine separate Datei eingebunden. Die Trennung von Formatierung und Inhalt hat auch den Vorteil, dass die Formatierung eines Web-Dokumentes ohne Eingriff in das Dokument angepasst werden kann, insofern die CSS als externe Datei eingebunden wurde.

Einbindung Nun wird anhand von Code 2.2 die Funktionsweise der CSS betrachtet. Zeile 6 zeigt die Einbindung von CSS als externe Datei mit `<link rel="stylesheet"/>`, wobei das `href`-Attribut den Pfad zur externen Datei angibt (Web-Adresse oder lokalen Pfad). Alternativ kann man auch – wie in diesem Beispiel – CSS direkt innerhalb eines `<style></style>`-Blocks beschreiben.

MIME-Type

Das Attribut `type="text/css"` in Code 2.2 Zeile 6 u. 7 zeigt die Angabe eines MIME-Type[a], der den Browser verständigt, um welchen Datentyp es sich bei der verlinkten Datei handelt. Besonders bei `<link rel="stylesheet"/>` ist diese Angabe wichtig, falls der Server den Datentyp nicht richtig angibt. [22][23]

[a] Multipurpose Internet Mail Extensions

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <meta charset="UTF-8">
5          <title>weise Sprüche</title>
6          <link rel="stylesheet" type="text/css" href="stl.css"/>
7          <style type="text/css">
8              .Kopfzeile {
9                  font-weight: bold;
10             }
11             .Inhalt::before {
12                 content:    ",,";
13             }
14             .Inhalt {
15                 font-style: italic;
16                 color:      green;
17             }
18             .Inhalt::after {
19                 content:    "‘‘";
20             }
21             .Fußzeile::before {
22                 content:    "- ";
23             }
24         </style>
25     </head>
26     <body>
27         <header class="Kopfzeile">Zitat</header>
28         <p class="Inhalt">Alles, was einen Anfang hat, hat auch
               ein Ende Neo.</p>
29         <footer class="Fußzeile">das Orakel</footer>
30     </body>
31 </html>
```

Code 2.2: Beispiel: HTML-Code mit CSS

Aufbau Der Definition von CSS liegen die sog. CSS-Selektoren zugrunde. Sie sind Anweisungen zum Heraussuchen einer Menge von zuformatierenden Elementen und können die Auswahl auf ein einziges Element eingrenzen. Sogar komplexe und riesige Dokumente lassen sich durch CSS-Selektoren leicht formatieren, insofern das Dokument mit Bedacht strukturiert wurde. Die Übersicht der finalen Spezifikation der Selektoren zu CSS3 ist unter dem folgenden Verweis abrufbar [24]. Jede Definition beginnt mit einem Selektor, gefolgt von einem Block in geschweiften Klammern, in dem die Stilattribute der durch den Selektor herausgesuchten Elemente definiert werden. Die Wertzuweisung zum jeweiligen Attribut erfolgt mit einem »:« und die Zuweisungen werden durch »;« voneinander getrennt.

Klassen In Code 2.2 werden drei Stilklassen beschrieben: Kopfzeile, Inhalt und Fußzeile. Man kann zwar die Elemente `<header>`, `<p>` und `<footer>` direkt manipulieren, aber dadurch werden alle weiteren Instanzen der Elemente auch den Stil übernehmen, was vielleicht nicht beabsichtigt ist. Die Klassenbildung ermöglicht eine weitere Stufe von Trennung, sodass Stile gezielt von bestimmten Instanzen von Elementen – und nicht allen Elementen desgleichen Typs – übernommen werden. Die Klassen werden durch `class`-Attribut für das jeweilige Element übernommen (Zeilen 30–32).

Definition Die Definition der Klassen erfolgt durch den Klassenselektor (der Klassenname mit einem vorangestellten Punkt). Zeilen 8–10 formatieren die Kopfzeile fettgedruckt. Zeilen 11–12 bzw. 18–20 weisen an, dass vor (`::before`) bzw. nach (`::after`) dem Inhalt Anführungszeichen hinzugefügt werden sollen. Zeilen 14–17 formatieren den Inhalt selbst kursiv und legen Grün als Schriftfarbe fest. Zeilen 21–23 stellen der Fußzeile einen Strich voran. Der Inhalt des `<style></style>`-Blocks kann auch als eine `.css`-Datei abgespeichert und wie in Zeile 6 eingebunden werden. Das Ergebnis ist in Abbildung 2.4 dargestellt. Das Beispiel zeigt im Vergleich zum Funktionsumfang der CSS nur die Spitze des Eisbergs. Formatierung für unterschiedliche Ausgabemedien, wie z.B. Darstellung bei einem PDF-Auszug[2] oder anspruchsvolle Formatierung mit Graphiken und Animationen sind u. a. mit der einfachen, doch mächtigen Syntax der CSS möglich.

Abbildung 2.4: Die Darstellung der HTML-Seite mit CSS-Formatierung von Code 2.2 in einem Internet-Browser

[2]Portable Document Format

Kaskadierung Auch mehrere, teilweise widersprüchliche CSS können gleichzeitig eingebunden werden. Die mehrfach definierten Stile für dasselbe Element werden nach Hierarchie und Wichtigkeit in mehreren Stufen ausgewertet und kaskadiert, woraus eine Menge aus Stilattributen, die sich gegenseitig nicht ausschließen, zusammengesetzt und für das Element übernommen wird. Die Kaskadierung macht die CSS sehr robust gegen Fehler, da Konflikte durch interne Priorisierung aufgelöst werden. Bei Bedarf kann man die Priorisierung teilweise umgehen, um einen wichtigen Stil durchzusetzen, indem man einem Stilattribut `!important` nachstellt [25].

2.4 JavaScript

Motivation Freiheit in Formatierung von Web-Dokumenten war erst der Anfang. Rechner hatten viel mehr anzubieten als nur passive Lesegeräte und Netscape® – eine 1994 gegründete Firma, die noch im selben Jahr mit der Vorstellung vom Netscape Navigator® Web-Browser einen Durchbruch erzielte – hat die mangelnde Dynamik von Webseiten erkannt und wollte das volle Potenzial der Web-Clients entfalten, damit Anwender bspw. bei Überprüfung von Benutzereingaben, wozu der Anwenderrechner imstande ist, nicht auf den Server angewiesen ist [26, Kap. 4].

Geschichte Netscape® engagierert 1995 den Programmierer Brendan Eich und die Firma Sun Microsystems® – Entwickler der Programmiersprache Java, die später in Oracle® eingegliedert wurde – um dynamisches Verhalten in Web-Dokumenten zu ermöglichen. Der Lösungsansatz war, sowohl Java® in den Netscape Navigator® zu integrieren als auch eine leichtgewichtige Skriptsprache zu entwickeln, die unmittelbar als Teil des HTML-Quellcodes geschrieben werden kann. Während Java® als eine vollumfängliche „Komponentsprache“ für hoch-bezahlte Programmierer gedacht war, die Komponenten für Web entwickeln, sollte die zu entwickelnde „Klebesprache“ Amateuren und Webdesignern das Zusammengekleben der Komponenten und Automatisierungsaufgaben erleichtern. So wurde die Skriptsprache JavaScript® von Brendan Eich erfunden, die Manipulation von Web-Dokumenten zur Laufzeit ermöglicht hat [26, Kap. 4][27]. JavaScript® wurde zwei Jahre später von ECMA unter der Spezifikation ECMA-262 als *ECMAScript* spezifiziert [28, S. vii].

2.5 Document Object Model

Motivation Um das Dokument zur Laufzeit zu manipulieren, ist eine Abstraktionsschicht – das sog. Document Object Model (DOM) – nötig, die das abstrakte Dokument als eine durchlaufbare, hierarchische Baumstruktur bereitstellt, mit der Programme interagieren können.

Geschichte Gegen das Ende des 20. Jahrhunderts war das Web nach seinem Durchbruch noch sehr entwicklungsbedürftig. Die heute allgegenwärtigen Web-Technologien und -Normen befanden sich noch in einer umkämpften Entwicklung. Als Netscape®

die erste Version von JavaScript® entwickelt hat, war die DOM-Implementierung von Netscape® sehr problembehaftet, was u. a. den Hauptkonkurent Microsoft® dazu motiviert hat, eine eigene Implementierung daraus abzuleiten [29, Abschn. 19.2][30]. Da zu diesem Zeitpunkt der Markt hauptsächlich von Netscape Navigator® und Microsoft Internet Explorer® dominiert war, die mit unvereinbaren Eigenimplementierungen von JavaScript®[3] und DOM geliefert wurden, war eine einheitliche Skript-Programmierung nicht einfach möglich. Diese Phase wird in der Literatur als der erste Browserkrieg (engl. *browser wars*) bezeichnet [31][32]. Code 2.3 überprüft, welche DOM-Implementierung in der Laufumgebung vorhanden ist und programmiert entsprechend.

Spezifikation Erst im Oktober 1997 veröffentlicht das W3C den ersten Entwurf vom DOM Stufe 1 und beschreibt es als ein Standardmodell der Zusammensetzung von Objekten in einem XML- oder HTML-Dokument, das als eine **plattform- und sprachunabhängige** Schnittstelle Programmen dynamischen Zugriff auf Dokumente zur Manipulation deren Struktur und Stil ermöglicht. [33]. Der Begriff DOM bezeichnet heute diese einheitliche Spezifikation. Die Weiterentwicklung vom DOM zusammen mit HTML wurde 2019 vom W3C an die WHATWG übergeben [14][15][16].

DOM Stufe 0

Das DOM Stufe 0 war die absolut erste Version, die mit Netscape Navigator® v2 eingeführt wurde und funktioniert – bei Browsern, die es noch unterstützen – konsistent, wurde aber nie offiziell spezifiziert [29, Abschn. 19.3][30].

```
if (document.all) {
    // Code für Microsoft-DOM (IE v4+)
} else if (document.layers) {
    // Code für Netscape-DOM (NS v4+)
} else if (document.getElementById) {
    // Code für W3C-DOM Level 1 (IE5+ und NS6+)
}
```

Code 2.3: Historischer JS-Code zur Erkennung der vorliegenden DOM-Implementierung zwecks Plattformkompatibilität v. a. zwischen Internet Explorer (IE) und Netscape Navigator (NS) [29, Abschn. 19.3, bearbeitet]

Aufbau Das DOM entspricht der hierarchischen, baumartigen Struktur – in diesem Kontext – eines HTML-Dokuments. Abbildung 2.5 zeigt das DOM von Code 2.2. `HTML` bezeichnet das Dokument, die Wurzel des Baums. `HEAD` und `BODY` sind die Zweige und wachsen bis hin zu den Endknoten bspw. `P` oder `FOOTER`. Sie haben wiederum Attribute wie `class` und evtl. einen Inhalt (`#text`). Auch im DOM kann man heute mithilfe von CSS-Selektoren Elemente heraussuchen und den Strukturbaum durchlaufen. Ist ein Element geortet, hat man Lese- und Schreibzugriff auf seine Attribute. Die Verbindung zwischen Knoten spielt im DOM eine wesentliche Rolle. Wenn ein Knoten entfernt wird, werden auch seine untergeordneten Knoten entfernt, da es keinen Pfad mehr gibt, der zu diesen führt.

[3] Die leicht abgeänderte JavaScript® von Microsoft hieß JScript

```
DOCTYPE: html
HTML
  HEAD
    META charset="UTF-8"
    TITLE
      #text: weise Sprüche
    LINK rel="stylesheet" type="text/css" href="style.css"
    STYLE type="text/css"
      #text: .Kopfzeile {font-weight: bold;} .Kopfzeile {font-weight: bold;} .Inhalt::before {content: „;} .Inhalt {font-style:
      italic; color: green;} .Inhalt::after {content: "“";} .Fußzeile::before {content: "- ";}
  BODY
    HEADER class="Kopfzeile"
      #text: Zitat
    P class="Inhalt"
      #text: Alles, was einen Anfang hat, hat auch ein Ende Neo.
    FOOTER class="Fußzeile"
      #text: das Orakel
```

Abbildung 2.5: Die DOM-Hierarchie von Code 2.2 (Quelle: [34], eigener Code)

Nun wird versucht, die HTML-Seite von Abbildung 2.4 mit JavaScript®-Code 2.4 zu manipulieren. Zeile 1 definiert eine Funktion names `citeSmith()`. Die DOM-Methode `document.getElementsByClassName()` liefert ein Array aus allen Elementen mit dem angegebenen Klassennamen zurück. Das es in unserem Beispiel jeweils nur eine Instanz von jeder Klasse existiert (vgl. Code 2.2), wird in Zeilen 3 u. 4 jeweils der Verweis auf das einzige (erste) Element durch den Indexoperator `[0]` ausgewählt und in Variablen `Zitat` bzw. `Autor` gespeichert. In Zeile 6 wird das Attribut `innerHTML`, der den Inhalt von `Zitat` beinhaltet mit einem neuen Zitat überschrieben. In Zeile 7 wird das `color`-Attribut vom Stil des Zitats auf rot gesetzt. Der lokal gesetzte Wert hat Vorrang über die von CSS vererbte Farbe. Als Letztes wird in Zeile 8 der Name vom Autor aktualisiert.

innerHTML vs. innerText

Der Unterschied zwischen `innerHTML` und `innerText` besteht darin, dass `innerText` den zugewiesenen String als reinen Text annimmt. Dagegen interpretiert `innerHTML` den String als HTML-Code und die `<br>`-Tags werden in Zeilenumbrüche umgewandelt. Dies stellt bei Darstellung von Texten aus nicht vertrauenswürdigen Quellen eine Angriffsfläche dar, weil in `<script></script>`-Block eingeschlossener Text clientseitig als Skript ausgeführt werden kann. Diese Angriffsart wird als Cross-Site Scripting (XSS) bezeichnet.

```
1 function citeSmith()
2 {
3     Zitat = document.getElementsByClassName("Inhalt")[0];
4     Autor = document.getElementsByClassName("Fußzeile")[0];
5
6     Zitat.innerHTML = "Das Binäre, das das Wesen der Dinge
          formt:<br>Nullen und Einsen, Licht und Dunkelheit,<br>
          die Wahl haben oder nicht, Anderson und Smith!"
7     Zitat.style.color = "red";
8     Autor.innerText = "Agent Smith";
9 }
```

Code 2.4: Beispiel JS-Code zur Änderung von HTML-Code 2.2 während der Laufzeit

Einbindung Die Funktion Kann wie CSS innerhalb von `<head></head>` entweder direkt in HTML-Code in einem `<script></script>`-Block definiert, oder auch durch `<script src="script.js"type="text/javascript"/>` als externe Datei mit `.js`-Endung eingebunden und zur Laufzeit bei bestimmten Ereignissen – z. B. wenn die Seite vollständig geladen wurde oder per Knopfdruck – aufgerufen werden. Abbildung 2.6 zeigt die manipulierte HTML-Seite nach dem Aufruf der Funktion.

Abbildung 2.6: Die HTML-Seite von Abbildung 2.4, die zur Laufzeit mit dem JavaScript®-Code 2.4 manipuliert wurde.

2.6 JavaScript Object Notation

JavaScript® Object Notation (JSON) ist die Syntax zur textuellen Darstellung von JavaScript®-Datenobjekten, die aufgrund der einfachen und kompakten Grammatik, die zukünftige Änderungen der Spezifikation sehr unwahrscheinlich macht, als eine robuste Syntax für einen einheitlichen Datenaustausch zwischen allen Programmiersprachen dienen kann. Die JSON hat allerdings keine Kenntnis über die Semantik der Informationen [35, S. iii].

Eine semantische Beschreibung mit der JSON-Syntax ist immerhin bis zu einem gewissen Niveau möglich, z. B. durch das von IETF spezifizierte JSON-Schema, die Definition von Randbedingungen ermöglicht, gegen die ein JSON-Objekt überprüft werden kann. Man kann also die erwartete Struktur eines JSON-Objekts mit einem anderen JSON-Objekt beschreiben [36].

2.7 Representational State Transfer

Geschichte Als Roy Thomas Fielding zwischen 1994 – 1995 bei W3C und IETF an der ersten Spezifikation des Hypertext Transfer Protocol (HTTP) – das Protokoll zur Übertragung von HTML-Dokumenten über das Internet – mitgewirkt hat, entwickelte er Representational State Transfer (REST) als „ein Mittel zur Kommunikation von Web-Konzepten“, also ein model, das die gewünschte Funktionsweise des Webs beschreibt [37, Abschn. 6.1]. Aus dem Grund sind REST und HTTP eng verwandt, obwohl Fielding

REST als eine plattformunabhängige Architektur formuliert hat. Fielding hat 2000 REST im Rahmen seiner Dissertation veröffentlicht [37].

Konzept REST (dt. gegenständlicher Zustandsübertragung) ist ein Architekturstil mit Betonung auf eine einheitliche, zustandslose Schnittstelle. Bei der Zustandslosigkeit geht es darum, dass der Client bei der Kommunikation mit dem Server davon ausgeht, dass der Server kein Gedächtnis für den bisherigen Kommunikationsverlauf mit dem Client hat, sodass er den Kontext der abgefragten Daten (:= state) bei jeder Abfrage so konkret angibt (:= representional), dass die zurückzuliefernden Daten eindeutig zu identifizieren sind. REST bietet folgende Vorteile: verbesserte Sichtbarkeit bei der Überwachung der Datenverkehr zwischen Client und Server, da der Kontext der übertragenen Daten in jeder Abfrage ohne Bezug auf vorherige Kommunikation nachvollziehbar ist, verbesserte Zuverlässigkeit, da Fehler bei der Übertragung einer Nachricht die zukünftige Kommunikation nicht beeinträchtigt und verbesserte Skalierbarkeit aufgrund der Resourcenfreigabe, die durch fehlende serverseitige Zustandsspeicherung ermöglicht wird [37, Abschn. 5.1.3].

2.8 Responsive Web Design

Der Begriff *Responsive Web Design* (dt. reaktionsfähiges Webdesign) wurde 2010 von Ethan Marcotte etabliert. Er geht in einem geleichnamigen Artikel auf die wachsende Anzahl der web-fähigen Geräte – wie bspw. Smartphones und Spielkonsolen – und im weiteren Sinne die Vielfältigkeit der Ausgabemedien und deren Auflösung bzw. Seitenverhältnisse ein. Er demonstriert, wie man bei der Gestaltung von Webseiten mithilfe von Mediumabfragen der CSS (engl. media queries) auf das jeweilige Ausgabemedium geeignet reagieren kann, um die optimale Darstellung der Webseite auf allen web-fähigen Geräten zu gewährleisten [38][39].

2.9 Responsive Grid Layout

Ein *Responsive Grid Layout* (dt. reaktionsfähige Rasteranordnung) ist eine Umsetzung des reaktionsfähigen Webdesigns. Der Begriff wird mehrdeutig eingesetzt, da es unterschiedliche plattformabhängige Interpretationen gibt, wie bspw. von Google® definiertes *Responsive Layout Grid* auf der MaterialDesign-Webseite, die Richtlinien fürs Design von Android-Apps festlegt [40]. Für diese Arbeit relevant ist das *CSS Grid Layout*, das erst 2011 vom W3C eingeführt wurde [41]. Beim *CSS Grid Layout* wird über ein Webelement – z. B. den Grundriss des Ausgabemediums – ein Raster aus orthogonalen (waagerechten und senkrechten) Linien aufgespannt, das das Webelement auf abstrakte Rasterzellen verteilt, über die sich untergeordnete Webelemente ausstrecken können. Statt die Dimensionierung der Elemente anzupassen, werden die Webelemente an Rasterregionen – also rechteckige Gruppen zusammenhängender Rasterzellen – angebunden, die sich bei Größenänderung des Rasters proportional anpassen. [42, §3]. Die hierarchische Entkopplung von der reaktionsfähigen Anordnung und Webelementen bietet noch mehr Flexibilität: Das Raster passt sich bei Änderungen

an und die Elemente folgen den Stil der ihnen jeweils zugewiesenen Rasterregion. Eine Verschachtelung von Rastern ermöglicht komplexes Verhalten für anspruchsvolle Benutzerschnittstellen. Abbildung 2.7 zeigt ein abstraktes *Grid Layout* zur Aufteilung einer Webseite auf Regionen.

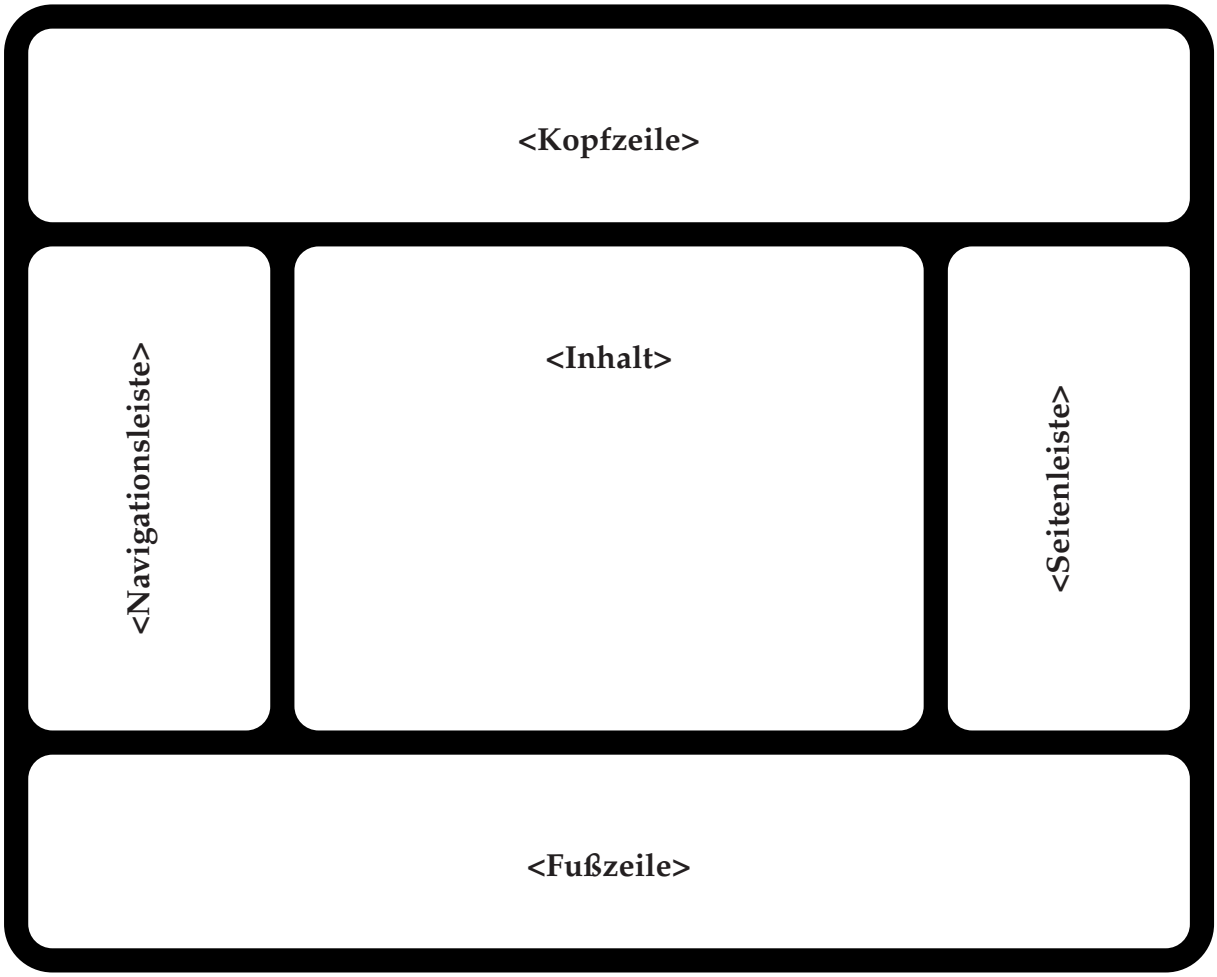

Abbildung 2.7: Beispiel CSS Grid Layout zur proportionalen Aufteilung einer Webseite auf Kopfzeile, Navigationsleiste, Inhalt, Seitenleiste und Fußzeile (Quelle: [42, §4, bearbeitet]).

Kapitel 3

Methodik

In diesem Kapitel geht es um die Vorgehensweise der Auftragsdurchführung. Es wird davon ausgegangen, dass sich der Leser schon mit den Grundlagen des Projektmanagements und im weiteren Sinne der Anforderungsanalyse auskennt. Hierfür wird das Buch **Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements-Engineering** von **Helmut Balzert** [43, S. 433–587] als Nachschlagewerk verwendet.

3.1 Analyse

Zu Beginn sollen die Anforderungen an das zu entwerfende Dashboard ermittelt werden. Die Voraussetzung dafür ist, dass die Personen, die an das System überhaupt eine Anforderung stellen dürfen, identifiziert werden [43, S. 455 u. 504]. Als nächstes sind die technischen Rahmenbedingungen zu bestimmen, insbesondere die Umgebungen, in denen das Dashboard entwickelt bzw. ausgeführt wird [43, S. 460]. Diese dürfen als Anforderungen an die Architektur des Dashboards betrachtet werden.

3.1.1 Stakeholder-Analyse

Bei der Stakeholder-Analyse geht es um die Erkennung von Instanzen (Personen und Organen), deren Ansprüche bei der Projektplanung berücksichtigt werden sollen. Sie müssen nicht notwendigerweise die Nutzer des Endproduktes sein, sondern auch jeder, der mit dem Produkt in Berührung kommen wird bzw. einen Anspruch auf die Qualität des Endproduktes hat, kann ein Stakeholder sein. Da es sich um kein öffentliches Produkt handelt, sind die Stakeholder jedoch ausschließlich interne Instanzen. Wir listen die Anspruchsberechtigten hierarchisch auf (Abbildung 3.1). Es gibt 4 Instanzarten, die als Stakeholder betrachtet werden: Unternehmen, Werk, Abteilung und Mitarbeiter. Wir werfen zunächst einen Blick auf die Ansprüche der Stakeholder auf der jeweiligen Ebene:

Auf Unternehmensebene: Das Produkt muss in die IT-Landschaft[1] der Siemens AG passen (siehe Unterabschnitt 3.1.3). Außerdem gibt es interne Randbedingungen für das allgemeine Aussehen von Web-Applikationen, die im Rahmen dieser Arbeit jedoch nicht behandelt werden (vgl. Abschnitt 1.5).

Auf Werksebene: Das Dashboard wird nicht nur von anderen deutschsprachigen Werken, wie z.B. dem Elektronikwerk Fürth (EWF) verwendet werden, sondern auch von internationalen Werken wie Siemens Elektronikwerk Chengdu (SEWC) in China und dem neu gegründeten Elektronikwerk Singapur (EWS). Daher ist es sinnvoll, das Dashboard zunächst auf Englisch anzubieten oder Mehrsprachlichkeit bei der Konzeption zu berücksichtigen.

Auf Abteilungsebene: Das Dashboard wird hauptsächlich von Qualitätsexperten, aber auch von anderen Abteilungen in Zusammenarbeit mit Qualitätsexperten verwendet. Im Grunde soll das Dashboard von allen verwendbar sein, damit der Zusammenarbeit nichts im Wege steht.

Auf Mitararbeiterebene Auf Mitarbeiterebene gibt es eine Unterscheidung, nämlich die Art der Interaktion mit dem Dashboard:

- Nutzer: Die Anforderungen müssen zunächst erfasst werden (Siehe Unterabschnitt 3.1.2).
- Entwickler: Ein Entwickler wird für die Wartung und Weiterentwicklung des Dashboards zuständig sein und hat somit andere Ansprüche, wie z.B. Code-Qualität (:= kompakter, gut strukturierter und sauber dokumentierter Code).

3.1.2 Anforderungsanalyse

In diesem Abschnitt werden die funktionalen Anforderungen an das Dashboard ermittelt. Unter funktionalen Anforderungen versteht man konkrete Dienste, die man vom Produkt erwartet. Diese sind durch Abnahme- bzw. Akzeptanzkriterien so lückenlos zu spezifizieren, dass die Akzeptanzkriterien nach der Umsetzung des Produktes zur Formulierung von Testfällen zwecks Beurteilung der Erfüllung der Anforderungen dienen können. [43, vgl. S. 456 u. 471].

Das Dashboard wird im Rahmen eines agilen Projektes entwickelt. Daher werden die Anforderungen in Form von User Stories erfasst. Balzert zeigt 2 Beispiele für agile Satzschablonen aus der Literatur [43, vgl. S. 497–501]. Hier wird eine formalere, einseitige Schablone für die User Stories in LaTeX formuliert, die den Stakeholder, den Wunsch und den Nutzen trennt, um Lesbarkeit zu erleichtern sowie Akzeptanzkriterien und weitere Kommentare auf derselben Seite darstellt.

Immerhin sind User Stories oft aufgrund der fachfremden Formulierung keine gut-formulierte Anforderungen und müssen im Nachhinein weiter verhandelt, validiert

[1] Information Technology

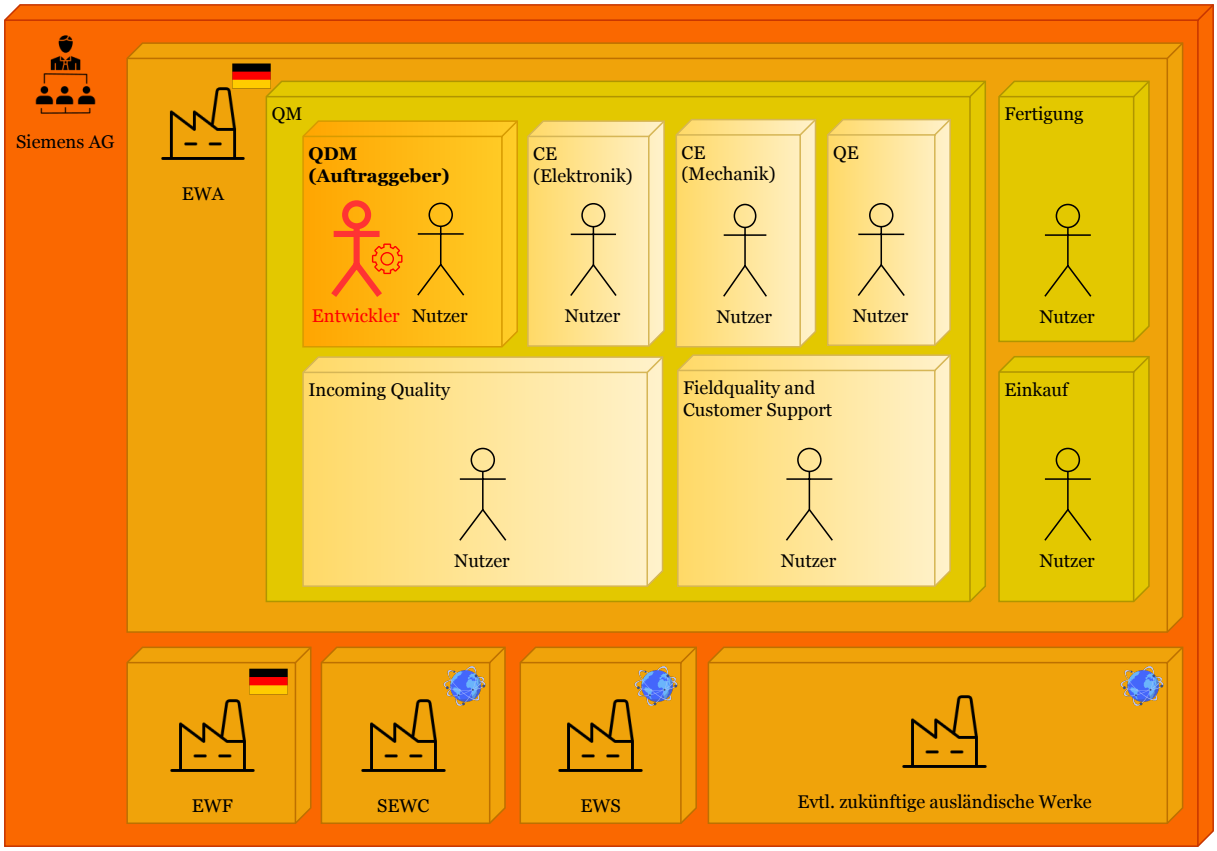

Abbildung 3.1: Hierarchische Übersicht der Stakeholder: das Unternehmen, die Werke, die Abteilungen und Mitarbeiter. Einfachheitshalber wurde auf die interne Hierarchie weiterer Werke verzichtet (Quelle: eigene Darstellung).

und konkretisiert werden [43, vgl. S. 475 u. 500]. Um den unnötigen Aufwand zu sparen, werden die Anwender-Interviews protokolliert, in Absprache mit dem Auftraggeber ausgewertet und zu User Stories verarbeitet [43, vgl. S. 507–510]. Drei Key-User aus den drei Abteilungen QE, CE-Elektronik und CE-Mechanik werden interviewt, um die Anwender und ihre Bedürfnisse besser kennenzulernen und ihren Arbeitsalltag bei der Konzeption zu berücksichtigen. Auch der Auftraggeber hat im Vorfeld Im Rahmen des Scout-NextGen-Projekts Anforderungen durch umfängliche Interviews abgenommen, wodurch die Idee des Dashboards überhaupt zustande gekommen ist. Zusammen dienen die Interviews als die Grundlage der Anforderungsspezifikation.

Die erfassten User Stories sind unter dem gleichnamigen Kapitel oder auch unter dem Tabellenverzeichnis zu finden. Die User Stories bis User Story #008 wurden vom Auftraggeber als Vertreter der von ihm befragten Anwender abgenommen und dienen als Grundanforderungen des Dashboard-Konzepts. Ab User Story #009 sind die Anforderungen im Laufe des Entwurfsprozesses zustande gekommen, validiert bzw. weiter spezifiziert worden. Die User Stories werden nach Wichtigkeit priorisiert. Eine Übersicht der Abhängigkeiten zwischen den User Stories zeigt die Abbildung 6.1. Die Übersicht soll bei der Konzeption und Implementierung in Betracht gezogen werden, damit die entworfenen Module richtig zusammenarbeiten und in das Gesamtkonzept passen.

3.1.3 Ist-Analyse

Nachdem die Anforderungen spezifiziert worden sind, wird die vorliegende Systemlandschaft ermittelt. Diese ist entscheidend für die Entwicklung, Umsetzung und Integration des Dashboards in das Gesamtsystem.

Zur Datenhaltung kommt die Oracle-Datenbank® in Einsatz, die mit Oracle-APEX®[2] ausgestattet ist. APEX® ist eine in der Oracle-Datenbank® integrierte, webbasierte graphische Umgebung zur aufwandarmen Erstellung von Web-Anwendungen, die eine unkomplizierte Anbindung an die Oracle-Datenbank® durch PL/SQL-Abfragen[3] ermöglichen.

PL/SQL

PL/SQL ist eine prozedurale Sprache, die von Oracle® speziell zur Einbettung von SQL-Anweisungen in serverseitige Datenbank-Prozeduren entwickelt wurde.
ORACLE

I. d. R. sollten die SQL[4]-Abfragen serverseitig als eine PL/SQL-Prozedur (ähnlich wie eine CGI[5]) bereitgestellt und clientseitig durch AJAX[6] aufgerufen werden. Dieser Aufwand wird von APEX® automatisiert, was Oracle-APEX® zu einer leistungsstarken Wahl zur Bereitstellung von auf einer Oracle-Datenbank® basierten Diensten macht.

APEX® beschleunigt zwar die Erstellung von Webseiten mit dynamischen Inhalten dank diversen Vorlagen und Web-Elementen, wirkt jedoch bei Erstellung von Webseiten mit dynamischer Struktur eher einschränkend. Es besteht viel Eingriffsbedarf durch JavaScript®, CSS und evtl. diverse Bibliotheken. Die Webseite wird also eher in APEX® angesiedelt sein als durch sie entwickelt.

APEX® bietet eine Vielzahl an Diagrammvorlagen zur Visualisierung von Daten. Scout verwendet allerdings die HighCharts-Bibliothek zur Visualisierung von Qualitätsdaten. HighCharts ist eine Bibliothek für fortgeschrittene Datenvisualisierung, die Web-Entwicklern ein hohes Maß an Konfiguration von Diagrammen ermöglicht. HighCharts ist auf JavaScript® basiert und somit in allen Webumgebungen integrierbar. Die Diagramme werden als SVG-Vektorgraphiken[7] generiert und eigenen sich gut für dynamische Dimensionierung.

3.1.4 Stand der Technik

Oracle-APEX® ermöglicht schnelle Implementireung von Dashboards. Es gibt allerdings 2 Voraussetzungen: Man muss zum Zugriff auf die Entwicklungsumgebung

[2] Application Express®
[3] Procedural Language / Structured Query Language
[4] Structured Query Language
[5] Common Gateway Interface
[6] Asynchronous JavaScript And XML
[7] Scalable Vector Graphics

und die Datenbank berechtigt sein und grundlegende Fachkenntnisse im Bereich Webentwicklung und Datenbanken mitbringen. Außerdem ist APEX® keine WYSIWYG-Umgebung[8], d. h. die Anpassung und die Bedienung der Webseite erfolgen in 2 getrennten Umgebungen bzw. Ansichten. Das macht Oracle-APEX® zu keinem passenden Kandidat für Dashboards, die von Benutzer erstellt und verwaltet werden sollen.

WYSIWYG

What You See Is What You Get (dt. „Was du siehst, ist das, was du bekommst“) ist ein Begriff zur Beschreibung von Bearbeitungsumgebungen, in denen die visuelle Darstellung des Inhalts unmittelbar nach jedem Bearbeitungsschritt dem eigentlichen, resultierenden Ausgangszustand entspricht. D. h., es besteht keinen Schritt (z. B. Compilier-Vorgang) zwischen der Bearbeitung und der Darstellung. Beispiel hierfür sind interaktive Webseiten.

Zunächst wird im Internet nach evtl. vorhandenen Lösungen gesucht. Eine Recherche ergibt eine Vielzahl an Dashboard-Vorlagen oder auch Webentwicklern bzw. Firmen, die Entwicklung eines maßgeschneiderten Dashboards anbieten. Sie passen aber auch nicht zur dynamischen Natur des Konzepts. Es ist allerdings in diesem Stadium noch schwer zu beurteilen, was die passende Lösung anbietet. Zunächst muss das Konzept mindestens grobtechnisch herauskristalisiert werden, damit die Umsetzungsanforderungen feststehen. Erst dann müssen der Funktionsumfang und die Grenzen der jeweiligen Plattform in Kenntnis gebracht werden, damit Beurteilung über die Eignung des Lösungsansatzes überhaupt möglich ist.

3.2 Konzepterstellung

In diesem Abschnitt werden die Anforderungen in Zusammenarbeit mit dem Betreuer ausgewertet und validiert, wodurch Das Konzept des Dashboards Gestalt findet. Begonnen wird mit der Suche nach Stereotypen, also Begriffen, die sich wörtlich oder auch sinnlich öfter in den Anforderungen wiederfinden. Sie dienen als Grundbausteine oder Komponenten des Dashboards. Als Nächstes wird das Zusammenspiel der Komponente festgestellt, wodurch das Grobkonzept spezifiziert wird.

Dashboard (dt. Tafel) Das Dashboard stellt die Weichen für das Konzept. Dashboard ist eine Fläche, an die beliebige Objekte (:= Elemente) angebracht werden können, wie Merkzettel auf einer Tafel (Abbildung 3.2). Der nächste Schritt besteht in der Spezifizierung der Elemente: Was darf alles auf der Tafel stehen?

[8] What You See Is What You Get

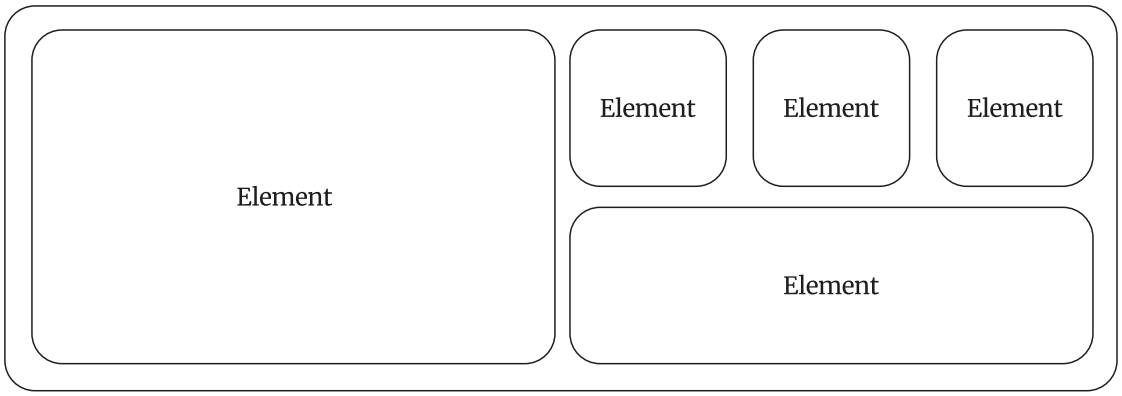

Abbildung 3.2: Wireframe eines Dashboards mit beliebigen Elementen (Quelle: eigene Darstellung).

Frame (dt. Rahmen) Anwender brauchen Auswertungen, die i. d. R. in Form eines Diagramms visualisiert werden, Tabellen und sog. Key Performance Indicators (KPIs), die im Grunde Kennzahlen sind (vgl. User Story #003). Sie sind alle dynamische Inhalte, die in quasi Echtzeit von *Scout-NextGen* generiert werden. Die Quelle des Inhalts ist also extern. Warum extern? Obwohl das Dashboard im *Scout-NextGen* integriert sein wird, werden in der Zukunft möglicherweise Elemente aus anderen internen Plattformen eingebunden, die für das Dashboard bereitgestellt werden. Die Quelle des Inhalts soll also allgemein gehalten werden (vgl. User Story #008). Die Elemente mit dynamischen Inhalten werden als Frame (dt. Rahmen) bezeichnet, da die Inhalte hinter denen im Laufe der Zeit wie Bilder ausgetauscht werden (Abbildung 3.3). Die Bezeichung Window (dt. Fenster) würde auch passen, wäre jedoch mit dem Broweserfenster leicht zu verwechseln.

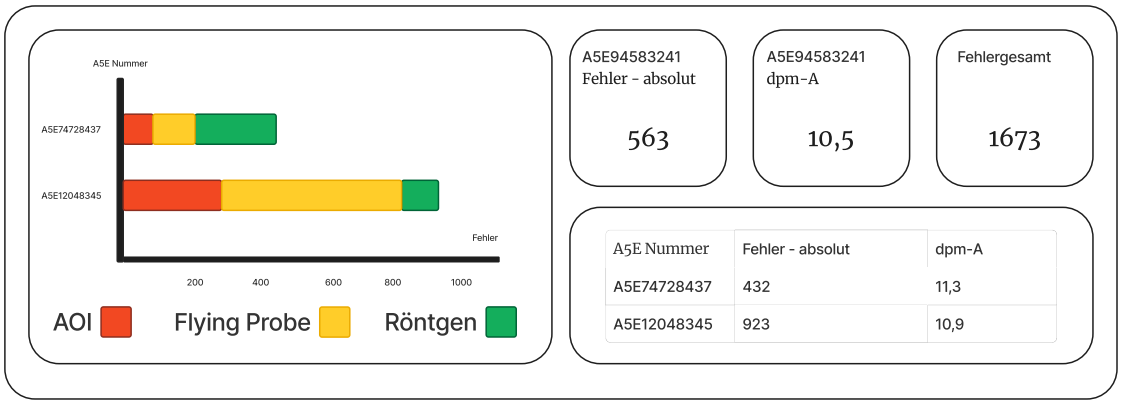

Abbildung 3.3: Wireframe eines Dashboards mit Diagrammen, Tabellen und Kennzahlen. Die Daten sind nicht echt (Quelle: eigene Darstellung).

Alignment (dt. ausgerichtete Anordnung) Bei der Konzeption stellt sich die Frage, was passiert, wenn die Breite des Fensters unterschritten wird, oder das Dashboard auf Endgeräten mit verschiedenen Bildschirmgrößen – z. B. mobilen Endgeräten – besucht wird (User Story #006). Eine klassisches reaktionsfähiges Webdesign ist für dieses Szenario nicht geeignet: Normalerweise gibt es ein *gut bekanntes Layout*, für das Breakpoints definiert werden, also Grenzen der Fensterbreite, an denen die Oberfläche eine neue Strategie zur optimalen Darstellung der Inhalte anwendet, indem sie die Anordnung und Dimensionierung der Elemente anpasst. Aber es gibt kein *gut bekanntes Layout*, da das Dashboard vom Anwender gestaltet wird. Wie soll sich das Dashboard anpassen?

Es ist nicht ratsam, Diagramme und Statistische Kennzahlen dynamisch so klein werden zu lassen, dass sie nicht mehr lesbar sind. Auch nicht jedes Seitenverhältis passt zu jedem Diagramm: Ein in einem balkenförmigen Rechteck gestauchtes Kuchendiagramm ist nicht im Sinne des Erfinders. Man braucht eine allgemein anwendbare Logik, die sich nicht auf dynamische Dimensionierung von Elementen stützt. Die Elemente sollen ihre Größen und Seitenverhältnisse behalten. Wandern die auf dem Bildschirm nicht passenden Elemente vielleicht einfach nach unten? Werden sie nach links oder rechts bzw. oben oder unten ausgerichtet? Oder vielleicht nach einer möglichen Kombination von beiden? Das reaktionsfähige Webdesign braucht eine ausgerichtete Anordnung der umziehenden Elementen. Die naheliegendste Variante ist eine Ausrichtung nach oben und links (Abbildung 3.4), die dem Lesefluss entspricht. Man kann dies möglicherweise in Einstellungen des Dashboards auswählen.

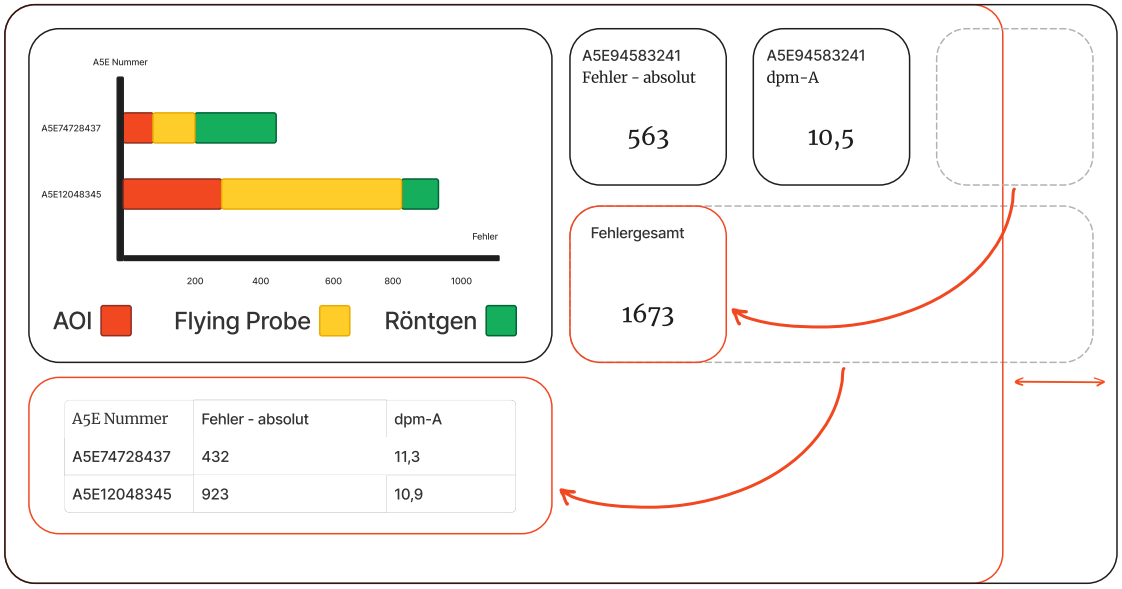

Abbildung 3.4: Wireframe des beispielhaften Dashboardverhaltens beim Unterschreiten der Fenstergröße, wenn Elemente ausgerichtet nach oben und links neugeordnet werden. Die Daten sind nicht echt (Quelle: eigene Darstellung).

Group (dt. Gruppe) Wie man in Abbildung 3.4 sieht, können kleinere Elemente (die Kennzahlen oben rechts) ihre Anordnung verlieren oder auch möglicherweise voneinander getrennt werden. Die Anordnung von zusammenhängenden Elementen spielt beim schnellen Auffinden und der visuellen Wahrnehmung von Informationen eine wichtige Rolle. Vielleicht hat der Anwender die Elemente bewusst so angeordnet. Kann man verhindern, dass zumindest die Anordnung von einer Elementgruppe bestehen bleibt? So kommt die Idee einer *Gruppe* zustande (User Story #012). Eine *Gruppe* von Elementen wird vom Dashboard als ein einziges Element behandelt und darf somit bei Anpassung des Dashboards nicht aufgeteilt werden (Abbildung 3.5).

Es muss allerdings eine Grenze dafür geben, wie weit sich eine Gruppe ausdehnen darf. Wenn eine Gruppe die Fensterbreite erreicht, wird der Umbruch der Gruppenelemente verhindert und die Reaktionsfähigkeit des Dashboard dadurch beeinträchtigt. Es ist sinnvoll, dass eine Gruppe höchstens den Dimensionen des größten Elementes entsprechen darf, z. B. die Hälfte der Fensterbereite, da das größte Element auch nicht umgebrochen werden kann und als die Grenze der Reaktionsfähigkeit des Dashboards betrachtet werden darf. Es bleibt außerdem noch aus, wie man auf intuitive Weise und mit wenig Interaktionen auf Rechner sowie mobilen Endgeräten Elemente gruppieren oder die Gruppierung aufheben kann.

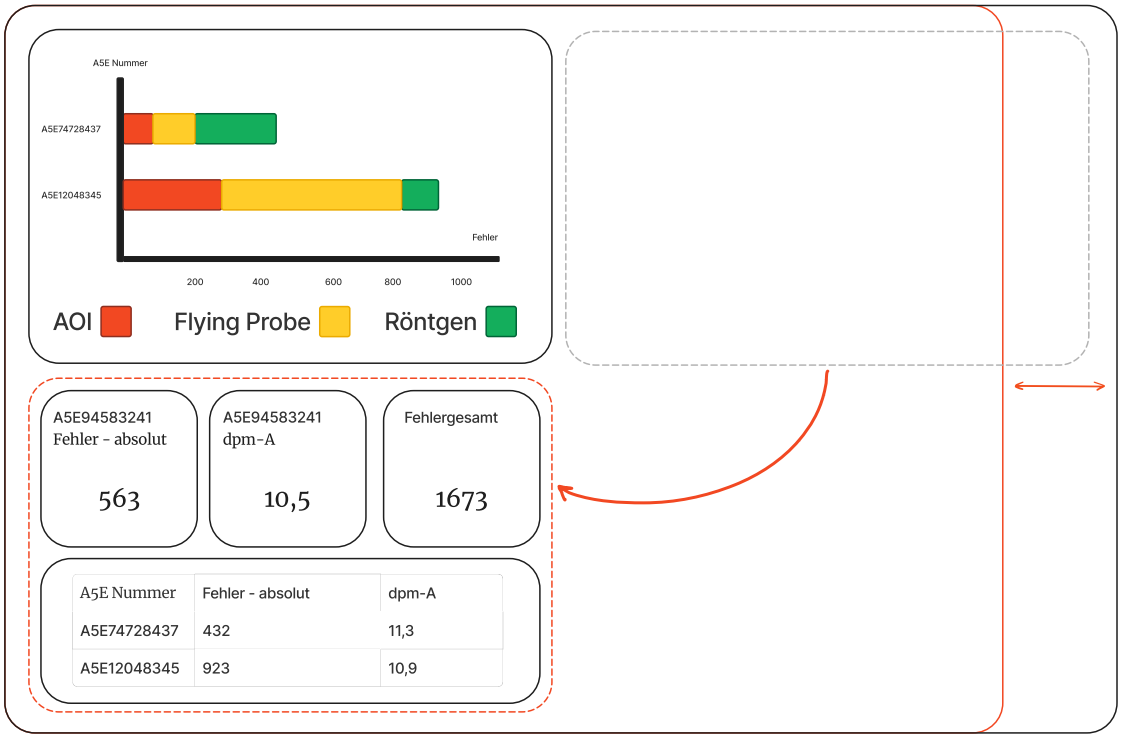

Abbildung 3.5: Wireframe einer Gruppe: Die Elemente innerhalb einer Gruppe wandern zusammen. Die Daten sind nicht echt (Quelle: eigene Darstellung).

Category (dt. Kategorie) Eine weitere Möglcihekeit der Gruppierung bietet die Kategorie (User Story #011). Bei einer Kategorie geht es zwar um die Gruppierung von Elementen, jedoch werden die Elemente nicht fest angeordnet und besitzen anders als bei Gruppen eine Beschriftung. Eine Kategorie verhält sich wie ein verschachteltes Dashboard. Sie grenzt ihre untergeordneten Elemente ein und verbessert die Auffindbarkeit mit einer Überschrift. Außerdem ist eine Kategorie zuständig für die Reaktionsfähigkeit der untergeordneten Elemente. Das ermöglicht noch eine weitere Stufe zur Reduzierung der Entropie bei Umordnung der Elemente, da zusammenhängende Elemente die eigene „Blase“ nicht verlassen (Abbildung 3.6).

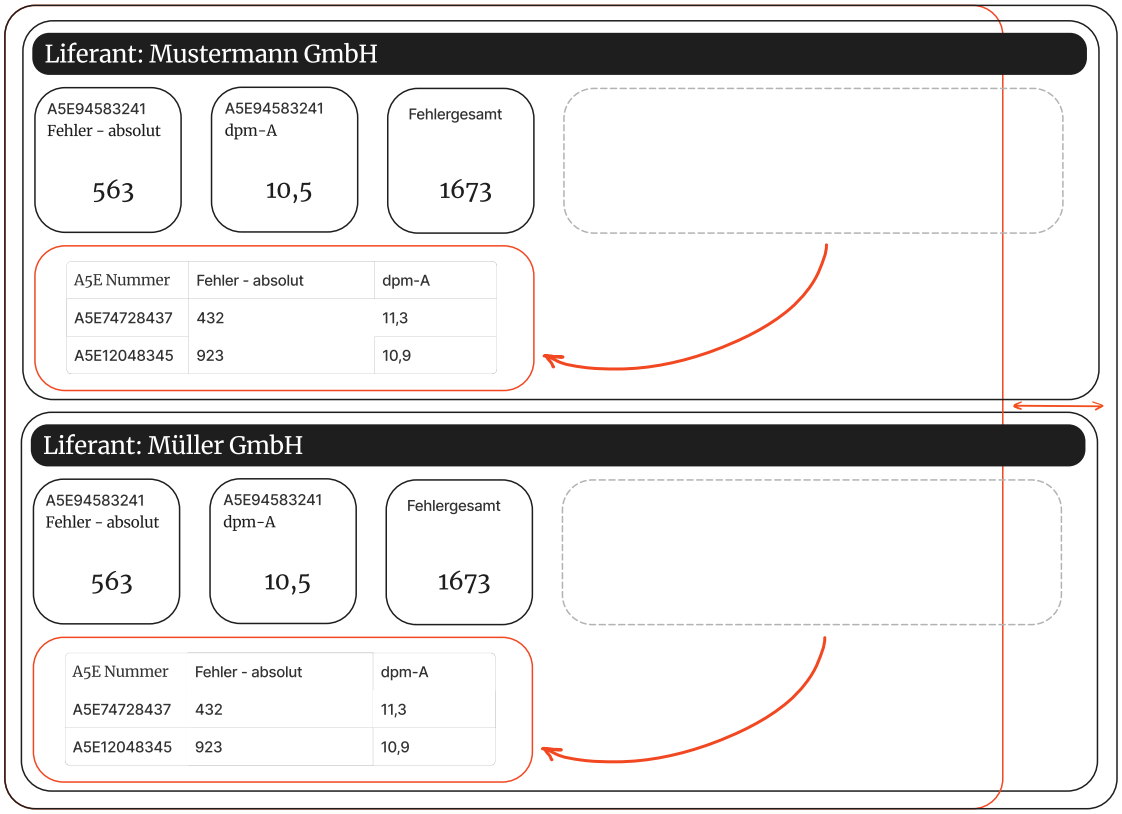

Abbildung 3.6: Wireframe einer Kategorie: Wie ein Unter-Dashboard umfasst und verwaltet eine Kategorie ihre Elemente selbst. Die Daten sind nicht echt (Quelle: eigene Darstellung).

Nun, dass die Grundanforderungen an das Konzept feststehen, kann mit der Recherche nach Implementierungsmöglichkeiten begonnen werden. Die Anordnung von Elementen auf der Oberfläche kann am besten durch ein Grid Layout umgesetzt werden.

3.3 Implementierungsmöglichkeiten

In diesem Abschnitt wird ein Vergleich zwischen Rasterlayout-Bibliotheken durchgeführt, um Siemens eine geeignete Implementierungsmöglichkeit vorzuschlagen. Eine Suche nach Grid Layout im Internet ergibt viele Bibliotheken, die für diese Arbeit nicht relevant sind. Als Grid Layout bezeichnet man nur die Anordnung an sich. Eine verfeinerte Suche nach *draggable grid* (dt. Raster mit verschiebbaren Elementen) liefert passende Bibliotheken auf Basis von JavaScript®, aber auch die typisierte Implementierung von Microsoft®– das Type Script – und auch einige die mit jQuery geschrieben worden sind. jQuery ist eine Skript-Sprache, die auch mit JavaScript® geschrieben wurde, aber mit ihrer kompakten Syntax Entwicklern viel Codierungsaufwand erspart. Daraus werden 5 Bibliotheken ausgesucht, die auf den ersten Blick einen soliden Eindruck machen. Alle werden mit MIT-Lizenz angeboten, d. h., dass die Entwickler die Einbindung in kommerziellen Anwendungen ohne Gebühr erlauben. Folgend werden die Bibliotheken verglichen, aber bevor dies geschehen kann, müssen einige Vergleichsmerkmale festgellegt werden, die eine Beurteilung ermöglichen. Der volle Funktionsumfang der Bibliotheken ist nicht unbedingt für diese Arbeit relevant:

Draggable (dt. verschiebbar) und Droppable (dt. ablegbar) Die Verschiebbarkeit wurde schon erwähnt und ist für diese Arbeit entscheidend (vgl. User Story #005). Es ermöglicht Elemente mit der Maus oder dem Finger über den Hintergrund zu verschieben. Es hat aber an sich keine automatische Anordnung zur Folge. Die Ablegbarkeit beschreibt, ob eine Region oder ein Element das Plazieren eines schwebenden Elementes erlaubt bzw. dafür Platz freimacht oder die Platizierung verweigert. Sie sind komplementäre Voraussetzungen, die die Umordnung von Elementen ermöglichen.

Sortable (dt. sortierbar) vs. Swappable (dt. austauschbar Die Sortierbarkeit von Elementen über das Raster ermöglicht erst die automatische Anordnung nach jeder Verschiebung bzw. Änderung und im weiteren Sinne, das reaktionsfähige Verhalten des Rasters. Die Ausrichtung der Sortierung kann z. B. bei der Bibliothek Muuri eingestellt werden (s. u.). Die Austauschbarkeit beschreibt dagegen, dass beim Verschieben eines Elementes auf ein anderes Element die Plätze vertauscht werden, statt für ein neues Element Platz zu schaffen, sodass die dazwischen liegenden Elemente – insofern die ausgetauschten Elemente dieselbe Größe teilen – nicht verschoben werden müssen.

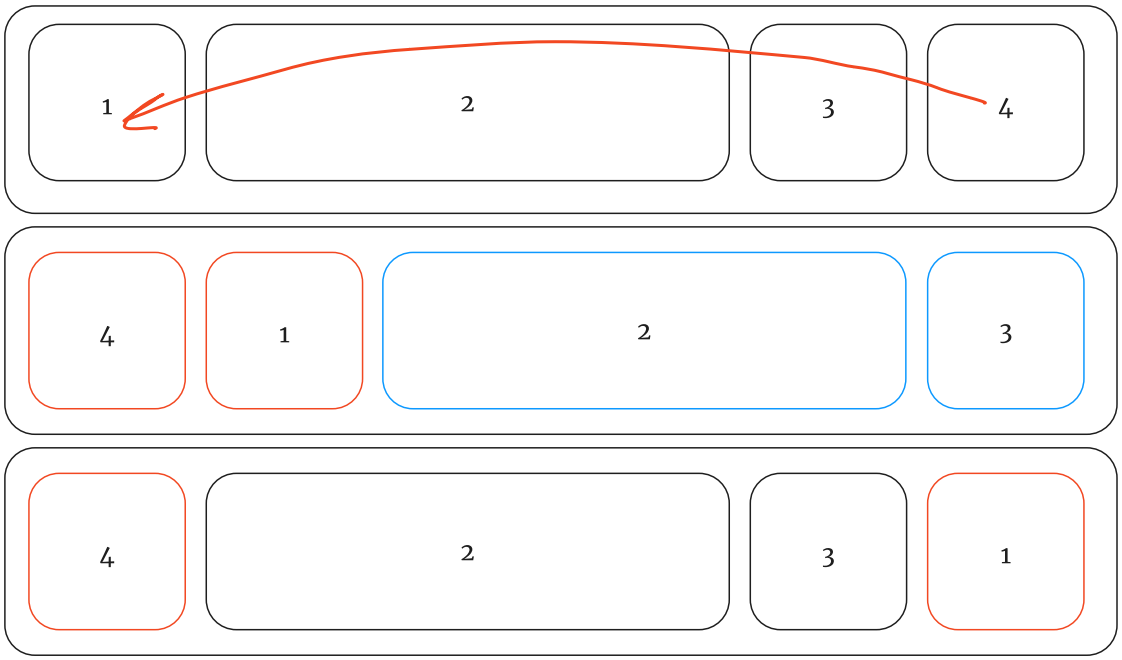

Abbildung 3.7: Die Abbildung zeigt den Unterschied zwischen Sort (Mitte) und Swap (unten) nach Verschiebung von Element 4 auf Element 1. Beim Swap werden die Elemente 2 und 3 nicht verschoben, da die vertauschten Elemente gleich groß sind (Quelle: eigene Darstellung).

Nestable (dt. verschachtelbar) Bietet Raster an, die ineinander verschachtelt werden können. Für die Implementierung von Kategorien und Gruppen sehr relevant. In der Regel sollte es bei jeder Bibliothek möglich sein, da die Raster nur Webelemente sind und Webelemente verschachtelt werden können. Aber hinter dem dynamischen Verhalten steckt viel event handling. Ob die Ereignisse der verschachtelten Raster auf einander abgestimmt sind, muss sich noch bestätigen lassen.

Drag handle (dt. Schiebegriff) Das ist ein Steuerelement, das beim Vorhandensein das Ziehen an Elementen nur an einer bestimmten Stelle erlaubt. Das könnte bei mobilen Endgeräten die Wahrscheinlichkeit reduzieren, dass beim Scrollen die Elemente aus Versehen verschoben werden. Dieses Element ist eigentlich bei verschiebbaren Webelementen immer vorhanden. Wenn die Größe des verschiebbaren Steuerlementes als die des Elementes kleiner ist und durch ein Symbol als solches gekennzeichnet wurde, spricht man von einem Drag Handle.

3.3.1 Muuri

Vorteil Die Bibliothek [44] eignet sich auf den ersten Blick sehr gut zur Umsetzung der in der Terminologie beschriebenen ausgerichteten Anordnung (vgl. User Story #006). Eine saubere, verständliche Dokumentation ist auch vorhanden. Die Ausrichtung

der Anordnung ist änderbar und es wird eine Filterfunktion angeboten, die in Zusammenhang mit User Story #017 eine schnelle Suche nach Elementen bei Dashboards mit vielen Elementen ermöglichen kann.

Nachteil Eine Verschachtelung von Grids wird allerdings in der Dokumentation nicht explizit angegeben. Eine Testimplementierung bestätigt das Fehlen der Koordinierung zwischen hierarchischen Grids. Warum könnte das problematisch sein? Wenn man bspw. versucht ein Element aus einer Kategorie herauszuziehen, können sich das Element und die Kategorie zusammen bewegen.

Es besteht Eingriffsbedarf in die Implementierung von Muuri zur Entkopplung der Schiebesteuerelemente von verschachtelten Grids. Da Muuri auch die nicht minifizierte Version des Codes mit nachvollziehbaren Bezeichnungen und Kommentare für Entwickler bereitstellt, wäre eine Anpassung denkbar.

3.3.2 Sortable

Sortable [45] ist eine Bibliothek mit Fokus auf Sortierung und Grid. Sie macht mit mehrfachen Verschachtelungen einen soliden Eindruck, wo Muuri scheitert. Außerdem sind Vertauschen von Elementen und Mehrfachauswahl der zu verschiebenden Elemente möglich. Man kann die Empfindlichkeit des Grids bei der Reaktion auf Verschiebevorgänge einstellen. Das verbessert das reaktive Verhalten beim Umordnen von Elementen. Außerdem kann man den Schiebegriff aktivieren, um eine versehentliche Verschiebung der Elemente zu verhindern.

3.3.3 Gridstack

Vorteil Der Umfang von Funktionen bei Gridstack [46] ist durchaus zufriedenstellend. Man kann zusätzlich zur Grid-Verschachtelung noch die Elementgröße ändern.

Nachteil Die Bibliothek bietet ein reaktionsfähiges Grid, d. h., die Dimensionen der Elemente werden unter Berücksichtigung der Seitenverhältnisse angepasst. Man kann zwar Breakpoints aktivieren, damit die Elemente umgebrochen werden, aber aufgrund der fehlenden Sortierung, ist die Reihenfolge der Elemente nicht vorhersagbar. Eigentlich kein Nachteil, aber eignet sich nicht für den Anwendungsfall dieser Arbeit.

3.3.4 Fazit

Der Autor empfiehlt Muuri und Sortable als die Bibliotheken der Wahl zur Implementierung der Webanwendung. Die Dokumentation von Muuri ist kompakter und bietet weitere nützliche Funktionen. Sortable ist mehr auf die Eigenschaften des Grids fokusiert und funktioniert sehr robust. Sollte eine Verschachtelung von Grids nicht aufwandarm bei Muuri möglich sein, kann man auf Sortable zurückgreifen.

Kapitel 4

Scout-NextGen Dashboard Konzept

4.1 Architektur

In diesem Kapitel wird Siemens die Vorgehensweise zur Implementierung des Scout-NextGen-Dashboards vorgeschlagen. Abbildung 4.1 zeigt die Architektur der Systemlandschft, in der das Dashboard umgesetzt wird, in einem UML-Komponenetendiagramm. Zunächst wird die Bedeutung der angewandten UML-Notation beschrieben, sollte es dem Leser nicht klar sein.

Eine «*component*» ist ein modularer Teil eines Systems, das seine interne Struktur von der Umgebung abgrenzt und sein Verhalten durch seine Zusammenarbeit mit anderen Komponenten durch Schnittstellen[1] definiert. Eine Komponente kann mit einem vergleichbaren Modul, das genau dieselben Schnittstellen bereitstellt[2] oder braucht[3], ausgetauscht werden [47, S. 224]. Eine «*interface*» ist eine Dienstvereinbarung, d. h. der Dienstanbieter – hier die Oracle® REST Data Services (REST) – soll genau den vom Dienstabnehmer – hier der Web-Client – erwartete Dienstleistung bereitstellen [47, Abschn. 10.4]. Die gestrichelten Linien, die in einem leeren Dreieck enden, zeigen die Bereitstellung des Dienstes bzw. die Implementierung der Schnittstelle. Die mit «*use*»-Schlüsselwort gestrichelten Pfeilen zeigen die Abnahme des Dienstes bzw. die Abhängigkeit der Komponente von der Schnittstelle für eine einwandfreie Funktionsweise. Der Stereotyp «*Subsystem*» beschreibt eine «*component*» als Einheit der hierarchischen Zerlegung des groß-dimensionierten Gesamtsystems, in diesem Fall einer 3-schichtigen Web-Architektur. Eine Web-Architektur ist im Grunde eine Client-Server-Architektur, bei der die Kommunikation zwischen Abnehmer (Client) und Dienstanbieter (Server) durch Web-Protokollen wie HTTP abgewickelt wird. Ein «*Service*» ist eine zustandslose Komponente, die einen Dienst bereitstellt, hier die Web-Schnittstellen zur Datenbank [47, S. 682][48, S. 196].

[1] engl. interface
[2] engl. provided interface
[3] engl. required interface

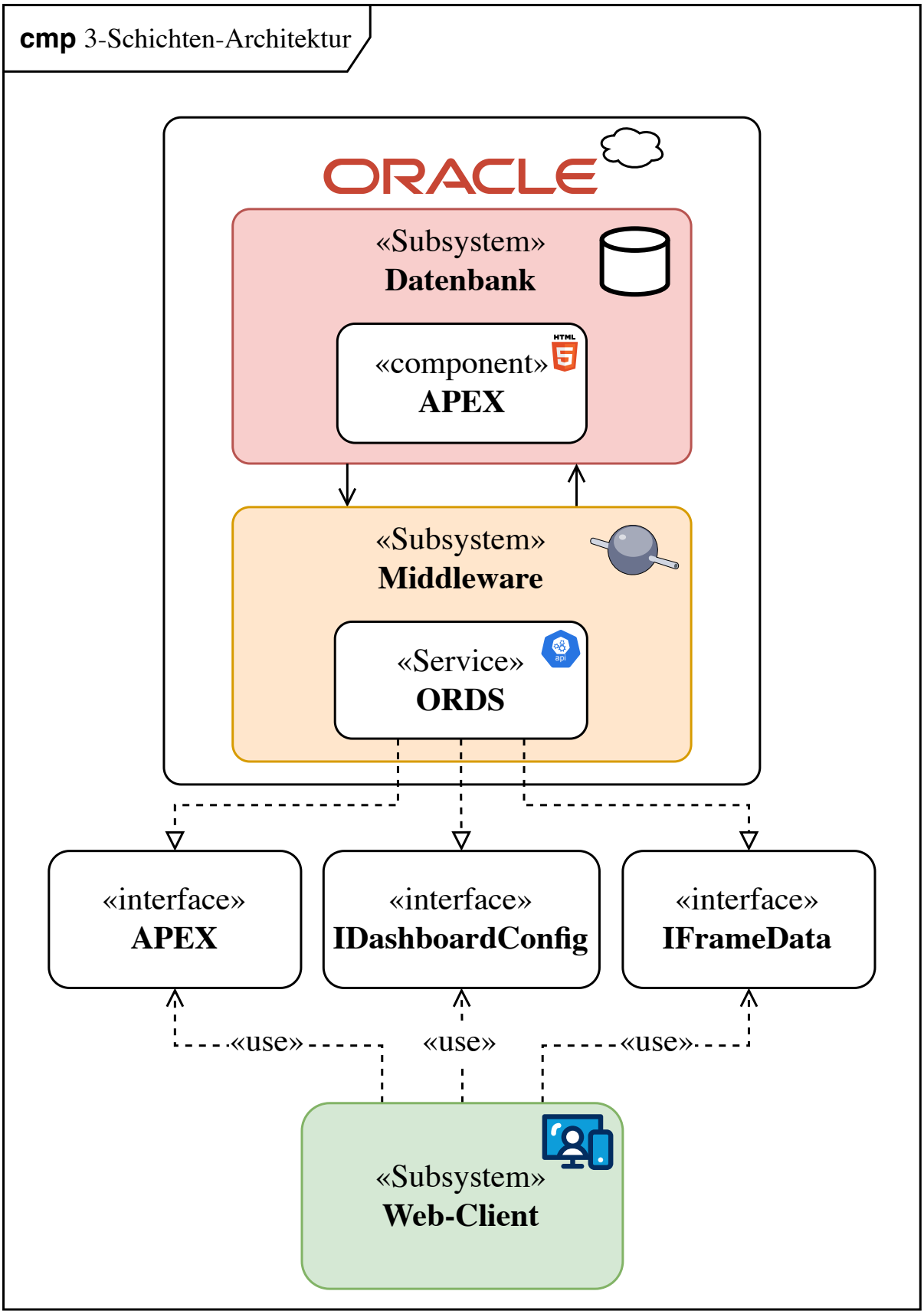

Abbildung 4.1: Das UML-Komponentendiagramm zeigt die Zerlegung der Implementierungsarchitektur auf 3 logische Schichten: Datenbank, Middleware und Client (Quelle: eigene Darstellung).

Ebene vs. Schicht

IBM® sensibilisiert gegen Verwechslung von Ebene (engl. layer) mit Schicht (engl. tier): Ebene weist auf logische (:= software- bzw. code-technische) und Schicht auf physische (:= topologische bzw. infrastrukturtechnische) Verteilung bzw. Trennung der Zuständigkeiten hin (vgl. [49][50], [51][52]). Auch im akademischen Sprachgebrauch spricht man im algebraischen Raum von einer 2-dimensionalen abstrakten Ebene und in der Elektrotechnik von einer physischen Schicht einer Diode.

Immerhin wird oft in der deutschen Literatur zwischen layer und tier durch logische Schicht und physische Schicht unterschieden. In dieser Arbeit wird auch diese Ausdrucksweise verwendet.

Das Diagramm zeigt die Verteilung der Zuständigkeiten auf 3 logische Schichten: Die Server-Dienste die durch Oracle® Datenbank bereitgestellt werden, u. a. APEX Web-Server, der eigentlich durch eine Datenbankprozedur abgewickelt wird, die Zwischenschicht, die als Vermittler zwischen beiden Enden die REST-basierte Kommunikation ermöglicht, und der Web-Client, z. B. ein Rechner oder ein mobiles Endgerät. Die ersten beiden Dienste werden zusammen durch Oracle® Cloud bereitgestellt.

4.2 Webanwendung

4.2.1 Entitäten

Abbildung 4.2 zeigt das ER-Modell[4] der Grundbausteine des Dashboards. Beginnend von oben gibt es 2 abstrakte Entitäten: das Element und der Container. Sie sind beide mit Menüs versehen, die je nach Realisierung relevante Optionen enthalten können. Alle Realisierungen dieser Objekte vererben ein Menü, obwohl das aus Platzgründen in Abbildung 4.3 nicht gekennzeichnet ist

Abstrakte Klassen

Unter Abstrakt ist zu verstehen, dass die Klasse in dieser Auslegung so allgemein formuliert wurde, dass es in der Realität nie konkret realisiert wird und nur grundlegende Eigenschaften für Elemente dieser Art festlegen, die bei der Realisierung (solide Linie beendet mit einem Dreieck) erst spezifiziert werden müssen, wo durch ein konkretes Objekt zustande kommt.

Element Elemente sind Endknoten der Dashboard-Hierarchie. Sie enthalten keine untergeordnete, im Definitionsraum des Dashboards definierte Objekte. Von diesem Typ gibt es 2 konkrete Realisierungen: Frames und Notes. Sie haben i. d. R. eine definierte Größe (siehe Abbildung 4.3).

[4] Entity Relationship

Container Container sind Sammlungen, die Elemente beinhalten dürfen. Sie haben keine festgelegte Größe, da diese die untergeordneten Elemente umfassen müssen. Die Größe muss also durch die Anwendungslogik in der Laufzeit nach jeder Änderung aus Fensterdimensionen und der sich aus der Anordnung der Unterelemente ergebenden Raum berechnet werden. Diese Anwendungslogik soll von der abstrakten Klasse als Ereignishandler (engl. event handler) vererbt werden, die bei Änderungen die Anordnung des Dashboards auffrischt. Somit werden alle Realisierungen der Container-Klasse imstande sein, selbst ihre untergeordneten Elemente reaktionsfähig zu positionieren. Diese Anwendungslogik wird von den meisten Grid-Bibliotheken bereits abgedeckt.

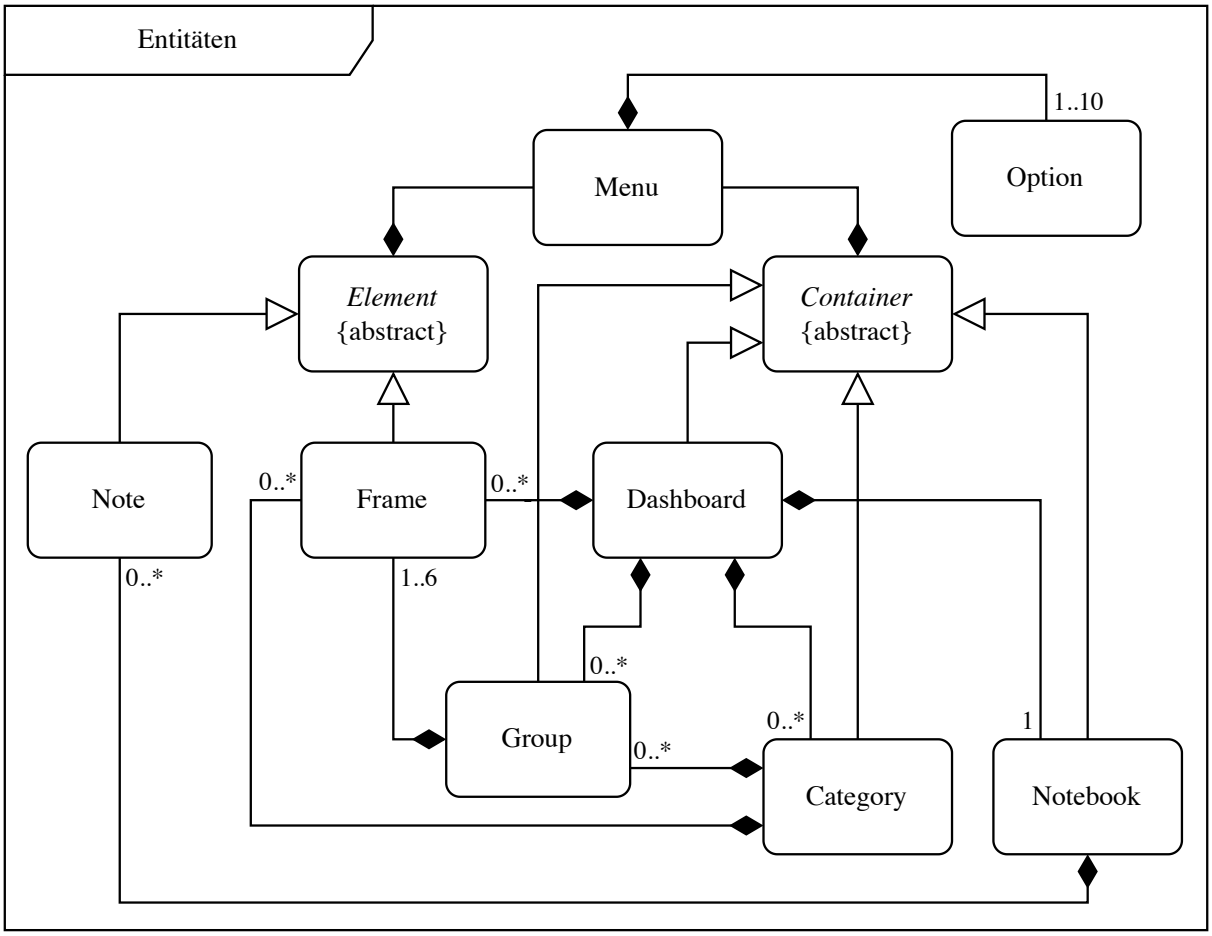

Abbildung 4.2: Das Entity Relationship (ER) Diagramm zeigt die Objektdomäne und die Beziehungen zwischen Objekten (Quelle: eigene Darstellung).

Die wichtigste Realisierung stellt das Dashboard dar. Ein Dashboard darf alle Element und Container enthalten, mit Ausnahme von Note. Technisch gesehen kann jeder Container alle Elemente enthalten. Es wurde die Designentscheidung getroffen, dass man einen gesonderten Container namens Notebook als Ablage für Notizen vorsieht, der bspw. als Seitenleiste ein- und ausgeblendet werden kann. Das maximiert den vorhandenen Platz für Auswertungen (Frames) und erleichtert die Auffindbarkeit der Notizen.

Eine Category ist wie ein kleineres, verschachteltes Dashboard und dient der Organisation der Elemente nach Relevanz. Eine weitere Verschachtelung von Kategorien darf allerdings nicht möglich sein. Eine Group bezeichnet einen Container mit begrenzter Größe, etwa die des größtmöglichen Elementes. Es soll bei kleineren zusammenhängenden Elementen wie Kennzahlen die Elemente anbinden, sodass sie mit derselben Anordnung zusammen verschoben werden. Die reaktionsfähige Anwendungslogik ist im Grunde ausgeschaltet und die Dimension ist strikter einzugrenzen. Allerdings könnte eine Gruppe beim Unterschreiten der Breite doch umbrechen, um Lesbarkeit zu verbessern.

Die mögliche Hierarchie wird hier nochmal zusammengefasst: Ein Dashboard darf unmittelbar mit Ausnahme von Note alle Objekte enthalten. Außerdem hat jedes Dashboard immer genau ein Notebook. Categories dürfen Groups und Frames enthalten. Groups dagegen dürfen maximal bis zu 6 Frames enthalten und Notebook darf nur Notes enthalten. Die Anzahl der Frames, die sich auf einem Dashboard befinden, darf aus Performanz- und Übersichtlichkeitsgründen eine Grenze nicht überschreiten. Diese Grenze ist nach der Implementierung durch Performanztests festzulegen und müssen von durch einen Dienst überwacht und eingehalten werden.

4.2.2 Anwendungslogik

Das Klassendiagramm in der Abbildung 4.3 zeigt ein objekt-orientiertes Modell als Architektur des Webanwendung. Die Datentypen wurden aus Platzgründen gesondert in Abbildung 4.4 komplementär zum Klassendiagramm abbgebildet.

Nun wird die Anwendungslogik der Webanwendung anhand eines Bespielszenarios beschrieben, damit man den Ablauf besser nachvollziehen kann. Diesem Szenario liegt die Annahme zugrunde, dass für jede konkrete Klasse jeweils eine CSS-Klasse definiert worden ist, die den Stil der Klassen geeignet anpasst, insbesondere alle Container-Elemente müssen als Gridelemente angemessen konfiguriert werden.

Abruf Das Szenario beginnt an einem Web-Client, z. B. einem Rechner. Der Benutzer ruft das Dashboard des Scout-NextGen von APEX ab: `https://scout.nex/Dashboard?dashboardId=2067`. Die Webanwendung, also die mit CSS und JavaScript® angereicherte Webseite wird von APEX heruntergeladen und führt nach der vollständigen Ladung ein Skript aus, das die in Abbildung 4.3 dargestellte Klassen initialisiert. Es wird überprüft, ob eine `dashboardId` im URL[5] angegeben wurde. Wenn nicht, bedeutet das, dass der Anwender sein persönliches Dashboard abruft.

[5]Uniform Resource Locator

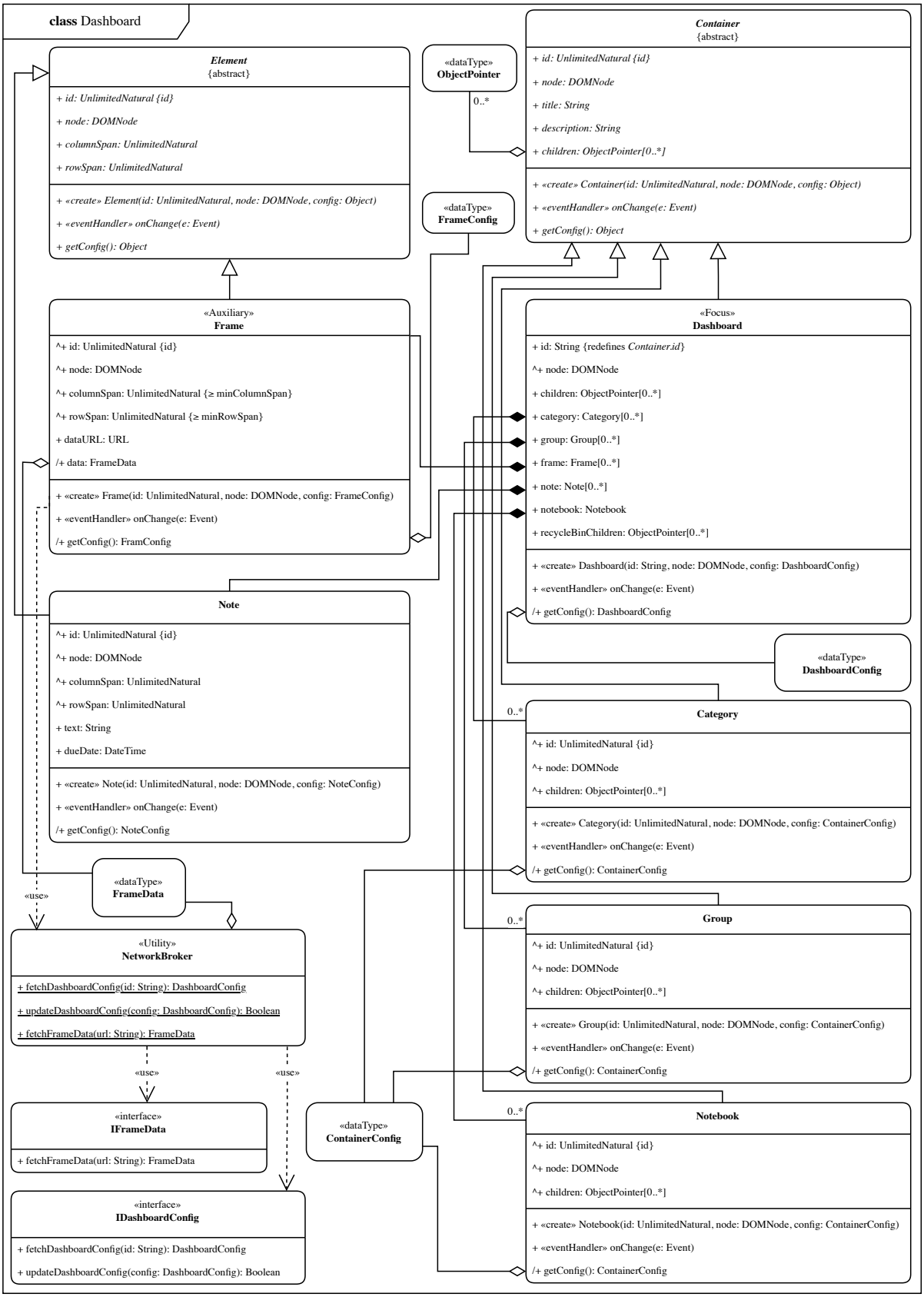

Abbildung 4.3: Das UML-Klassendiagramm zeigt den objektorientierten Aufbau des Dashboards (Quelle: eigene Darstellung).

Das Persönliche Dashboard

Das persönliche Dashboard existiert für jeden Anwender genau einmal und ermöglicht als Startpunkt das Anlegen und Verwalten weiterer Dashboards.

Aufbau Vgl. Code 4.1, auf dessen Zeilen sich die hochgestellten Nummern beziehen: Das Skript ruft durch das Utility `NetworkBroker` die Dashboard-Konfiguration ab[1] und legt die einzige Instanz des Dashboards an[2]. In diesem Beispiel ist `cfg` der strukturierte Datentyp des Dashboards und `dashboard` das lebende Objekt, das anhand der Daten ins Leben gerufen wurde. Das Objekt bekommt die `id = "2067"`[7]. Der Konstruktor der Klasse vermerkt die Wurzel der Webseite, also `document.body` als den entsprechenden Knoten im `node`-Attribut[8], versieht ihn mit der eigenen CSS-Klasse[9], sodass der Stil angepasst und evtl. das Grid aktiviert wird, und bindet den eigenen EventHandler für Änderungen an den HTML-Knoten an[10]. D. h., bei Änderungen des Knoten – z. B. Änderung der Bildschirmgröße – wird die Methode `onChange()` aufgerufen und passt das Dashboard neu an. Hier wird wieder die Analogie zum DOM stark betont: Eine JavaScript®-Klasse und ein HTML-Knoten sind jetzt miteinander verwoben. Der statische HTML-Knoten kann nicht auf Änderungen reagieren, aber informiert eine dynamische Klasse, die geeignete Aktionen vornimmt.

```
1  cfg = NetworkBroker.fetchDashboardConfig("2067")
2  dashboard = new Dashboard("2067", document.body, cfg)
3
4  //Der Konstruktor der Dashboard-Klasse
5  Dashboard(id, node, config)
6  {
7      this.id = id;
8      this.node = node;
9      this.node.className = "Dashboard";
10     this.node.addEventListener("change", this.onChange);
11     //Der Rest des Codes
12 }
```

Code 4.1: Der Beispiel-Aufbau des Dashboard-Konstruktors

«Utility», «Fokus», «Auxiliary»

«Utility» ist eine Werkzeugeklasse, die ausschließlich statische Methoden anbietet und keine Instanzen hat. In diesem Fall implementiert `NetworkBroker` die `IDashboardConfig` und `IFrameData` Schnittstellen, um Daten für Dashboards bzw. Frames abzurufen oder ggf. zu senden. eine *«Fokus»*-Klasse beschreibt die Kernfunktionalität einer Anwendung, die von einer oder mehreren *«Auxiliary»*-Klassen abhängt. Kurzgefasst: Das Dashboard steht im Fokus und erfüllt mit Hilfe von Frames die Grundanforderungen der Webanwendung [47, Abschn. 22.3].

Die Konstruktor legt im nächsten Schritt neue Instanzen der untergeordneten Objekte

an, indem der Konstruktor der jeweiligen Klasse mit der Konfiguration des Objektes aufgerufen wird. Falls das Unterobjekt auch ein Container ist, legt dies analog die eigenen Unterobjekte an, wodurch das baumartige Klassenmodell der Dashboard-Konfiguration zustande kommt. Aufgrund der noch nicht feststehenden Wahl der Bibliothek, mit der das Konzept umgesetzt werden wird, ist eine genau Spezifizierung der Anwendungslogik schwer. Wichtig dabei ist die zentrale Rolle des Dashboards: Das Dashboard fungiert als der zentrale Verwalter der Objekte. Bei Änderungen, wie bspw. Hinzufügen oder Löschen von Elementen, müssen die Elternobjekte bis hin zum Dashboard informiert werden, damit das Dashboard die ihm vorliegende Konfiguration aktualisiert und evtl. andere betroffene Unterobjekte in der Rückrichtung benachrichtigt, sodass alle auf demselben Stand sind.

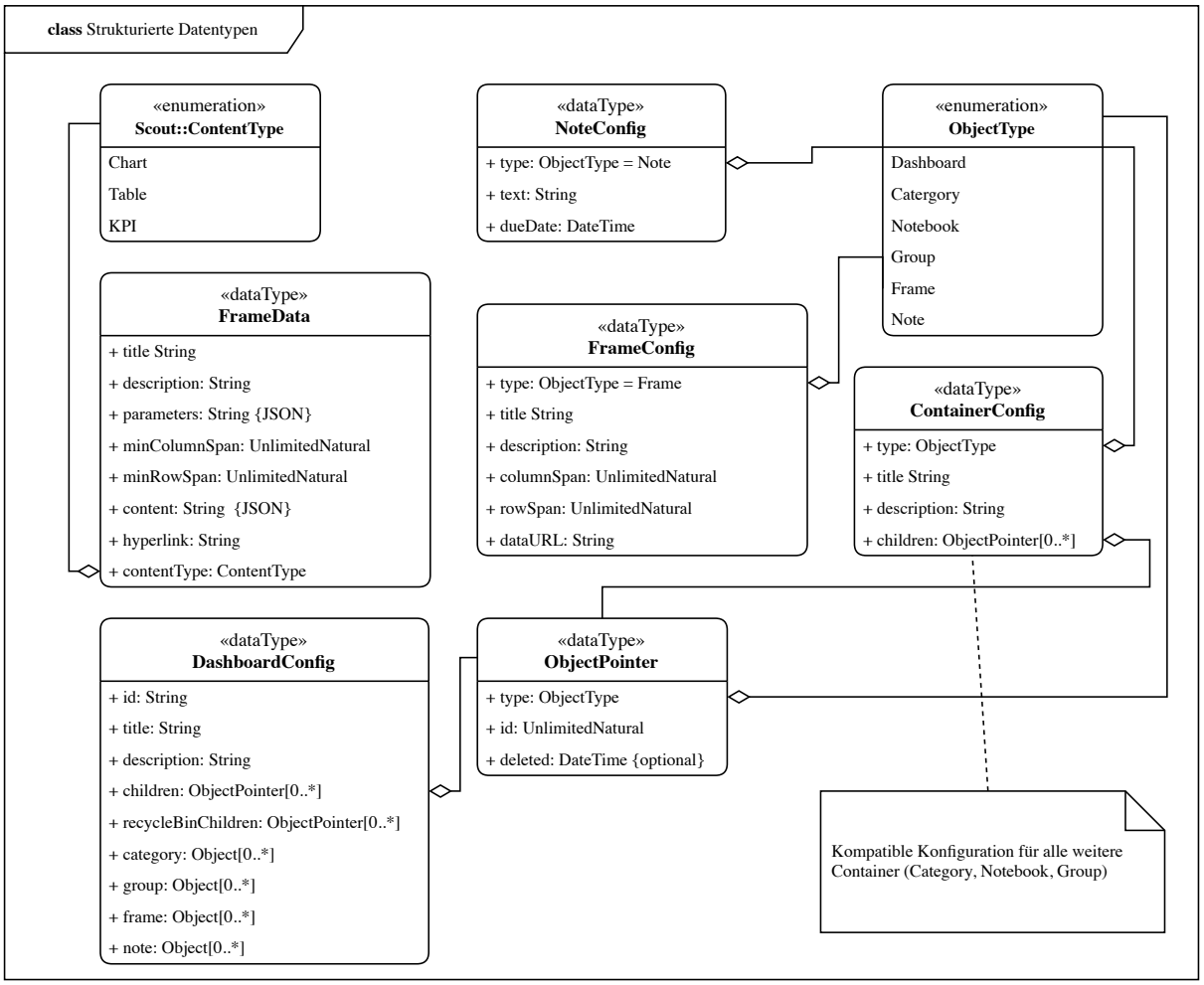

Abbildung 4.4: Das UML-Klassendiagramm der strukturierten Datentypen zur Abbildung 4.3 (Quelle: eigene Darstellung).

4.3 Datenmodell

4.3.1 Strategie

Die Daten kommen clientseitig in JSON zustande und werden auch über die REST-basierte Datenschnittstelle in JSON von POST bzw. PUT Methoden entgegengenommen und auf die Datenbank übertragen. Bis Version 19c der Oracle Datenbank war eine Spalte vom Typ `VARCHAR2` mit der Randbedingung `is json` anzulegen, um eine einfache syntaktische Validierung zu ermöglichen, ob der eingetragene Text ein valides JSON-Dokument ist. Seit der Version 21c ist JSON als Datentyp in der Datenbank integriert. D. h., die Randbedingung `is json` ist implizit in einer JSON-Spalte enthalten. Außerdem ist die Festlegung weiterer Randbedingungen zur Validierung durch JSON-Schema möglich [53]. JSON-Schema sind JSON-Dokumente, die die Struktur anderer JSON-Dokumente beschreiben und somit als Validierungskriterien für JSON-Dokumente verwendet werden können, z. B. durch Festlegung des Datentypes jedes Eintrags oder Bestimmung des erlaubten Wertebereichs. Die Schemen unterstützen sogar reguläre Ausdrücke. Die Validierung ist besonders für die Robustheit und Sicherheit des Konzepts wichtig. Ein einheitlicher Einsatz von JSON auf allen 3 Ebenen erhöht somit die Transparenz und reduziert den Implementierungsaufwand: Clientseitiges JavaScript®, Datenübertragung über JSON-REST und serversitige Validierung durch JSON-Schema und Ablage in JSON-Spalten auf der Datenbank.

4.3.2 Sicherheit

Denial of Service Die von Clients erstellten oder modifizierten JSON-Dokumenten sind nicht vertrauenswürdig. Es ist immer davon auszugehen, das ein Rechner infiziert wurde und versucht die Infrastruktur anzugreifen oder sich zu verbreiten. Alle Einträge in JSON-Dokumenten müssen auf die maximum erlaubte Länge geprüft werden. Ein infizierter Client könnte sehr große JSON-Dokumente an den Server schicken, um einen DoS[6]-Angriff durchzuführen.

Cross-Site Scripting Außerdem ist das Risiko eines XSS-Angriffs hoch, da die Dashboards mit anderen Anwendern geteilt werden können. D. h., datentechnisch werden andere Anwender JSON-Dokumente, die von einem anderen Anwender hochgeladen wurden, herunterladen und in JavaScript®-Objekte verwandeln. Dabei entsteht die Gefahr, das der Prototyp des JSON-Dokuments manipuliert wurde und bei Zuweisung zu einem JavaScript®-Objekt den Prototyp auf Objektebene mit Schadcode infiziert. Dieser Angriff wird als Prototype Pollution (dt. Verschmutzung des Prototyps) bezeichnet [54][55]. Als Gegenmaßnahme dürfen JSON-Dokumente auf keiner Hierarchieebene den Schlüssel `__proto__` beinhalten. Dies ist durch die strikte Definition des Schemas und den Schlüssel `"unevaluatedProperties": false` in einem JSON-Schema möglich. Auf diese Weise werden nicht nur `__proto__`-Felder nicht zugelassen, sondern auch alle andere Felder, die im Schema nicht explizit definiert worden sind [56]. Das Schema

[6] Denial of Service

muss so strikt definiert werden, dass auch nicht gesäuberte Felder nicht zugelassen werden, z. B. HTML-Tags wie `<script>` [57].

4.3.3 Datenmodell

Die Datenhaltung in JSON erfolgt i. d. R. hierarchisch, wobei in Datenbanken ein relationales Datenmodell zum Einsatz kommt. Während beliebige Strukturen in JSON möglich sind , müssen die Datenstrukturen in der Datenbank fest definiert sein. Es liegt daran, dass die Konzepte auf unterschiedliche Probleme ausgelegt sind. JSON sollte eine von Menschen sowie Maschinen leicht lesbare Syntax zum Austausch von objektorientierten Daten zwischen Programmiersprachen liefern. Relationale Datenbanken finden ihre Stärke in effizienter Verwaltung von Daten und Gewährleistung ihrer Integrität. Anhand eines Beispiels wird erläutet, wie die Datenhierarchie bei JSON-Dokumenten einer effizienten Datenschnittstelle im Wege stehen kann und folglich wird eine Lösung dafür vorgeschlagen.

Code 4.2 zeigt ein hierarchies Datenmodell. Es handelt sich um die Konfiguration eines Dashboards mit einer untergeordneten Kategorie, die eine Gruppe mit 2 Frames (Kennzahlen) beinhaltet. Die Reihenfolge der Objekte im jeweiligen Array kennzeichnet die Reihenfolge ihrer Darstellung in dem Elternobjekt. Angenommen, der Anwender möchte die Gruppe auflösen und den ersten Frame auf die Wurzel des Dashboards verschieben. Der erste Frame muss von Zeile 14 auf die Zeile 5 verschoben werden und der zweite Frame von Zeile 20 auf Zeile 10.

Nachteil Die Änderungen dieser Daten müssen auch auf die Datenbank gespiegelt werden. Obwohl man auch auf der Datenbank mit Punkt-Notation durch die JSON-Hierarchie navigieren kann, müssen immerhin relativ große Datenstrukturen zwischen den Hierarchieebenen verschoben werden. D. h., an einer Stelle gelöscht und an einer anderen Stelle wieder eingefügt werden. Ist das überhaupt notwendig?

```
1  {
2      type: "Dashboard",
3      title: "Mein Dashboard",
4      description: "Auswertung",
5      children: [
6          {
7              type: "Category",
8              title: "A5E234325",
9              description: "Auswertungen zu A5E234325",
10             children:[
11                 {
12                     type: "Group",
13                     children:[
14                         {
15                             type: "Frame",
16                             columnSpan: 2,
17                             rowSpan: 2,
```

```
18                                  dataURL: "https://scount.next/ords?
                                       contentType=KPI&reportType=dmp_A
                                       &matNr=A5E234325"
19                              },
20                              {
21                                  type: "Frame",
22                                  columnSpan: 2,
23                                  rowSpan: 2,
24                                  dataURL: "https://scount.next/ords?
                                       contentType=KPI&reportType=
                                       fehler_abs&matNr=A5E234325"
25                              }
26                          ]
27                      }
28                  ]
29              }
30      ]
31  }
```

Code 4.2: Beispiel: Hierarchisches Datenmodell

Code 4.3 zeigt die vorgeschlagene Variante. Die Hierarchie ist abgeflacht, indem jeder Objekttyp auf der Wurzelebene der Konfiguration durch eine geordnete Menge vertreten ist. Es ist also bekannt, wo sich die Objekte des jeweiligen Typs aufhalten, und zwar ohne unnötige Hierarchietiefe. Statt Objekte ineinander zu verschachteln, werden Zeiger auf die Objekte gespeichert: Tupel aus dem Objekttyp und dem Index des Objekts innerhalb der geordneten Menge desselben Typs dienen als Pseudo-Id. Dadurch sind die Objekte eindeutig identifizierbar, wie die Zeilen- bzw. Spaltennummern innerhalb einer Matrix (vgl. Code 4.4). Um den ersten Frame von Zeile 26 dem Dashboard unterzuordnen, wird der Zeiger auf den ersten Frame auf Zeile 5 verschoben und, um die Gruppe aufzulösen, wird der Zeiger auf Frame 2 von Zeile 21 auf Zeile 11 verschoben. Die Gruppe auf Zeile 17 und der Zeiger auf Zeile 12 werden gelöscht.

```
1  {
2      type: "Dashboard",
3      title: "Mein Dashboard",
4      description: "Auswertung",
5      children: [],
6      category: [
7          {
8              type: "Category",
9              title: "A5E234325",
10             description: "Auswertungen zu A5E234325",
11             children:[
12                 {type: "Group", index: 0},
13             ]
14         }
15     ],
16     group: [
```

```
17          {
18              type: "Group",
19              children:[
20                  {type: "Frame", index: 0},
21                  {type: "Frame", index: 1},
22              ]
23          }
24      ],
25      frame: [
26          {
27              columnSpan: 2,
28              rowSpan: 2,
29              dataURL: "https://scount.next/ords?contentType=KPI&
                    reportType=dmp_A&matNr=A5E234325"
30          },
31          {
32              columnSpan: 2,
33              rowSpan: 2,
34              dataURL: "https://scount.next/ords?contentType=KPI&
                    reportType=fehler_abs&matNr=A5E234325"
35          }
36      ],
37  }
```

Code 4.3: Beispiel: Abgeflachtes Datenmodell

Vorteil Die Daten sind leicht auffindbar und die Hierarchietiefe ist reduziert. Statt der zu verschiebenden Objekte werden sehr kleine Datenstrukturen (die Zeiger) gelöscht und neu geschrieben. Wenn man den Typ auf eine Aufzählung umstellt und die Schlüssel kompakter benennt, schrumpft der Zeiger auf das Nötigste und optimiert auch die Netzwerkdatenverkehr bei der Synchronisierung des Dashboards mit dem Server. Beispiel: `{t:4,i:1}`. Die eingebrachte Ordnung in der Datenstruktur erleichtert außerdem die Definition eines JSON-Schemas zur serverseitigen Validierung. Dieses Modell kann auch bei der Umsetzung des Papierkorbs (User Story #010) sehr gelegen kommen, da die Daten zentral von der Dashboard-Klasse verwaltet werden und beim Entfernen von Objekten nicht die Objekte gelöscht werden müssen, sondern die Zeiger. Die Zeiger können jedoch auch auf einen Container verschoben werden, der beim Öffnen des Papierkorbs eingeblendet wird und die Wiederherstellung der entfernten Objekte auf das Dashboard ermöglicht.

Nachteil In diesem Speziallfall muss auch die Kategorie den Zeiger auf die aufzulösende Gruppe entfernen, aber die Gruppe weißt nicht von ihrem Elternknoten, da nur eine einfache Verkettung vorliegt. Dem Verzicht auf doppelte Verkettung – also Verkettung nach Kinder- sowie Elternknoten – liegt die Optimierung des Speicherbedarfs zugrunde. Es müssen also alle Objektinstanzen darüber informiert werden, dass sie den Zeiger auf die Gruppe – falls vorhanden – entfernen müssen, da die Zeiger in diesem Datenmodell an die Stelle der Objekte treten und wie bei den Objekten

dürfen sie jeweils genau in einem Exemplar vorhanden sein. Außerdem muss man darauf achten, dass die Anordnung im Array als Id des Objektes dient. Wenn also ein Zwischenelement gelöscht werden muss, darf das Array nicht neu geordnet werden. Stattdessen wird das Objekt auf `null` gesetzt, um die Anordnung zu bewahren, und beim hinzufügen des nächsten Objektes werden zuerst die `null`-Objekte wieder belegt.

Fazit Die transparente, abgeflachte Datenhierarchie bietet viele Vorteile. Allerdings muss die fehlende Hierarchie in Daten mit transparenter Kommunikation zwischen Klassen kompensiert werden.

```
1  //Konfiguration des Dashboards
2  {
3      //Attribute des Dashboards
4          ...
5
6      //Array aus Zeigern auf Unterobjekte des Dashboards
7      children:   [{...}, {...}, {...}, ...],
8
9      //Arrays aus Konfigurationen von allen Dashboard-
           Unterobjekten
10     category:   [{...}, {...}, {...}, ...],
11     group:      [{...}, {...}, {...}, ...],
12     frame:      [{...}, {...}, {...}, ...],
13     note:       [{...}, {...}, {...}, ...]
14 }
15
16 //Konfiguration aller anderen Container
17 {
18     //Attribute des Containers
19         ...
20
21     //Andere Container haben nur Zeiger auf ihre Unterobjekte
22     children:   [{...}, {...}, {...}, ...]
23 }
```

Code 4.4: Beispiel: Datenmodell für die Dashboard-Konfiguration

Kapitel 5

Diskussion und Fazit

5.1 Zusammenfassung

In dieser Arbeit ging es um die Konzeption eines benutzerdefinierten Dashboards für die Qualitätsexperten der Siemens AG. Das Dashboard soll dem Anwender eine Übersicht über wichtige Informationen verschaffen, die von ihm entsprechend seinen Aufgaben zusammengestellt wurden, und somit als zentrale Überwachungstafel der neuen Version der hauseigenen Qualitätsdatenplattform *Scout-NextGen* dienen, die Qualitätsauswertungen in Form von diversen Diagrammen, Tabellen und Kennzahlen bereitstellt. Das Dashboard soll von jedem Anwender ohne IT-Fachkenntnisse bedienbar sein, d. h., der Anwender soll eigenständig Dashboards anlegen und verwalten, Auswertungen aus dem *Scout-NextGen* zum gewünschten Dashboard hinzufügen, und das Dashboard einrichten können, z. B. durch Umordnen oder Organisieren von Auswertungen. Das reaktionsfähige Verhalten des Dashboards soll es ermöglichen, dass sich das Dashboard jeder vom Anwender vorgenommenen Änderung automatisch anpassen lässt, z. B. durch Neusortierung der Auswertungen. Die Anpassung soll auch für gängige Bildschirmgrößen gelten, auch die der mobilen Endgeräte. Schließlich soll das Dashboard per Link mit anderen Mitarbeitern geteilt werden können, um eine Zusammenarbeit auf gemeinsame Aufgaben zu erleichtern.

Im Kapitel 2 wurde eine Übersicht der für diese Arbeit wichtigsten theoretischen Grundlagen präsentiert. Die Grundbegriffe wurden wo möglich in einem chronologischen Ablauf erklärt, um deren Zusammenhänge und die von ihnen zu lösenden Probleme besser nachzuvollziehen.

Im Kapitel 3 wurde die Vorgehensweise der Arbeit geschildert, beginnend mit der Analyse. Zunächst wurden die Stakeholder identifiziert und ihre Ansprüche wurden abstrahiert. Als Nächstes wurden die wichtigsten Anwender, die im Arbeitsalltag viel mit der Plattform *Scout* in Berührung kommen, über ihre Erwartungen von dem Dashboard befragt. Die Interviews wurden in Zusammenarbeit mit dem Auftraggeber ausgewertet und validiert, um allgemein formulierte, sich wiederholende oder den Aufgabenbereich des Dashboards nicht betreffende User Stories auszuschließen. Es wird versucht, prägnante Akzeptanzkriterien zu formulieren, die das Ziel der Anfor-

derungen präzise umreißen und als Bewertungskriterien für das Konzept und Testfälle für die künftige Implementierung geeignet sind. Komplementär zu den Anforderungen wurde die IT-Landschaft der Siemens AG, woraufhin das Dashboard konzeptioniert werden soll, kurz erwähnt. Der Stand der Technik wurde in Kenntnis gebracht, um die Relevanz der zu leistenden Arbeit zu schildern. Die Grundkonzepte des Dashboards wurden spezifiziert, um die Basis für den Ausbau des Konzepts bereitzustellen und die Begriffe wurden begleitend mit groben Wireframes fürs Obeflächendesing veranschaulicht. Letzendlich wurde ein Bibliothekenvergleich durchgeführt und eine Bibliothek zur aufwandarmen Implementierung empfohlen.

Im Kapitel 4 wurden die Arbeitsergebnisse präsentiert. Es wurde erklärt, wie Siemens AG das Scout-NextGen-Dashboard in der ihrer IT-Landschaft umsetzen und einbinden kann. Zunächst wurde die Schichtenarchitektur angegeben, um die Zuständigkeitsbereiche zu trennen. Als Nächstes wurde anhand des ER-Modells die Beziehung der Objekte in der Fachdomäne veranschaulicht und dann durch UML-Klassendiagramme konkretisiert. Die grobe Anwendungslogik wurde beschrieben, die abhängig von der Auswahl der Bibliothek mit Anpassungen umgesetzt werden soll. Die Datenstrukturen einschließlich der Datenschnittstellen wurden angegeben. Analog zur objektorientierten Struktur der Daten wurde auch das Datenschema angegeben, nach dem die Daten auf der Datenbank abgespeichert werden.

5.2 Evaluation

Das Konzept besteht aus 2 Hauptteilen: Die Anwendungslogik der Webanwendung, mit Vorschläge zur Umsetzung der Benutzerschnittstelle, und Datenmodelle zur persistenten Speicherung auf der Oracle Datenbank und Übertragung durch die ORDS-Datenschnittstelle.

Eine tauglicher Prototyp wäre wünschenswert, selbst wenn der Fokus der Arbeit auf dem Konzept liegt. Es gibt noch Raum für Verbesserungen: Das Menü (User Story #009) soll noch realisiert werden, aber hat keinen großen Einfluss auf die Leistung des Dashboards, da es hauptsächlich als Ansiedlung für Dashboard-Aktionen wie das Hinzufügen (User Story #003) und Entfernen (User Story #004) von Elementen gedacht ist. Die Liste der angelegten Dashboards (User Story #002) kann durch eine Methode in der Klasse *NetworkBroker* umgesetzt werden, die die nach Benutzer-Id des Eigentümers gefilterte Dashboard-Konfigurationen von der Datenbank abfragt. Es ist noch zu klären, ob der Papierkorb (User Story #010) von der Dashboard-Klasse implementiert wird, oder eine neue Klasse bzw. weitere Instanz des Dashboards die Aufgabe übernimmt. Immerhin ist durch die zentrale Verwaltung der Objektkonfigurationen durch die Dashboard-Klasse, wie unter Unterabschnitt 4.3.3 erläutert wurde, eine wichtige Vorarbeit in der Richtung geleistet worden. Auf die Benachrichtigungsschnittstelle (User Story #015), den PDF-Auszug (User Story #016) und das Suchfeld (User Story #017) wurde aufgrund der niedrigen Priorität verzichtet, die anhand der Anforderungsspezifikation jedoch integrierbar sind. Die Anwendungslogik sollte spezifischer sein, ist aber aufgrund der nicht feststehende Wahl der Bibliothek allgemein gehalten worden.

Das Dashboard ist eine moderne Webanwendung und soll auf Rechnern und mobilen Endgeräten bedienbar sein. Die Änderungen des Dashboards sollen in quasi Echtzeit mit dem Server synchronisiert werden. Es ist allerdings noch ein offener Punkt bzw. ungeklärt, was geschieht, wenn das Dashboard auf mehreren Endgeräten gleichzeitig offen ist. Es muss eine Kommunikationsschnittstelle oder eine Art Broker auf dem Server vorhanden sein, der die Änderungen mit wenig Verzögerung auf alle aktive Sitzungen überträgt. Zwei nebeneinander geöffnete Dashboards sollen also mit einer akzeptablen Verzögerung bei Änderungen synchronisiert werden. Sonst ist es nicht klar, welche Änderung auf dem Server übernommen würde und die Änderungen werden u. U. überschrieben. Eine mögliche Überbrückung wäre, auf der Datenbank eine Spalte für die letzte Änderung anzulegen und jede 5–10 Sekunden Änderungen abfragen und das Dashboard bei Bedarf neuladen.

Die Anforderungen (User Stories) wurden gut spezifiziert, sodass sie zur zukünftigen Verbesserungen und Erweiterungen des Konzepts herangezogen werden können. Das Konzept liefert zwar gute Einblicke in die unterschiedlichen Bereiche der Webentwicklung – von Oberflächendesign bis in die Datenbank – und kann einem Entwickler gute Leitfaden bei der prototypischen Umsetzung des Dashboards liefern, ist jedoch als eigenständige technische Dokumentation zur Umsetzung eines ausgereiften Produktes nicht geeignet.

5.3 Einschränkungen

Das größte Problem war die Einarbeitung, da der Autor auf dem Aufgabengebiet (Webentwicklung) nicht spezialisiert ist. Die Informatik ist auch sehr breit gefächert: Von Eingebetteten Systemen und hardwarenaher Programmierung durch low-code bis hin zur Infrastruktur. Die Philosophien hinter den Vertiefungsrichtungen unterscheiden sich sehr stark. Je mehr man im Laufe der Arbeit über das Fachgebiet lernt, desto mehr fallen einem bessere Alternativen ein und der Wunsch nach Überarbeitung der schon geleisteten Arbeit wächst, was ein großes Problem für die Einhaltung der Abgabefrist darstellt. Immerhin ist das bei einer Abschlussarbeit vorgesehen und ist sogar das Ziel, mit so einer Situation in Berührung zu kommen und versuchen, das Gleichgewicht zwischen der Fachkompetenz, der Leistung und der zeitlichen Effizient herzustellen. Warum sollte es denn sonst diesen Abschnitt geben?

Außerdem war der Bibliothekenvergleich nicht sehr zufriedenstellend. Man soll sich entweder bis ins Detail mit mehreren Bibliotheken auseinandersetzen, vorausgesetzt dass sie gut dokumentiert sind, was viel Zeit kostet, oder sich auf die bereitgestellten Demos verlassen. Die Demos können aber trügerisch sein: Sie können weitere Anwendungslogik beinhalten, die nicht von der Bibliothek bereitgestellt wird. Dadurch könnte man sich den Implementierungsaufwand leicht verschätzen. Wenn eine Bibliothek bspw. „tolle Unterstützung für mobile Endgeräte“ angibt, ist die Aussage mit Vorsicht zu genießen, was u. a. dazu geführt hat, dass keine prototypische Implementierung vorhanden ist.

5.4 Ausblick

Bei Angelegenheit würde der Autor die User Experience (UX) feinschleifen: Wie kann man den Interaktionsbedarf mit dem Dashboard minimieren? Wie kann man die Arbeit mit dem Dashboard noch intuitiver gestalten? Besonders interessant ist die Bedienung an mobilen Endgeräten, da einerseits die Interaktionsart mit dem Dashboard aufgrund der Touchbedienung sich unterscheidet und andererseits dem Dashboard nicht viel Platz zur Verfügung steht. Was der Autor persönlich bei der Bedienung mobiler Endgeräte als unangenehm empfindet, ist die unabsichtliche Betätigung eines Steuerelements beim Scrollen. Auf dem Dashboard konnte bspw. ein Frame beim Scrollen unabsichtlich verschoben werden. Die richtige Herausforderung ist, trotz der unterschiedlichen Interaktionsarten eine gemeinsame Sprache (engl. design language) für die Bedienung des Dashboards an Rechnern und mobilen Endgeräten zu finden, sodass sich der Anwender beim Hin- und Herwechseln zwischen Geräten nicht umgewöhnen muss (vgl. ISO 9241-210:2019, Ergonomie beim menschenzentrierten Designs).

Wenn man von Intaraktionsarten spricht, darf man die größte Revolution der Mensch-Maschinen-Interaktion nicht außer Acht lassen: Mit der Entwicklungsgeschwindigkeit der KI[1] ist in naher Zukunft die Anbindung des Dashboards an die GenAI[2] zu erwarten. Durch Sprachbefehle soll es zukünftig möglich sein, verfeinerte Auswertungen zum Dashboard hinzuzufügen, ohne die Auswertung auf *Scout-NextGen* überhaupt aussuchen zu müssen.

5.5 Fazit

Im Rahmen dieser Arbeit sollte ein Konzept für ein benutzerdefiniertes Dashboard für *Scout-NextGen* – die Qualitätsdatenplattform des Elektronikwerk Ambergs – bei Siemens AG entwickelt werden, sodass die Qualitätsexperten eigenständig eigene Sammlungen aus für sie relevanten Qualitätsauswertungen in Form von diversen Diagrammen, Tabellen und Kennzahlen zusammenstellen können.

Nach der Anforderungsspezifikation wurden die Anforderungen ausgearbeitet und validiert, wodurch das Dashboard-Konzept zustande gekommen ist. Als Arbeitsergebnis wird eine Architektur aus 2 Modulen – Webanwendung und Datenmodell – zusammen mit Leitfaden zur prototypischen Umsetzung des Dashboards und einer Bibliothekempfehlung zur Implementierung der Webanwendung dem Aufgabesteller geliefert.

[1] künstliche Intelligenz
[2] Generative Artificial Intelligence

Kapitel 6

Anhang: User Stories

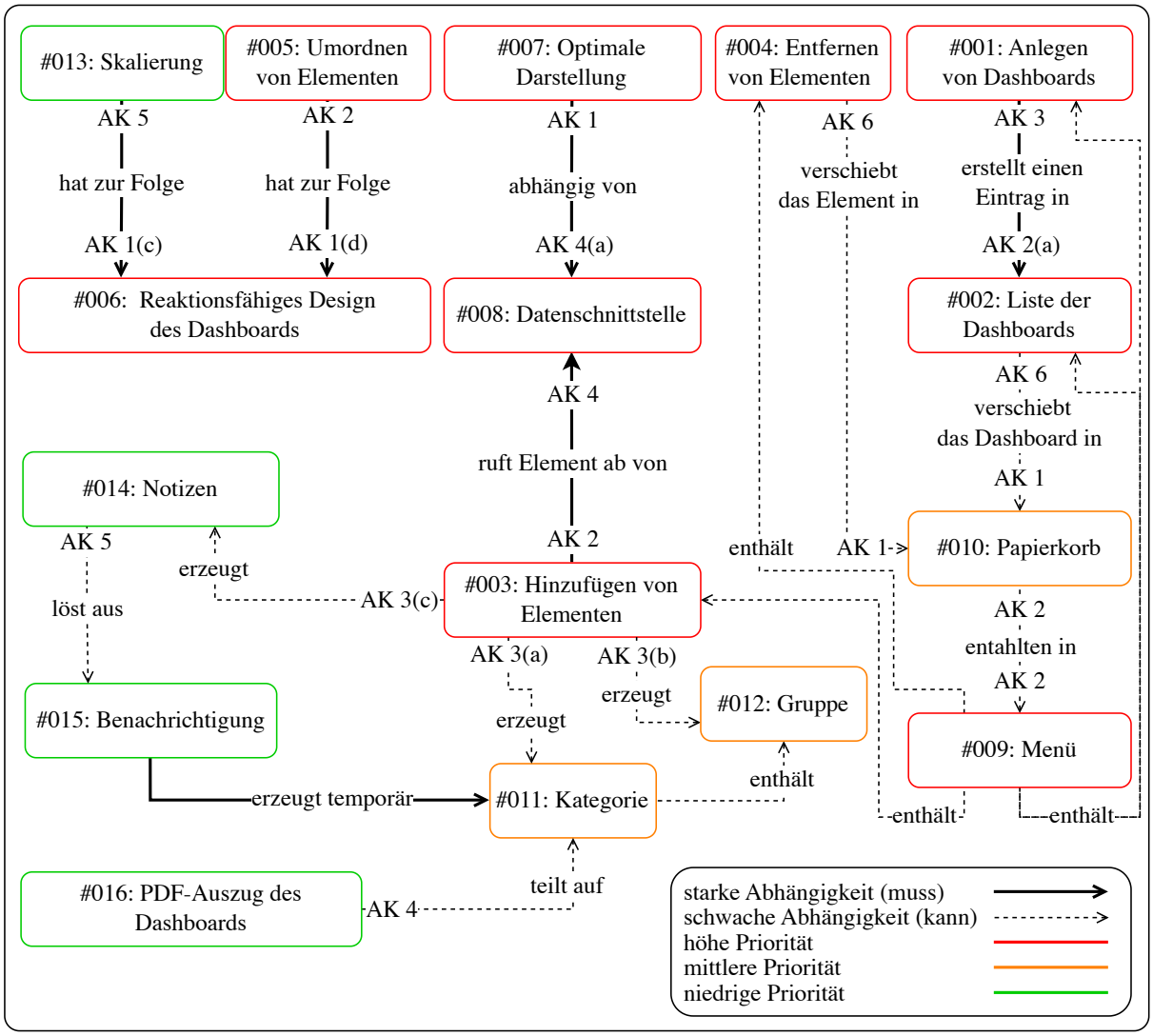

Abbildung 6.1: Die Abhängigkeiten zwischen den formulierten User Stories evtl. bis zum betroffenen Akzeptanzkriterium (AK) hin. Der Pfeil zeigt die Leserichtung. Beispiel-Lesung von oben rechts: Durch Anlegen von Dashboards (AK 3 von User Story #001) wird ein neuer Eintrag in Liste der Dashboards (AK 2(a) von User Story #002) erzeugt.

User Story #001: Anlegen von Dashboards (Vorrang: hoch)

Stakeholder	Auftraggeber (QDM), Nutzer
Wunsch	Anlegen bzw. Erzeugen neuer Dashboard-Instanzen
Nutzen	Aufgabenbezogene bzw. teilbare Dashboards oder auch als Dashboard-Vorlage für neue Nutzer

Akzeptanzkriterien:

1. Eine neue Instanz des Dashboards kann erzeugt werden:
   (a) entweder als neues (leeres) Dashboard
   (b) oder durch Duplizieren eines vorhandenen Dashboards (als Vorlage)
2. Das neuerzeugte Dashboard ist persistent und durch einen eindeutigen Link abrufbar und teilbar.
3. Das neuerzeugte Dashboard wird unter „meine Dashboards“ auf der Liste der Dashboards aufgelistet (siehe User Story #002: Liste der Dashboards).

Kommentare:

1. Sollen angelegte Dashboards für andere Nutzer zugänglich sein?
   Ja, für alle Scout-User
2. Dürfen die geteilten Dashboards von jedem mit Zugriff auf den Link bearbeitet werden oder nur vom Eigentümer?
   Nur vom Eigentümer
3. Was geschieht mit einem Dashboard, nachdem das Eigentümerkonto nicht mehr existiert?
   Als besitzerlos markieren, wenn nach einer Frist kein Abruf erfolgt, dann löschen.

User Story #002: Liste der Dashboards (Vorrang: hoch)

Stakeholder	Auftraggeber (QDM), Nutzer
Wunsch	Eine Liste der angelegten, besuchten und favorisierten Dashboards.
Nutzen	Überblick, Verwaltung und Zugriff auf alle relevanten Dashboards.

Akzeptanzkriterien:

1. Eine Liste wird bereitgestellt.
2. Die Liste umfasst getrennt:
   (a) die eigenen angelegten Dashboards bzw. *meine Dashboards* (siehe User Story #001: Anlegen von Dashboards) und
   (b) eine Liste der favorisierten Dashboards (Favoriten) und
   (c) eine Historie der geteilten Dashboards (zuletzt besucht).
3. Ein (durch den Link) geteiltes Dashboard kann:
   (a) beim Besuchen oder
   (b) durch die Historie der geteilten Dashboards als Favorit markiert werden.
4. Nach Favorisieren wandert der Dashboard-Eintrag unter die Favoriten.
5. Nach Abwählen als Favorit kehrt der Dashboard-Eintrag unter die zuletzt Besuchten zurück.
6. Die eigenen Dashboards können gelöscht werden und wandern zur Dashboard-Liste des Papierkorbs (siehe User Story #010: Papierkorb).

Kommentare:

1. Wie viele Dashboards darf die Historie umfassen und wie lang?
   Die 5 zuletzt besuchten
2. Wie viele Dashboard darf ein Nutzer maximal anlegen?
   Gerne Konfigurierbar

User Story #003: Hinzufügen von Elementen (Vorrang: hoch)

Stakeholder	Auftraggeber (QDM), Nutzer
Wunsch	Hinzufügen von neuen Elementen
Nutzen	Personalisierbarkeit des Dashboards

Akzeptanzkriterien:

1. Ein Auswertung- oder KPI-Element kann auf *Scout-NextGen* per Knopfdruck zu einem Dashboard der Wahl hinzugefügt werden.
2. Die Elementinformationen werden per Datenschnittstelle abgerufen (siehe User Story #008: Datenschnittstelle).
3. Sonstige Elemente können auch zum Dashboard hinzugefügt werden:
   (a) Kategorie (siehe User Story #011: Kategorie),
   (b) Gruppe (siehe User Story #012: Gruppe),
   (c) Notizen (siehe User Story #014: Notizen).
4. Ein neues Element (z.B. eine neue Scout-Auswertung) erscheint auf dem Dashboard.
5. Das Element ist beim Neuladen des Dashboards (persistent) vorhanden.

Kommentare:

1. Modalität:
   (a) Wie wählt man das Ziel-Dashboard aus?
   Die Liste der Dashboards einblenden -> zum ausgewählten Dashboard senden
   (b) Welche Schnittstelle wird benutzt und welche Informationen werden benötigt?
   REST-API bevorzugt. Alle (von Element/Auswertung abhängigen) Parameter sind im REST-URL enthalten.
2. Anordnung: Soll das neue Element beim Hinzufügen
   (a) z.B. durch die Maus frei positionierbar sein oder
   (b) vorerst am Anfang/Ende des Dashboards zur anschließenden Anordnung durch den Nutzer abgelegt werden?
   Vorpositioniert
   (Idee: ganz oben taucht eine neue Kategorie auf, die neue Elemente umfasst)
3. Dimension: Wie groß soll das jeweilige Element sein und warum?
   (Idee: Die Mindestgröße wird je nach Inhalt von der Schnittstelle bestimmt)

User Story #004: Entfernen von Elementen (Vorrang: hoch)

Stakeholder	Auftraggeber (QDM), Nutzer
Wunsch	Entfernen von vorhandenen Elementen
Nutzen	Personalisierbarkeit des Dashboards

Akzeptanzkriterien:

1. Ein vorhandenes Element kann vom Dashboard entfernt werden.
2. Das entfernte Element wandert unter den Papierkorb (siehe User Story #010: Papierkorb).
3. Das Element soll beim Neuladen des Dashboards (persistent) verschwinden.

Kommentare:

1. Modalität: Wie erfolgt genau das Entfernen von Elementen?
   (a) Per Elementmenü/-option oder
   (b) Interaktiv durch das Bewegen in einen Papierkorb. Siehe User Story #010: Papierkorb.
2. Anpassbarkeit: Wie sollen sich andere Elemente beim Entfernen eines Elementes neu anordnen?
3. Widerrufbarkeit:
   (a) Das Entfernen widerrufen können. (Siehe User Story #010: Papierkorb) oder
   (b) Erfolgt das Entfernen nach einer Bestätigung?

User Story #005: Umordnen von Elementen (Vorrang: hoch)

Stakeholder	Auftraggeber (QDM), Nutzer
Wunsch	Umordnen von vorhandenen Elementen
Nutzen	Personalisierbarkeit des Dashboards

Akzeptanzkriterien:

1. Die Positionen bzw. die Anordnung der vorhandenen Elemente können geändert werden.
2. Dashboard passt sich auf die Änderung automatisch an (siehe User Story #006: Reaktionsfähiges Design des Dashboards).
3. Die Änderungen sollen beim Neuladen des Dashboards persistent gelten.

Kommentare:

1. Modalität: Nur durch die Maus oder auch per Tastatur?
   Tastatur-Kürzel als Fleißarbeit
2. Anpassbarkeit: Wie sollen sich andere Elemente beim Bewegen eines umzuordnenden Elements verhalten, sodass die neue Anordnung nach dem Platzieren dem Nutzer klar ist?
3. Implementierung: Sind schon Frameworks/Bibliotheken vorhanden (z.B. Rasterlayout-basierte), die die Anordnung geschickt lösen?

User Story #006: Reaktionsfähiges Design des Dashboards (Vorrang: hoch)

Stakeholder	Auftraggeber (QDM), Nutzer
Wunsch	Dynamische Dimensionierung bzw. Anordnung der Elemente bei Änderungen des Dashboards
Nutzen	Reaktionsfähige bzw. anpassungsfähige Darstellung auf quasi allen Bildschirmgrößen und Geräten

Akzeptanzkriterien:

1. Die Größen bzw. Positionen der Elemente werden bei:
   (a) Größenänderungen Des Browserfensters (z.B. Verlassen des Vollbildmodus),
   (b) Änderung der Elementgröße,
   (c) Skalierung (siehe User Story #013: Skalierung) oder
   (d) Umordnen von Elementen (siehe User Story #005: Umordnen von Elementen)
   automatisch angepasst.
2. Beim Vergrößern des Browserfensters bzw. Verkleinern der Elemente werden passende Elemente jeweils auf eine Zeile zusammenrücken.
3. Beim Verkleinern des Browserfensters bzw. Vergrößern der Elemente nicht passende Elemente sollen vertikal umgebrochen und durch Runterscrollen erreichbar sein.

Kommentare:

1. Modalität:
   (a) Welche Eigenschaft dient als Grundlage für die Größenanpassung der Elemente?
   Metrische Größe (cm/mm) mit Skalierungsmöglichkeit
   (b) Welche Eigenschaft bestimmt die Reihenfolge und die Orientierung/Ausrichtung der Elemente bei einem Umbruch?
2. Implementierung: Sind schon Frameworks/Bibliotheken vorhanden, die die Aufgabe geschickt lösen?
   (a) Wie sollen sich Elemente anpassen, die größer als die neue Fenstergröße sind?
   (b) Wie sollen sich Elemente verschiedenen Plattformen anpassen? (PC, Laptop, Handy, Tablet usw.)

User Story #007: Optimale Darstellung (Vorrang: hoch)

Stakeholder	Auftraggeber (QDM), Nutzer
Wunsch	Optisch und inhaltlich verständliche und einheitliche Darstellung der Elemente
Nutzen	Optimale Erkennbarkeit bzw. Unterscheidbarkeit des Kontexts bei geleichartigen Elementen

Akzeptanzkriterien:

1. Folgende Informationen sollen durch die Schnittstelle abrufbar sein und angezeigt werden (siehe User Story #008: Datenschnittstelle):
   (a) Die zur Verständlichkeit bzw. Identifizierbarkeit der Inhalte nötigen Informationen:
      i. Kurztitel
      ii. Beschreibung
      iii. Parameter (bei Diagrammen/Tabellen)
   (b) Die Informationen zur praktischen Dimensionierung der Elementart:
      i. Seitenverhältnis des Elementes
      ii. Mindestgröße des Elementes
      iii. Schriftgröße des Inhalts
2. Die kleinste Schriftgröße aller vorhandenen Elemente wird für alle Elemente übernommen, um eine einheitliche Schriftgröße zu gewährleisten.

Kommentare:

User Story #008: Datenschnittstelle (Vorrang: hoch)

Stakeholder	Auftraggeber (QDM), Nutzer
Wunsch	Eine REST-basierte Datenschnittstelle
Nutzen	Befüllen von Dashboard-Frames mit Inhalten aus diversen Quellen (z.B. Scout-NextGen)

Akzeptanzkriterien:

1. Frames können einen URL als Datenquelle enthalten (REST-API).
2. Der URL enthält alle nötigen Parameter zum Abruf des Frame-Inhaltes.
3. Auf Abfrage gibt die Datenquelle ein JSON-Objekt in Textformat zurück.
4. Das Objekt enthält:
   (a) Die Daten zur passenden Dimensionierung des Inhalts (siehe User Story #007: Optimale Darstellung).
   (b) Ein JSON-Objekt mit dem strukturierten Inhalt.
   (c) Parameter zur richtigen Interpretation und Aufbereitung des JSON-Inhaltes (z. B. Diagrammtyp, Tabelle, Kennzahl).

Kommentare:

User Story #009: Menü (Vorrang: hoch)

Stakeholder	Nutzer
Wunsch	Ein Menü für das Dashboard bzw. Elemente.
Nutzen	Unterbringung von und Zugriff auf relevante(n) Steuerelemente(n) mit geringem Platzbedarf.

Akzeptanzkriterien:

1. Das Menü ist durch ein Steuerelement oder eine Interaktion (z.B. bei mobilen Endgeräten) einblendbar.
2. Ein Kreisförmiges Menü enthält relevante Optionen für das jeweilige Dashboard/Element.
3. Die Optionen werden nach unten und mittig am Rande des Kreises abgebildet.
4. Beim Überfliegen mit der Maus (oder Tasten auf mobilen Endgeräten) wird die Beschreibung der Option angezeigt.
5. Beim Klicken (oder schieben ins Zentrum des Kreises auf mobilen Endgeräten) wird die Option ausgewählt/betätigt.

Kommentare:

User Story #010: Papierkorb (Vorrang: mittel)

Stakeholder	Nutzer
Wunsch	Papierkorb für gelöschte Elemente.
Nutzen	Widerrufbarkeit von versehentlich gelöschten Elementen.

Akzeptanzkriterien:

1. Gelöschte Elemente werden auf den Papierkorb verschoben (siehe User Story #004: Entfernen von Elementen und User Story #002: Liste der Dashboards).
2. Der Papierkorb ist durch eine Menüoption erreichbar (siehe User Story #009: Menü).
3. Die gelöschten Elemente können wiederhergestellt oder auf andere Dashboards übertragen werden.
4. Die Elemente werden nach einer (evtl. feststellbaren) Aufbewahrungsfrist endgültig gelöscht.

Kommentare:

Der Papierkorb ist ein gesondertes Dashboard, das als Friedhof für gelöschte Elemente dient.

User Story #011: Kategorie (Vorrang: mittel)

Stakeholder	Auftraggeber (QDM), Nutzer
Wunsch	Anlegen von beschrifteten Kategorien unter einem Dashboard.
Nutzen	Inhaltlich zusammenhängende Elemente werden nicht verstreut und sind leicht auffindbar.

Akzeptanzkriterien:

1. Ein Steuerelement zum Erstellen einer neuen Kategorie wird angeboten.
2. Eine neue Kategorie erscheint nach der Erstellung auf dem Dashboard.
3. Eine Kategorie kann umbenannt und mit einer Beschreibung versehen werden.
4. Kategorien können wie Elemente auf dem Dashboard verschoben werden (siehe User Story #005: Umordnen von Elementen).
5. Kategorien können aufgelöst werden (Entbündelung von Elementen).
6. Kategorien können entfernt werden (siehe User Story #010: Papierkorb).

Kommentare:

1. Eine Kategorie ist im Grunde ein Dashboard (Eine in Dashboard verschachtelte Sammlung von Elementen).
2. Jede Kategorie nimmt einen rechteckigen Platz gleicher Bereite und entsprechend ihren Elementen anteiliger Höhe des Dashboards in Anspruch und verwaltet/ordnet die ihr untergeordneten Elemente selbst (verschachteltes Dashboard). Bei nicht gefüllten Kategorien werden die Elemente der nächsten Kategorie optisch auf der nächsten Zeile anfangen.
3. Eine Verschachtelung von Kategorien ist nicht möglich.
4. Alle sich auf dem Dashboard befindlichen Elemente (einschl. Kategorien) werden vom Dashboard geordnet.
5. Elemente können zwecks Auffindbarkeit nach Hinzufügen bzw. Benachrichtigung unter eine künstliche, temporäre Kategorie wandern (siehe User Story #002: Hinzufügen von Elementen und User Story #015: Benachrichtigung).

User Story #012: Gruppe (Vorrang: mittel)

Stakeholder	Nutzer
Wunsch	Gruppierung von Elementen
Nutzen	Verhindert, dass kleine zusammengehörende Elemente wie z.B. Kennzahlen durch einen Umbruch getrennt werden.

Akzeptanzkriterien:

1. Ein Steuerelement zum Erstellen einer neuen Gruppe wird angeboten.
2. Elemente können per Maus in die Gruppe verschoben werden.
3. Die Gruppe verhält sich wie ein einziges größeres Element und wird bei der Anpassung nicht umgebrochen.
4. Die Größe der Gruppe wächst/schrumpft beim Hineinziehen/Herausziehen von Elementen.
5. Die Größe und das Seitenverhältnis einer Gruppe dürfen die des größten Elementes nicht überschreiten.

Kommentare:

User Story #013: Skalierung (Vorrang: mittel)

Stakeholder	Auftraggeber (QDM), Nutzer
Wunsch	Ein Werkzeug zur Skalierung von Dashboard-Elementen.
Nutzen	Anpassung an die Sehstärke und den Abstand des Nutzers vom Bildschirm zwecks optimaler visueller Wahrnehmung von Inhalten.

Akzeptanzkriterien:

1. Ein Schieberegler wird bereitgestellt.
2. Die linke bzw. rechte Seite des Schiebereglers werden mit Symbolen zum Verkleinern (Lupe mit minus) bzw. Vergrößern (Lupe mit plus) gekennzeichnet.
3. Durch das Bewegen des Schiebereglers werden die Große der quadratischen Rasterelementen angepasst.
4. AK 3 hat zur Folge, dass jedes Dashboard-Fenster mit demselben Seitenverhältnis skaliert wird.
5. Anschließend werden die Dashboard-Elemente automatisch umgeordnet (siehe User Story #005: Umordnen von Elementen).
6. Der Skalierung von Schachtelelementen (Kategorie, Gruppe) liegen die skalierten Dimensionen der enthaltenen Elemente zugrunde.

Kommentare:

1. Wie viel betragen die minimale bzw. maximale Größe der Rasterelemente beim Skalieren?

User Story #014: Notizen (Vorrang: niedrig)

Stakeholder	Key-User 1 (QE), Key-User 2 (CE-Elektronik), Nutzer
Wunsch	Notizen evtl. mit Fälligkeitsdatum erstellen.
Nutzen	Verfassen von Markern und Kommentaren.

Akzeptanzkriterien:

1. Ein Steuerelement wird zum hinzufügen von Notizen bereitgestellt.
2. Ein gesonderte vertikale Kategorie für Notizen an der rechten Seite ist durch ein Steuerelement ein- und ausblendbar (siehe User Story #011: Kategorie).
3. Notizen sind bearbeitbar und löschbar.
4. Notizen können mit Dashboard-Elementen verlinkt werden.
5. Notizen können mit einem Fälligkeitsdatum versehen werden, um eine Benachrichtigung auszulösen (siehe User Story #015: Benachrichtigung).

Kommentare:

1. Idee: Beim Klicken auf ein Element evtl. vorhandene Notizen hervorheben.

User Story #015: Benachrichtigung (Vorrang: niedrig)

Stakeholder	Key-User 2 (CE: Elektronik), Key-User 3 (CE: Mechanik), Nutzer
Wunsch	Bei Erfüllung eines festgelegten zeitlichen bzw. logischen Auslösers benachrichtigt werden
Nutzen	• Erinnerung an Merker • Hinweis auf Anomalien • Benachrichtigung bei Tätigkeitsbedarf

Akzeptanzkriterien:

1. Eine Benachrichtigungsschnittstelle akzeptiert Benachrichtigungskriterien:
   (a) Parameter
   (b) Bedingung
   (c) Wert
2. Bei Erfüllung der Bedingung wird eine Benachrichtigung ausgelöst.
3. Eine neue, temporäre Kategorie wird ganz oben platziert (siehe User Story #011: Kategorie).
4. Die auslösenden Elemente wandern unter diese Kategorie.
5. Nach Wahrnehmung der Benachrichtigung wandern sie zurück in die ursprüngliche Position.

Kommentare:

1. Wann gilt die Benachrichtigung als wahrgenommen? (Anklicken, Lesen, Bestätigen?)
2. Sind auch sonstige Benachrichtigungen, die kein vorhandenes Element auf dem Dashboard betreffen, als Menüoption nötig?

User Story #016: PDF-Auszug des Dashboards (Vorrang: niedrig)

Stakeholder	Key-User 1 (QE), Nutzer
Wunsch	Druckversion des Dashboards als PDF-Datei speichern.
Nutzen	Archivieren bzw. Verweis auf einen älteren Stand.

Akzeptanzkriterien:

1. Ein Steuerelement:
   (a) Öffnet ein neues Browser-Fenster,
   (b) erzeugt die Druckversion und
   (c) öffnet das Druckmenü des Webbrowsers.
2. Die Druckeigenschaften (A4, Querformat, Puffer und Ränder) sind voreingestellt.
3. Alle Dashboard-Elemente werden vollständig dargestellt.
4. Auf der Seite nicht-passende Elemente werden (ggf. unter derselben wiederholt dargestellten Kategorie) auf der nächsten Seite fortgesetzt (siehe User Story #011: Kategorie).

Kommentare:

User Story #017: Eingabe-/Suchfeld (Vorrang: niedrig)

Stakeholder	Nutzer
Wunsch	Suche nach einem String.
Nutzen	Schnelle Auffindbarkeit von Auswertungen.

Akzeptanzkriterien:

1. Ein Textfeld zur Eingabe des Suchbegriffs wird bereitgestellt.
2. Die Suche nach einem textuellen Ausdruck im Titel, der Beschreibung oder den weiteren Parametern aller Dashboard-Elemente wird in Echtzeit durchgeführt.
3. Alle Dashboard-Elemente, bei denen der textuelle Ausdruck nicht gefunden wurde, werden ausgeblendet.
4. Ein leeres Suchfeld hat die Anzeige aller der Dashboard-Elemente zur Folge.

Kommentare:

1. Soll das Suchfeld durch ein Steuerelement (z.B. Lupe) oder schon beim Antippen eingeblendet werden?

Hilfsmittel und Urheberrechte

- Werkzeuge:
  - LaTeX
  - LaTeX-Vorlage der OTH-AW
  - Overleaf Online LaTeX-Editor
  - Inkscape: Bearbeitung von Vektorgraphiken
  - Figma: Erstellung von Software-Prototypen
  - Draw.io: Zeichnung von Diagrammen
- Hilfreiche Pakete + Erweiterungen (u. a.):
  - BibLaTeX + Biber: LaTeX-Pakete für die Bibliographie
  - glossaries-extra + bib2gls: LaTeX-Pakete für Abkürzungsverzeichnis/Glossar.
- Anleitungen und Foren für LaTeX:
  - LaTeX fürs Studium Kompaktkurs (Moodle-Kurs) [58] [1]
  - LaTeX: Einführung in das Textsatzsystem [59] [2]
  - CTAN: Dokumentation-Archiv für TeX-Pakete
  - OverLeaf Dokumentation
  - TeX-Forum von StackExcahnge
- Literaturrecherche:
  - ResearchGate
  - Semantic Scholar
  - Google Scholar
  - Google mit *filetype:pdf* Filter [3]

[1] Nur für Mitglieder der OTH-AW zugänglich!
[2] Bestellung nur über teilnehmende Hochschulen!
[3] Immer auf Copyright achten! Urheberrechtlich geschützte Bücher werden auch ohne Erlaubnis hochgeladen.

Graphik	Urheberrecht
	© 2024 Oracle®
	© Cornelsen Verlag GmbH, 2024

Quellcodeverzeichnis

2.1 Beispiel: HTML-Code 10
2.2 Beispiel: HTML-Code mit CSS 12
2.3 Historischer JS-Code zur Gewährleistung der DOM-Kompatibilität . . . 15
2.4 Beispiel: JS-Code zur Änderung einer Webseite 16
4.1 Der Beispiel-Aufbau des Dashboard-Konstruktors 38
4.2 Beispiel: Hierarchisches Datenmodell 41
4.3 Beispiel: Abgeflachtes Datenmodell 42
4.4 Beispiel: Datenmodell für die Dashboard-Konfiguration 44

Abbildungsverzeichnis

1.1 Zeitliche Entwicklung der Fertigungsqualität in EWA 2

2.1 Übersicht der Grundbegriffe 7
2.2 Die erste HTML-Webseite der Welt 8
2.3 Beispiel: HTML-Seite 11
2.4 Beispiel: HTML-Seite mit CSS-Formatierung 13
2.5 Beispiel: Baumstruktur eines DOM 16
2.6 Beispiel: Mit JavaScript® manipulierte HTML-Seite 17
2.7 Beispiel: CSS Grid Layout 19

3.1 Übersicht der Stakeholder 22
3.2 Wireframe: Dashboard 25
3.3 Wireframe: Frames 25
3.4 Wireframe: ausgerichtete Anordnung 26
3.5 Wireframe: Gruppe 27
3.6 Wireframe: Kategorie 28
3.7 Wireframe: Sort vs. Swap 30

4.1 3-Schichten-Architektur 33
4.2 ER-Diagramm 35
4.3 Klassendiagramm 37
4.4 Klassendiagramm – Datentypen 39

6.1 Abhängigkeiten der User Stories 50

Tabellenverzeichnis

6.1 User Story #001: Anlegen von Dashboards 51
6.2 User Story #002: Liste der Dashboards 52
6.3 User Story #003: Hinzufügen von Elementen 53
6.4 User Story #004: Entfernen von Elementen 54
6.5 User Story #005: Umordnen von Elementen 55
6.6 User Story #006: Reaktionsfähiges Design des Dashboards 56
6.7 User Story #007: Optimale Darstellung 57
6.8 User Story #008: Datenschnittstelle 58
6.9 User Story #009: Menü 59
6.10 User Story #010: Papierkorb 60
6.11 User Story #011: Kategorie 61
6.12 User Story #012: Gruppe 62
6.13 User Story #013: Skalierung 63
6.14 User Story #014: Notizen 64
6.15 User Story #015: Benachrichtigung 65
6.16 User Story #016: PDF-Auszug des Dashboards 66
6.17 User Story #017: Eingabe-/Suchfeld 67
6.18 Übersicht der urheberrechtlich geschützten Graphiken 69

Abkürzungsverzeichnis

Bezeichnung	Beschreibung
AI	Artificial Intelligence
AJAX	Asynchronous JavaScript And XML
APEX®	Application Express®
CE	Component Engineer
CERN	Conseil Européen pour la Recherche Nucléaire (franz.)
CGI	Common Gateway Interface
CPS	cyber-physische Systeme
CSS	Cascading Style Sheets
DI FA MF QM	Digital Industries, Factory Automation, Manufacturing, Quality Management
DIN	Deutsches Institut für Normung e. V.
DOM	Document Object Model
DoS	Denial of Service
dpm–A	Defekte Pro Million – Anschlüsse
DPMO	Defect Per Million Opportunities
DTD	Document Type Descriptor
ECMA	European Computer Manufacturers Association
EDV	elektronische Datenverarbeitung
ER	Entity Relationship
EWA	Elektronikwerk Amberg
EWF	Elektronikwerk Fürth
EWS	Elektronikwerk Singapur
GenAI	Generative Artificial Intelligence
HMI	Human Machine Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IANA	Internet Assigned Numbers Authority

Bezeichnung	Beschreibung
IBM®	International Business Machines®
IEC	International Electrotechnical Commission
IETF	Internet Engineering Task Force
ISO	International Oraganization for Standardization
IT	Information Technology
JS	JavaScript®
JSON	JavaScript® Object Notation
KI	künstliche Intelligenz
KPI	Key Performance Indicator
KVP	kontinuierlicher Verbesserungsprozess
MDN	Mozilla Developer Network
MIME	Multipurpose Internet Mail Extensions
o. J.	ohne Jahr
OMG®	Object Management Group®
ORDS	Oracle® REST Data Services
PDF	Portable Document Format
PL/SQL	Procedural Language / Structured Query Language
QDM	Quality Data Management
QE	Quality Engineer
QM	Quality Management
REST	Representational State Transfer
RFC	Request For Comments
SEWC	Siemens Elektronikwerk Chengdu
SGML	Standard Generalized Markup Language
SMD	Surface Mounted Device
SMT	Surface Mounted Technology
SPS	speicher-programmierbare Steuerung
SQL	Structured Query Language
SVG	Scalable Vector Graphics
TS	Type Script
UI	User Interface
UML®	Unified Modeling Language®
URL	Uniform Resource Locator
UX	User Experience

Bezeichnung	Beschreibung
W3C	World Wide Web Consortium
WEF	World Economy Forum
WHATWG	Web Hypertext Applications Technology Working Group
WWW	World Wide Web
WYSIWYG	What You See Is What You Get
XML	Extensive Markup Language
XSS	Cross-Site Scripting

Glossar

Bezeichnung	Beschreibung
AJAX	Asynchronous JavaScript And XML ist eine Methode für Webanwendungen zum asynchronen Datenabruf von einem Server in JSON oder XML Format. Unter asynchron ist zu verstehen, dass die Webanwendung das Laden der neuen Version der Webseite nicht abwarten muss, da die aktualisierten Daten im Hintergrund abgerufen werden, während die Webseite angezeigt wird.
CGI	Common Gateway Interface ist eine einheiteliche Schnittstelle zum Abrufen von Webseiten, die zum Zeitpunkt des Abrufs noch nicht existieren und z. B. von einer Datenbankprozedur erst generiert werden müssen.
cyber-physische Systeme	Informationstechnische vernetzte Systeme, die durch Sensorik bzw. Informationsaustausch mit anderen cyber-physischen Systemen die Umwelt wahrnehmen und durch Aktorik entsprechend mit der Umwelt interagieren.
Dashboard	„Computerprogramm, das **relevante** Informationen **zusammenfasst** und **übersichtlich** darstellt.“ DUDEN
Industrie-4.0	Die vierte industrielle Revolution, die durch Einsatz von vernetzten cyber-physischen Systemen (CPS) flexible, datengetriebene und kundenorientierte Fertigung ermöglicht.
Key-User	(dt. Schlüsselanwender) Erfahrener Anwender mit Überblick im eigenen Arbeitsumfeld

Bezeichnung	Beschreibung
Metasprache	Metasprachen sind Sprachen, die zur syntaktischen und semantischen Beschreibung bzw. Interpretation anderer Sprachen verwendet werden. DUDEN
PL/SQL	PL/SQL ist eine prozedurale Sprache, die von Oracle® speziell zur Einbettung von SQL-Anweisungen in serverseitige Datenbank-Prozeduren entwickelt wurde. ORACLE
Qualitätsdaten	„Daten über die Qualität von Einheiten, über die bei der Ermittlung der Qualität angewendeten Qualitätsprüfungen und über die dabei herrschenden Randbedingungen sowie ggf. über die jeweiligen Einzelforderungen an die Qualitätsmerkmale.“[8, S. 12]
RFC	Request For Comments (RFC) – auf Deutsch „Bitte um Stellungnahme/Kommentare“ – ist eine online Plattform zur Einreichung technischer Entwürfe zu Internet-Technologien (engl. drafts) zwecks Begutachtung durch andere Experten. Die Entwürfe werden evtl. nach mehreren Überarbeitungen als Spezifikationen zugelassen. RFC wird von Elitegruppen wie Internet Engineering Task Force (IETF) geführt.
schlanke Fertigung	(Engl. lean production/manufacturing) Die aus Japan stammende effizienzorientierte Fertigungsphilosophie, die nach Beseitigung von Verschwendung in vielerlei Hinsicht (zeitlich, logistisch etc.) **strebt**. [6, vgl. S. 10 u. 40]
SQL	Structured Query Language ist eine Syntax zum Ausführen von Datenbanktransaktionen wie Eintragen oder Abfragen von Informationen.
Stakeholder	Als Stakeholder (dt. Interessenvertreter) bezeichnet man in Projektmanagement anspruchsberechtigte Personen und Organisationen, deren Interessen bei der Entwicklung und den Eisatz eines Produktes berücksichtigt werden sollen. [43, S. 455 u. 599]
Unikatisierung	Versehen von Bauteilen mit Unikaten (:= Eindeutigen Identifikationsmerkmalen bzw. Kennzeichen) vor der Fertigung zwecks Rückverfolgbarkeit.
User Story	Eine User Story (dt. Benutzergeschichte) bezeichnet die natürlichsprachlich formulierte Anforderung aus Benutzersicht bei agilen Projekten.

Bezeichnung	Beschreibung
WYSIWYG	What You See Is What You Get (dt. „Was du siehst, ist das, was du bekommst“) ist ein Begriff zur Beschreibung von Bearbeitungsumgebungen, in denen die visuelle Darstellung des Inhalts unmittelbar nach jedem Bearbeitungsschritt dem eigentlichen, resultierenden Ausgangszustand entspricht. D. h., es besteht keinen Schritt (z. B. Compilier-Vorgang) zwischen der Bearbeitung und der Darstellung. Beispiel hierfür sind interaktive Webseiten.

Literatur

[1] Siemens Presse. „Fakten Elektronikwerk Amberg“. (23. Feb. 2015), Adresse: https://assets.new.siemens.com/siemens/assets/api/uuid:4a2a2d26-731a-42e0-a53d-a1501be80ad7/factsheet-amberg-de.pdf (besucht am 17. 01. 2024).

[2] Siemens Presse. „Bundeskanzlerin Merkel besucht Siemens-Vorzeigewerk der "Digitalen Fabrik"“. (23. Feb. 2015), Adresse: https://press.siemens.com/global/de/event/bundeskanzlerin-merkel-besucht-siemens-vorzeigewerk-der-digitalen-fabrik (besucht am 17. 01. 2024).

[3] Siemens Presse. „Siemens Elektronikwerk Amberg mit dem „Industrie 4.0-Award“ ausgezeichnet“. (5. Dez. 2018), Adresse: https://press.siemens.com/de/de/pressemitteilung/siemens-elektronikwerk-amberg-mit-dem-industrie-40-award-ausgezeichnet (besucht am 17. 01. 2024).

[4] Siemens Presse. „Siemens Elektronikwerk Amberg als digitale Leuchtturmfabrik benannt“. (31. März 2021), Adresse: https://press.siemens.com/de/de/pressemitteilung/siemens-elektronikwerk-amberg-als-digitale-leuchtturmfabrik-benannt (besucht am 17. 01. 2024).

[5] Siemens Presse. „Siemens Elektronikwerk Amberg – Die Digitale Fabrik“. (Feb. 2015), Adresse: https://assets.new.siemens.com/siemens/assets/api/uuid:8aa8837d-75ca-47fb-805e-96ff79531b4a/praesentation-de.pdf (besucht am 17. 01. 2024).

[6] R. Shah und P. Ward, „Defining and Developing Measures of Lean Production“, *Journal of Operations Management 25*, Jg. 25, Nr. 4, S. 785–805, Juni 2007. DOI: 10.1016/j.jom.2007.01.019. Adresse: https://www.researchgate.net/publication/222434298_Defining_and_Developing_Measures_of_Lean_Production.

[7] Deutsches Forschungszentrum für künstliche Intelligenz (DFKI). „Die vier Stufen der industriellen Revolution“. (2011), Adresse: https://www.dfki.de/fileadmin/_processed_/6/3/csm_vier-stufen-industrielle-revolution-dfki-de-web_b4b923f724.jpg (besucht am 18. 01. 2024).

[8] W. Geiger, „Terminus Technicus: Qualitätsdaten“, *QZ Qualität und Zuverlässigkeit 52*, Nr. 11, 2007.

[9] T. Berners-Lee. „Longer Biography“. Version 1.134, W3C. (13. Juni 2023), Adresse: https://www.w3.org/People/Berners-Lee/Longer.html (besucht am 17. 05. 2024).

[10] „The birth of the Web“, CERN. (o. J.), Adresse: https://home.web.cern.ch/science/computing/birth-web (besucht am 17. 05. 2024).

[11] T. Berners-Lee und D. W. Connolly, *Hypertext Markup Language - 2.0*, RFC 1866, Nov. 1995. DOI: 10.17487/RFC1866. Adresse: https://datatracker.ietf.org/doc/html/rfc1866.

[12] *Information technology - Document description and processing languages - Hypertext Markup Language (HTML)*, 1. Mai 2000. Adresse: https://www.iso.org/standard/27688.html.

[13] „W3C Wiki“, WHATWG. (28. Mai 2019), Adresse: https://wiki.whatwg.org/wiki/W3C#WHATWG_is_formed (besucht am 17. 05. 2024).

[14] „HTML Standard“, W3C. (o. J.), Adresse: https://www.w3.org/html/ (besucht am 17. 05. 2024).

[15] „W3C and WHATWG to work together to advance the open Web platform“, W3C. (28. Mai 2019), Adresse: https://www.w3.org/blog/2019/w3c-and-whatwg-to-work-together-to-advance-the-open-web-platform/ (besucht am 17. 05. 2024).

[16] „Memorandum of Understanding Between W3C and WHATWG“, W3C. (28. Mai 2019), Adresse: https://www.w3.org/2019/04/WHATWG-W3C-MOU.html (besucht am 17. 05. 2024).

[17] „HTML5 Differences from HTML4“, W3C. (Dez. 2014), Adresse: https://www.w3.org/TR/html5-diff/ (besucht am 17. 05. 2024).

[18] „HTML Living Standard“, WHATWG. (o. J.), Adresse: https://html.spec.whatwg.org/ (besucht am 17. 05. 2024).

[19] „Introduction to SGML for the W3C Markup Validator“, W3C. (o. J.), Adresse: https://validator.w3.org/docs/sgml.html (besucht am 18. 05. 2024).

[20] „A brief SGML tutorial“, W3C. (o. J.), Adresse: https://www.w3.org/TR/WD-html40-970708/intro/sgmltut.html (besucht am 20. 05. 2024).

[21] B. Bos. „A brief history of CSS until 2016“, W3C. (17. Dez. 2016), Adresse: https://www.w3.org/Style/CSS20/history.html (besucht am 19. 05. 2024).

[22] „MIME Types“, MDN. (o. J.), Adresse: https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types#textcss (besucht am 19. 05. 2024).

[23] „Media Types“, IANA. (o. J.), Adresse: https://www.iana.org/assignments/media-types/media-types.xhtml (besucht am 19. 05. 2024).

[24] „Selectors Level 3“, W3C. (6. Nov. 2018), Adresse: https://www.w3.org/TR/2018/REC-selectors-3-20181106/#selectors (besucht am 20. 05. 2024).

[25] „CSS: !important“, MDN. (o. J.), Adresse: https://developer.mozilla.org/en-US/docs/Web/CSS/important (besucht am 19. 05. 2024).

[26] A. Rauschmayer, *Speaking JavaScript®*. O'Reilly Media, Inc., Sep. 2014, ISBN: 9781449364991. Adresse: https://exploringjs.com/es5/.

[27] N. Hamilton. „The A-Z of Programming Languages: JavaScript® (archived)", Computerworld. (31. Juli 2008), Adresse: https://web.archive.org/web/20190908212626/https://www.computerworld.com.au/article/255293/a-z_programming_languages_javascript/ (besucht am 20. 05. 2024).

[28] *ECMAScript® 2023 language specification*, 14th Edition, ECMA-262, ECMA, 14. Juni 2023. Adresse: https://ecma-international.org/publications-and-standards/standards/ecma-262/.

[29] S. M. Schafer, *Web Standards Programmer's Reference: HTML, CSS, JavaScript®, Perl, Python®, and PHP*. Wrox, Aug. 2005, ISBN: 9780764588204. Adresse: https://www.oreilly.com/library/view/web-standards-programmers/9780764588204/.

[30] P.-P. Koch. „Level 0 DOM". (o. J.), Adresse: https://www.quirksmode.org/js/dom0.html (besucht am 22. 05. 2024).

[31] P. Krill. „JavaScript creator ponders past, future", IANA. (23. Juni 2008), Adresse: https://www.infoworld.com/article/2653798/javascript-creator-ponders-past-future.html (besucht am 20. 05. 2024).

[32] M. Andreesen und B. Eich. „INNOVATORS OF THE NET: BRENDAN EICH AND JAVASCRIPT (archiviert)", Netscape. (24. Juni 1998), Adresse: https://web.archive.org/web/20021003142542/http://wp.netscape.com/comprod/columns/techvision/innovators_be.html (besucht am 20. 05. 2024).

[33] „Document Object Model Specification", W3C. (9. Okt. 1997), Adresse: https://www.w3.org/TR/WD-DOM-971009/ (besucht am 19. 05. 2024).

[34] „Live DOM Viewer". (o. J.), Adresse: https://software.hixie.ch/utilities/js/live-dom-viewer/ (besucht am 22. 05. 2024).

[35] *The JSON Data Interchange Syntax*, 2nd Edition, ECMA-404, ECMA, Dez. 2017. Adresse: https://ecma-international.org/publications-and-standards/standards/ecma-404/.

[36] „What is JSON Schema?" (o. J.), Adresse: https://json-schema.org/overview/what-is-jsonschema (besucht am 26. 05. 2024).

[37] R. T. Fielding, „Architectural Styles and the Design of Network-based Software Architectures", Dissertation, University of California, Irvine, 2000. Adresse: https://ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf (besucht am 26. 05. 2024).

[38] E. Marcotte. „Responsive Web Design", A List Apart. (25. Mai 2010), Adresse: https://alistapart.com/article/responsive-web-design/ (besucht am 26. 05. 2024).

[39] „Media Queries Level 3", W3C. (21. Mai 2024), Adresse: https://www.w3.org/TR/mediaqueries-3/ (besucht am 26. 05. 2024).

[40] „Responsive layout grid", Google®. (o. J.), Adresse: https://m2.material.io/design/layout/responsive-layout-grid.html (besucht am 26. 05. 2024).

[41] „Grid Layout“, W3C. (7. Apr. 2011), Adresse: https://www.w3.org/TR/2011/WD-css3-grid-layout-20110407/ (besucht am 26. 05. 2024).

[42] „CSS Grid Layout Module Level 2“, W3C. (18. Dez. 2020), Adresse: https://www.w3.org/TR/css-grid-2/ (besucht am 26. 05. 2024).

[43] H. Balzert, *Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements-Engineering*, 3. Aufl. Spektrum Akademischer Verlag, 2009, ISBN: 978-3-8274-1705-3. Adresse: http://www.spektrum-verlag.de/978-3-8274-1705-3.

[44] *Muuri*. Adresse: https://muuri.dev (besucht am 31. 05. 2024).

[45] *SortableJS*. Adresse: https://sortablejs.github.io/Sortable/ (besucht am 31. 05. 2024).

[46] *Gridstack.js*. Adresse: https://gridstackjs.com (besucht am 31. 05. 2024).

[47] OMG®, *UML®*, Version 2.5.1, 5. Dez. 2017. Adresse: https://www.omg.org/spec/UML/2.5.1/PDF (besucht am 25. 04. 2024).

[48] H. Balzert, *Lehrbuch der Softwaretechnik: Entwurf, Implementierung, Installation und Betrieb*, 3. Aufl. Spektrum Akademischer Verlag, 2011, ISBN: 978-3-8274-1706-0. Adresse: www.spektrum-verlag.de/978-3-8274-1706-0.

[49] IBM®. „Was ist eine dreischichtige Architektur?“ (o. J.), Adresse: https://www.ibm.com/de-de/topics/three-tier-architecture#Die+drei+Schichten+im+Detail (besucht am 22. 04. 2024).

[50] IBM®. „What is three-tier architecture?“ (o. J.), Adresse: https://www.ibm.com/topics/three-tier-architecture#The+three+tiers+in+detail (besucht am 22. 04. 2024).

[51] Microsoft®Learn. „Was ist eine n-schichtige Architektur?“ (o. J.), Adresse: https://learn.microsoft.com/de-de/training/modules/n-tier-architecture/2-what-is-n-tier-architecture (besucht am 30. 04. 2024).

[52] Microsoft®Learn. „What is an N-tier architecture?“ (o. J.), Adresse: https://learn.microsoft.com/en-us/training/modules/n-tier-architecture/2-what-is-n-tier-architecture (besucht am 30. 04. 2024).

[53] *JSON Developer's Guide, 23ai*, F46733-03, Oracle®, 2024-05. Adresse: https://docs.oracle.com/en/database/oracle//oracle-database/23/adjsn/json-developers-guide.pdf.

[54] „What is prototype pollution?“, PortSwigger. (2019-09), Adresse: https://portswigger.net/web-security/prototype-pollution (besucht am 30. 05. 2024).

[55] „Prototype pollution“, SynkLearn. (o. J.), Adresse: https://learn.snyk.io/lesson/prototype-pollution/ (besucht am 30. 05. 2024).

[56] „Object: Unevaluated Properties“. (2019-09), Adresse: https://json-schema.org/understanding-json-schema/reference/object#unevaluatedproperties (besucht am 30. 05. 2024).

[57] „JSON Sanitizer“, OWASP. (o. J.), Adresse: https://github.com/OWASP/json-sanitizer (besucht am 30. 05. 2024).

[58] A. Aßmuth, *LaTeX fürs Studium Kompaktkurs*, OTH-AW Moodle-Kurs, Feb. 2014. Adresse: https://moodle.oth-aw.de/course/view.php?id=18.

[59] T. Sturm, *LaTeX: Einführung in das Textsatzsystem*, 11. Aufl. München: Institut für Mathematik und Informatik der Universität Bundeswehr, Aug. 2016. Adresse: https://www.luis.uni-hannover.de/de/services/kurse-beratung-und-support/handbuecher/it-handbuecher-und-ebooks/details/manuals/latex/.