

# Bericht über das Praktische Studiensemester

vom 01.03.2022 bis 31.08.2022

bei SEITENBAU

Abteilung Software Development  
and Services

**Praktikant/in:** Jannis Liebscher

Matr.-Nr.: 301645

Studiengang: AIN

Firmenbetreuer/in: Developer,  
Kathrin Wolpers

Datum der Abgabe: 17.10.2022

# Inhalt

Abkürzungsverzeichnis .....	3
Abbildungsverzeichnis.....	4
1    Einleitung .....	5
2    Der Betrieb .....	6
3    Arbeitsweise .....	7
4    Bearbeitete Aufgaben / Projekte .....	8
4.1 Aufgabe 1 .....	8
4.1.1 Problembeschreibung.....	8
4.1.2 Aufgabenstellung und Zielsetzung.....	8
4.1.3 Theoretische Grundlagen .....	8
4.1.4 Realisierung.....	9
4.1.5 Nachweis/Bewertung der Funktionsfähigkeit .....	11
4.1.6 Zusammenfassung .....	11
4.2 Aufgabe 2.....	12
4.2.1 Aufgabenbeschreibung und Zielsetzung.....	12
4.2.2 Theoretische Grundlagen .....	12
4.2.3 Realisierung.....	13
4.2.4 Nachweis/Bewertung der Funktionsfähigkeit .....	15
4.3 Aufgabe 3.....	16
4.3.1 Aufgabenbeschreibung und Zielsetzung.....	16
4.3.2 Theoretische Grundlage .....	17
4.3.3 Realisierung.....	17
4.3.4 Nachweis/Bewertung der Funktionsfähigkeit .....	19
5    Persönliche Eindrücke und Erfahrungen .....	20

# Abkürzungsverzeichnis

<i>PKP</i>	-	<i>Gemeinsames Planungs- und Kabinettmanagement Programm</i>
<i>IAM</i>	-	<i>Identity and Access Management</i>
<i>DokV</i>	-	<i>Dokumentenverwaltung</i>
<i>JVM</i>	-	<i>Java Virtual Machine</i>
<i>VM</i>	-	<i>Virtuelle Maschine</i>
<i>PAC</i>	-	<i>PDF Accessibility Checker</i>
<i>WAR</i>	-	<i>Web Application Archive</i>
<i>CI</i>	-	<i>Continious Integration</i>
<i>http</i>	-	<i>Hypertext Transfer Protocol</i>
<i>css</i>	-	<i>Cascading Style Sheets</i>
<i>svg</i>	-	<i>Scalable Vector Graphic</i>
<i>VRT</i>	-	<i>Visual Regression Tests</i>

# Abbildungsverzeichnis

Abb. 1	Beispielhafter Fehlerbericht von PAC 2021 .....	5
Abb. 2	Das fox:alt Attribut in einem Grafik Element des Stylesheets .....	6
Abb. 3	Ausschnitt der Fonts-Sektion der Apache FOP Konfigurationsdatei ...	6
Abb. 4	Die Tabelle der Mantelbogen PDF .....	7
Abb. 5	Die Tabelle der Mantelbogen PDF als Screenreader Ansicht in PAC3 .. .....	7
Abb. 6	Ausschnitt aus der Datei docker-compose.yaml .....	9
Abb. 7	Ausschnitt aus docker-compose.yaml, Konfiguration der Volumes von DokV-informationsdienst.....	10
Abb. 8	Ausschnitt aus docker-compose.yaml, Port Konfiguration des Proxys .....	11
Abb. 9	Das alte Layout der PKP Startseite .....	12
Abb. 10	Das neue Layout der PKP Startseite .....	12
Abb. 11	Thymeleaf Fragment, welches einen Quicklink darstellt .....	13
Abb. 12	Ausschnitt aus der SVG Datei (Links) mit den Kommandos die zu dem Pfeil Icon (Rechts) führen .....	13

# 1 Einleitung

Der vorliegende Bericht beschäftigt sich mit meiner Arbeit an zwei Software Projekten im Rahmen meines Praxissemesters. Bei beiden handelt es sich um webbasierte Verwaltungsprogramme im Rahmen der Dienstekonsolidierung der Bundesregierung. In beiden kommt Java als (Haupt-)Programmiersprache, sowie Spring(Boot) als grundlegendes Framework zum Einsatz.

Bei dem Programm „Dokumentenverwaltung“ (DokV) handelt es sich um eine Anwendung, die bereits seit über 10 Jahren im Einsatz ist. Hier werden hauptsächlich Bugfixes, Security Patches und kleinere Anpassungen vorgenommen. DokV besteht aus zwei Komponenten. Bei der ersten Komponente handelt es sich um die elektronische Verteilung von sog. Drucksachen. Die Zweite dient der elektronischen Einbringung von Dokumenten.

PKP wird aktiv entwickelt und umfasst eine Vielzahl von einzelnen Modulen, die v.a. von Mitarbeitern der Ministerien verwendet werden. Unter anderem können Vorhaben eingebracht und an die entsprechenden Stellen weitergeleitet werden. Des Weiteren können Fragen/Anfragen in elektronischer Form gestellt und beantwortet werden. Der Politische Kalender ermöglicht eine Übersicht über Tagesordnungspunkte wie z.B. Sitzungen, während es die eMappe den Abgeordneten ermöglicht, ihre Unterlagen in elektronischer Form mitzuführen und so während einer Sitzung Zugriff auf diese zu haben, ohne große Aktenordner mitzunehmen. Dazu kommt eine umfassende Verwaltung in der Systemadministratoren die Fraktionen, Berechtigungen und zum Teil auch einzelne Benutzer anpassen können.

Eng mit PKP verbunden ist das Identity and Access Management (IAM), in welches die Benutzerverwaltung ausgelagert wurde, um es zu ermöglichen sich mit einem einzigen Account nicht nur bei PKP sondern bei allen an das IAM angebundenen Systemen anzumelden.

## 2 Der Betrieb

SEITENBAU wurde 1996 von Stefan Eichenhofer, Jan Bauer und Florian Leinberger gegründet. Mittlerweile zählt das Unternehmen über 200 Mitarbeiter, die hauptsächlich am Hauptsitz in Konstanz arbeiten. Eine kleinere Niederlassung gibt es außerdem in Köln. SEITENBAU entwickeln maßgeschneiderte Software für seine Kunden, zu denen vor allem einige Bundesämter, aber auch andere Unternehmen, wie z.B. die Telekom gehören<sup>1</sup>.

Entsprechend des agilen Mindsets gibt es im Unternehmen nur flache Hierarchien und die einzelnen Teams organisieren sich selbstverantwortlich für ihre Projekte. Auch in den Teams sind alle Mitglieder gleichberechtigt, Entscheidungen werden nach gemeinsamer Beratung zusammen getroffen.

Ich wurde dem Team zugeteilt, dass sich um die Projekte DokV und PKP kümmert. Bei meinem Eintritt bestand das Team aus vier Entwicklern, sowie einem Scrum Master, einem Technical Consultant und der Projektleiterin. Drei Monate nach Beginn meines PSS wurde ein weiterer Entwickler eingestellt, wodurch das Team auf neun Mitarbeiter anwuchs.

Das Team kümmert sich um alles rund um die beiden o.g. Projekte. Die Entwicklung beinhaltet das Front- und Backend, das Testen und Ausliefern neuer Versionen sowie technischen Support rund um die Anwendungen. Hinzu kommen Verwaltungsaufgaben wie die Ausschreibung, um das Projekt nach Vertragsende weiter zu führen, oder die Rechnungsstellung.

---

<sup>1</sup> <https://www.seitenbau.com/kunden>

### 3 Arbeitsweise

Unser Team arbeitet nach dem Scrum-Prinzip. Dazu gehören regelmäßige (Online) Meetings mit den Kunden, um den Projektfortschritt zu präsentieren sowie Feedback einzuholen. Zudem rekapitulieren wir die zweiwöchigen Sprints regelmäßig in einer Retrospektive. Im Rahmen des sog. „Refinements“ wird der Arbeitsaufwand für neue, vom Kunden gewünschte Features abgeschätzt. Diese Umsetzungsaufgaben mit allen weiteren Maßnahmen wie z.B dem Testen der neuen Funktionen, werden als Story bezeichnet.

Im Sprint selbst werden diese Stories dann nochmal in einzelne Tickets unterteilt, d.h. es werden kleinere Schritte definiert, die im Idealfall parallel von mehreren Entwicklern bearbeitet werden können. Nach diesem „Task Breakdown“ kann sich jeder selbstständig als Bearbeiter eines Tickets eintragen und mit dessen Umsetzung beginnen. Dies geschieht im Issue Tracker Jira, in welchem unter anderem die Stories des aktuellen Sprints einsehbar sind. So bekommt man einen Überblick darüber wie weit einzelne Stories umgesetzt sind und wer gerade an was arbeitet. Dabei halten wir laufend, mindestens einmal täglich im Daily Scrum, Rücksprache mit den anderen Teammitgliedern. Das hilft zum einen dabei einen einheitlichen verständlichen Stil einzuhalten. Zum anderen gibt es aber auch neuen Mitgliedern die Möglichkeit Fragen zu stellen und erleichtert so die Einarbeitung. Gerade am Anfang aber auch bei neuen Themengebieten wird zusätzlich Pair-Programming eingesetzt. Diese Methode funktioniert sowohl vor Ort im Büro, als auch remote mittels Bildschirmübertragung sehr gut.

Zur Versionsverwaltung kommt Git in Verbindung mit Bitbucket als remote Repository zum Einsatz. Letzteres befähigt uns in Verbindung mit Jira z.B. einheitliche Git-Banches für einzelne Stories zu erstellen. Alle drei Systeme zusammen spielen außerdem im Review eine große Rolle. Dabei wird jedes abgeschlossene Ticket von einem anderen Teammitglied „überprüft“. Das beinhaltet vor allem ein Code-Review, sowie, wenn angemessen, einen kurzen Test, ob die Anwendung weiterhin kompiliert und startet. Das stellt zum einen eine hohe Code Qualität sicher, zum anderen nimmt es Neulingen die Angst etwas falsch zu machen.

## 4 Bearbeitete Aufgaben / Projekte

### 4.1 Aufgabe 1

Generierung von barrierefreien PDF-Dateien

#### 4.1.1 Problembeschreibung

DokV generiert drei verschiedene Typen von PDF Dateien. Das wären zum einen einfache Unterschriftenblätter. Dazu kommen sog. Mantelbögen, welche Informationen zur Verfügbarkeit einer Drucksache enthalten. Zuletzt gibt es eine Drucksachenmappe, welche an sich nur ein Deckblatt mit Inhaltsverzeichnis darstellt, welchem die eigentlichen Drucksachen angefügt werden. Alle drei Dateien sind für sehbehinderte Menschen teilweise nicht, oder nur schwer lesbar, da sie nicht "getagged" sind. Diese „Tags“ enthalten dabei zusätzliche Informationen über die Struktur des Dokuments, während das Aussehen unverändert bleibt.

#### 4.1.2 Aufgabenstellung und Zielsetzung

Ziel dieser Story ist es, dass die von DokV generierten PDF-Dateien im PDF Accessibility Checker (PAC) keine Fehler enthalten. Außerdem soll die Bibliothek Apache FOP, die zur PDF-Generierung verwendet wird, auf die aktuelle Version aktualisiert werden.

#### 4.1.3 Theoretische Grundlagen

Als Build-Tool wird Maven eingesetzt. Das Aktualisieren der Bibliothek Apache FOP geschieht somit durch das Anpassen der Versionsnummer in der Datei pom.xml. Diese enthält alle für den Bau des Projektes notwendigen Abhängigkeiten. Maven lädt diese in das lokale Repository und nimmt sie beim Bau in die WAR-Datei auf, ein spezielles Java Archiv (JAR) das darauf ausgelegt ist, als Webanwendung gestartet zu werden. Im Anschluss muss geprüft werden, ob und falls ja welche Codeanpassungen vorgenommen werden müssen. Erst wenn die Anwendung wieder startet und optisch identische PDF-Dateien erstellt werden, wird damit begonnen die eigentliche PDF-Generierung anzupassen.

Die Barrierefreiheit wird mit dem PDF Accessibility Checker (PAC) überprüft. Dieser zeigt Fehler an, die behoben werden müssen, damit eine PDF als barrierefrei gilt. Ein Beispiel für einen solchen Bericht ist in Abbildung 1 zu sehen. Wichtig für die Barrierefreiheit sind vor allem ein Strukturbaum, an dem sich Screenreader



orientieren können, sowie Alternativtexte für Abbildungen. Es müssen aber alle der angezeigten Fehler behoben werden.

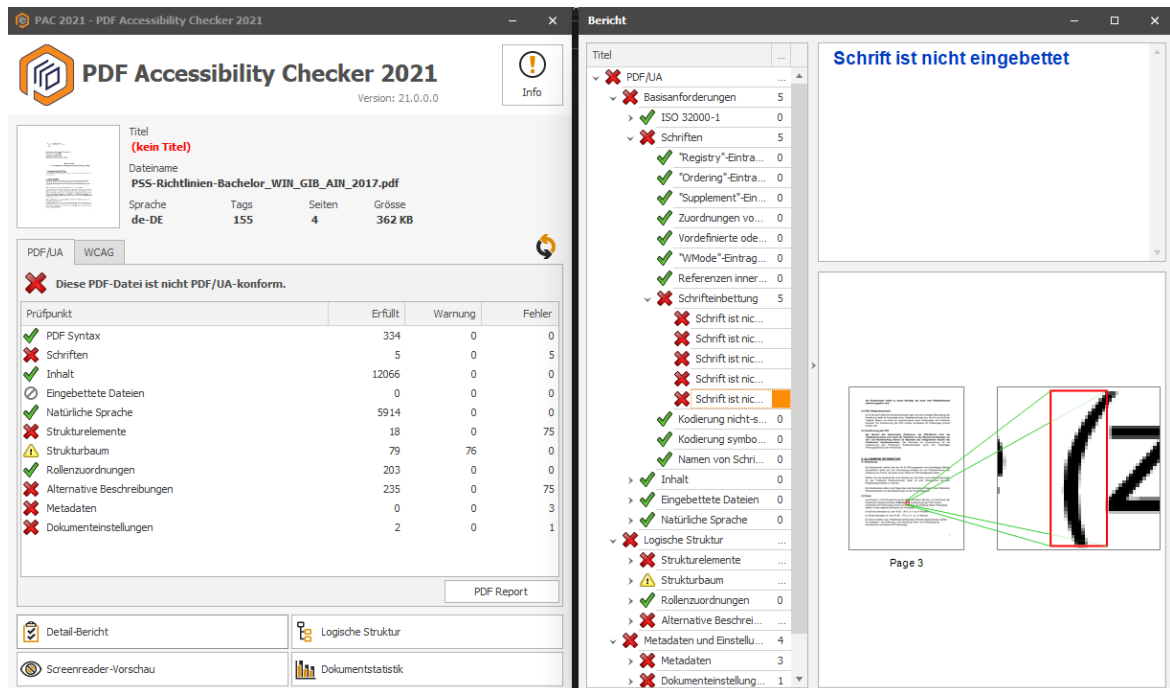


Abbildung 2 Beispielhafter Fehlerbericht von PAC 2021

Für die PDF-Generierung benötigt Apache FOP ein XSL-Stylesheet. Dies ist ein HTML-ähnliches Template, welches die Struktur des Ausgabedokuments festlegt. Für dynamische Daten gibt es Platzhalter. Bei der Erzeugung der PDFs wird neben dem Stylesheet auch noch eine XML Datei eingelesen, welche die Daten enthält, mit denen die Platzhalter ersetzt werden.

#### 4.1.4 Realisierung

Das Anheben der Version Apache FOP in der pom.xml führt wie erwartet zu Compiler Fehlern, hervorgerufen durch Syntaxänderungen in Apache FOP. Ein Blick in die Dokumentation und das Änderungsprotokoll gibt Aufschluss darüber, welche Anpassungen vorgenommen werden müssen. Im Zuge dieser Anpassung wird das Attribut „Accessibility“ des „FOUserAgent“ auf „true“ gesetzt. Der UserAgent ist in der Version 1.0 optional, wird nun aber benötigt um die PDF-Erstellung zu konfigurieren. Das Setzen dieses Attributs behebt einige Fehler die PAC3 anzeigt. Die übrigen Fehler müssen manuell behoben werden. Das geschieht durch Anpassung der Konfigurations-XML-Datei sowie der XSL-Stylesheets.

Das Setzen eines Alternativtitels für eine Grafik geschieht im Stylesheet und beinhaltet lediglich das Setzen eines Attributs im entsprechenden Element, siehe Abbildung 2. Dadurch wird ein versteckter Text hinzugefügt, den Screenreader vorlesen können.

```

72 <fo:block space-before.optimum="10pt" space-after.optimum="30pt" text-align="right" id="start">
73 <fo:external-graphic fox:alt-text="Schwarzer Bundesadler auf weißem Grund">
74   <xsl:attribute name="src">
75     url(<xsl:value-of select="/dm:drucksachenmappe/dm:image" />)
76   </xsl:attribute>
77   <xsl:attribute name="content-height">60px</xsl:attribute>
78 </fo:external-graphic>
79 </fo:block>

```

Abbildung 3 Das fox:alt Attribut in einem Grafik Element des Stylesheets

PAC3 bemängelt außerdem das Fehlen einer PDF/UA (Universal Accessibility) Kennzeichnung. Diese wird eingefügt, indem man in der Konfigurationsdatei das Element „<pdf-uy-mode>PDF/UA-1</pdf-ua-mode>“ hinzufügt.

Da bei normalen PDFs der Dateiname als Titel angezeigt wird, muss zudem der Eintrag „DisplayDocTitel“ gesetzt werden. Bei dieser Gelegenheit wird auch noch die Sprache des Dokuments festgelegt. In diesem Fall werden die entsprechenden Elemente im Stylesheet eingefügt.

Zur Bestehung der Barrierefreiheitsprüfung müssen außerdem alle verwendeten Schriftarten ins Dokument eingebettet werden, so dass diese immer korrekt angezeigt werden, auch wenn die entsprechende Schriftart auf dem Zielrechner selbst nicht vorhanden ist. In der Konfigurationsdatei kann dafür ein „Fonts“ Block eingefügt werden, zu sehen in Abbildung 3. Die entsprechenden .ttf Dateien der Schriftarten müssen in der Dateistruktur des Projekts vorhanden sein und werden über eine relative URL aufgelöst.

```

<fonts>
  <font kerning="true" embed-url="georgia.ttf">
    <font-triplet name="Georgia" style="normal" weight="normal"/>
    <font-triplet name="GeorgiaMT" style="normal" weight="normal"/>
  </font>
  <font kerning="true" embed-url="georgiab.ttf">
    <font-triplet name="Georgia" style="normal" weight="bold"/>
    <font-triplet name="GeorgiaMT" style="normal" weight="bold"/>
  </font>

```

Abbildung 4 Ausschnitt der Fonts-Sektion der Apache FOP Konfigurationsdatei

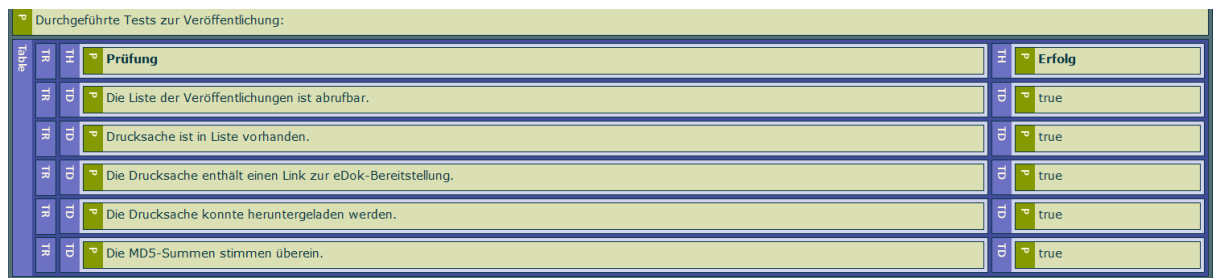
Tabellen, wie sie im Mantelbogen vorkommen, können für Screenreader-Nutzer schwer verständlich sein, wenn nicht klar ist, ob eine Zelle eine Überschrift oder Daten enthält. Dazu kann eine Zelle als TH (Table Header) oder TD (Table Data) markiert werden. In Abbildung 4 sieht man die „Normalansicht“ der Tabelle, Abbildung 5 zeigt die Tabelle in der Screenreader-Ansicht. Die fettgedruckten

Einträge in Abbildung 5 zeigen, dass es sich um Überschriften handelt.

Durchgeführte Tests zur Veröffentlichung:

Prüfung	Erfolg
Die Liste der Veröffentlichungen ist abrufbar.	true
Drucksache ist in Liste vorhanden.	true
Die Drucksache enthält einen Link zur eDok-Bereitstellung.	true
Die Drucksache konnte heruntergeladen werden.	true
Die MD5-Summen stimmen überein.	true

Abbildung 5 Die Tabelle der Mantelbogen PDF



Durchgeführte Tests zur Veröffentlichung:	
Prüfung	Erfolg
Die Liste der Veröffentlichungen ist abrufbar.	true
Drucksache ist in Liste vorhanden.	true
Die Drucksache enthält einen Link zur eDok-Bereitstellung.	true
Die Drucksache konnte heruntergeladen werden.	true
Die MD5-Summen stimmen überein.	true

Abbildung 6 Die Tabelle der Mantelbogen PDF als Screenreader Ansicht in PAC3

#### 4.1.5 Nachweis/Bewertung der Funktionsfähigkeit

Zum Nachweis der Funktionsfähigkeit müssen Standardkriterien wie der erfolgreiche Abschluss aller Unit-/Integration- und Systemtests erfüllt sein. Gleichzeitig dürfen alle generierten PDFs in PAC3 keine Fehler erzeugen. Der Status der Tests kann in Jenkins, dem Monitoring Tool der Continuous Integration (CI) Pipeline, eingesehen werden. Im Idealfall sieht man dort eine „grüne“, also erfolgreich durchgelaufene Pipeline. Außerdem werden manuelle Tests durchgeführt, bei denen die wichtigsten Funktionen noch einmal durch ein Teammitglied kontrolliert werden. Nachdem eine Auswahl an erzeugbaren PDFs in PAC3 keine Fehler mehr aufweist, werden diese zusätzlich noch stichprobenartig durch einen Barrierefreiheitsspezialisten von SEITENBAU mit einem Screenreader überprüft.

#### 4.1.6 Zusammenfassung

Diese erste Aufgabe hat mir besonders dabei geholfen, das Build Tool Maven besser zu verstehen, mit dem ich hier zum ersten Mal in Kontakt kam. Außerdem war es interessant zu sehen, wie man mit Java, unter Zuhilfenahme einer 3rd Party Bibliothek, PDF-Dateien erstellen kann.

## 4.2 Aufgabe 2

### Dockerisierung von DokV

#### 4.2.1 Aufgabenbeschreibung und Zielsetzung

DokV soll dockerisiert werden, d.h. alle Module, bzw. deren Komponenten sollen in Docker Containern ausgeführt werden. Auf die Datenbank trifft das schon zu, die restlichen Komponenten laufen aber als Webapps in einem Tomcat Webserver, welcher direkt auf dem Zielsystem installiert ist. Die Komponenten sollen nach der Umstellung jeweils in einem eigenen Docker Container ausgeführt werden, in welchen wieder ein Tomcat Webserver läuft. Des Weiteren sollen Images der einzelnen Container über einen einzelnen Befehl mittels Docker Compose gestartet werden können. Die automatisierte Test-Pipeline soll wie zuvor funktionieren.

#### 4.2.2 Theoretische Grundlagen

Um einen Container zu starten, wird ein Image benötigt, welches die von DokV verwendete JVM Adoptium sowie einen Tomcat Webserver enthält. Mit diesem Image können dann beliebig viele Container gestartet werden. Das fertige Image wird in die Docker Registry gepushed. Aus dieser wird dann das Image „gepulled“, wenn ein Container gestartet werden soll.

Docker Compose ist ein Tool, das darauf ausgelegt ist, eine Anwendung zu starten, die aus mehreren Docker Containern besteht. In einer .yaml-Datei wird dazu für jeden Container ein Service definiert, der neben dem zu startenden Image alle wichtigen Einstellungen enthält.

Jenkins ist eine webbasierte CI Software, die bei beiden Projekten zum Starten der Test-Pipeline eingesetzt wird. In dem sog. Jenkinsfile wird beschrieben, wie das Projekt auf den Virtuellen Test-Maschinen (Test-VMs) gebaut und wie die einzelnen Pipeline Stages ausgeführt werden sollen.

Da bei jedem Git-Push eine Test-Pipeline läuft, muss außerdem der automatisierte Startup der Anwendung auf den Test-VMs angepasst werden. Dazu wird das Automatisierungswerkzeug Ansible verwendet. In dem sog. Playbook kann dort beschrieben werden wie ein System aussehen muss, damit z.B. eine Webanwendung darauf gestartet werden kann. Ansible muss lediglich auf der verwaltenden Maschine installiert sein und führt die gewünschten Anweisungen dann per Secure Shell (ssh) auf dem Zielsystem aus. So werden automatisiert Verzeichnisse erstellt, in denen Daten gespeichert, Konfigurationsdateien an die vorgesehenen Stellen kopiert oder benötigte Prozesse durch z.B. Shell Befehle gestartet werden sollen. Auch das Aufräumen von eventuell angefallenen Testdaten

kann durch Ansible vorgenommen werden. So wird sichergestellt, dass jeder Testdurchlauf auf derselben Grundlage erfolgt und reproduzierbar ist.

### 4.2.3 Realisierung

Für das Bauen der Images verwenden wir Paketo Buildpacks. In einem Shell Skript wird Paketo ausgeführt. Dabei wird angegeben, wo sich das von Maven erstellte WAR-Archiv befindet und welche Buildpacks verwendet werden sollen, so dass im fertigen Image Tomcat und die JVM Adoptium vorhanden sind. Außerdem werden einige Konfigurationsdateien vom Build ausgeschlossen, die nicht für den Produktiv- oder Test-Betrieb notwendig sind. Ist das Image erfolgreich erstellt, soll es zunächst lokal eingebunden werden. Dafür wird die docker-compose.yaml, die bisher nur die Datenbank als Container startet, um weitere Services erweitert. Hier wird unter anderem das Image angegeben, aus dem der Container erstellt werden soll.

Abbildung 6 zeigt den Anfang der Konfiguration des Containers DokV-informationsdienst. Die Platzhalter in der Form „\${}“ werden mit Daten aus der .env Datei ersetzt. Dabei ist „.env“ der Name, nachdem Docker Compose defaultmäßig sucht, um Platzhalter zu ersetzen. In dieser Datei stehen pro Zeile ein Platzhalter und der Wert, durch den dieser ersetzt werden soll. Der Zwischenschritt über die Platzhalter ist nicht notwendig, erleichtert aber das ändern der Werte, da diese alle an einem Platz stehen, statt überall über die Docker-Compose Datei verteilt zu sein.

```
DokV-informationsdienst:
  container_name: DokV-informationsdienst
  image: "${DOCKER_REGISTRY_DOMAIN}/${DOKV_INFORMATIONSDIENST_VERSION}"
  environment:
    JAVA_TOOL_OPTIONS: -Dfile.encoding=UTF-8
    SERVICE_BINDING_ROOT: /platform/bindings
    TZ: Europe/Berlin
  mem_limit: 2G
  pids_limit: 4096
```

Abbildung 7 Ausschnitt aus der Datei docker-compose.yaml

Ein Image ist schreibgeschützt, ein Container kann darin also keine Daten speichern. Stoppt und löscht man einen laufenden Container, so gehen ins Dateisystem geschriebene Daten wie z.B. Logdateien verloren. Um das zu verhindern werden sog. Volumes erstellt. Diese „spiegeln“ das „virtuelle Verzeichnis“ eines laufenden Containers auf das des Hostsystems. Wird der Container gelöscht, verbleiben die Dateien der Volumes auf dem Hostsystem. Dazu wird unter dem Punkt „volumes“ definiert, welches „echte“ Verzeichnis auf dem Hostsystem welchem „virtuellen“ Verzeichnis im Container entsprechen soll, zu sehen in Abbildung 7. Für die Container Verzeichnisse werden absolute Pfade angegeben, während die „echten“ Verzeichnisse relativ zur docker-compose.yaml angegeben werden. Dieser Teil ist

recht fehleranfällig, da Tippfehler in den Pfaden dazu führen, dass leere Verzeichnisse erstellt werden und kein Error geworfen wird. Daher ist es wichtig, sich zu vergewissern, dass alle Volumes vorhanden sind, z.B. durch IDE Funktionen mit denen man in das Dateisystem eines Containers schauen kann. Zudem markiert das „:ro“ am Ende der Pfade die Verzeichnisse als read only.

```
volumes:
  - ${DOKV_CA_CERTS}:/platform/bindings/:ro
  - ${DOKV_INFORMATIONSDIENST_CONFIG_PATH}:/workspace/WEB-INF/classes/config:ro
  - ${DOKV_INFORMATIONSDIENST_LOG4J_CONFIG_PATH}:/workspace/WEB-INF/classes/log4j.properties:ro
```

Abbildung 8 Ausschnitt aus docker-compose.yaml, Konfigurations der Volumes von projekt-informationsdienst

Einige Komponenten greifen per http auf andere zu. Da bisher alle Komponenten auf demselben System liefen, funktionierte das einfach über „http://localhost:port“. Eine Komponente, die in einem Docker Container läuft, ist darüber nun nicht mehr direkt erreichbar. Z.B. ist der NGINX Proxy, der bisher auf Port 80 lief, nun nur noch innerhalb des Containers darüber zu erreichen. Um das zu ändern, können ähnlich wie zuvor die Volumes nun auch Ports gespiegelt werden. In Abbildung 8 sieht man, dass für den Proxy der Container Port 80 mit dem Port 80 des Host Systems verknüpft wird. Damit der Proxy nun aber die Weiterleitung vornehmen kann, muss er mit den Containern der übrigen Komponenten kommunizieren. Dazu kann in docker-compose.yaml ein Netzwerk konfiguriert werden, dem dann alle Container hinzugefügt werden. Die Docker Engine sorgt dann dafür, dass aus Containersicht alle Komponenten weiterhin über localhost ansprechbar sind.

```
DokV-nginx:
  container_name: DokV-nginx
  image: "${DOCKER_REGISTRY_DOMAIN}/${DOKV_NGINX_VERSION}"
  user: "${DOKV_CONTAINER_UID}:${DOKV_CONTAINER_GID}"
  ports:
    - 80:80
```

Abbildung 9 Ausschnitt aus docker-compose.yaml, Port Konfiguration des Proxys

Nachdem der Container lokal startet und die Anwendung wie gewohnt funktioniert, ist der nächste Schritt, dafür zu sorgen, dass auch auf den VMs ein Container gestartet wird. Der Bau des Images erfolgt weiterhin über das Shell Skript, allerdings wird dieses nun in der Build-Stage der Jenkins Pipeline aufgerufen. Außerdem wird das Image in die firmeneigene Docker Registry gepusht, damit es von allen VMs aus per pull Anweisung zugänglich ist. Dafür werden die entsprechenden Shell- und Docker Befehle in den Jenkinsfile geschrieben.

Nun muss eine VM für das Ausführen der Tests vorbereitet werden. Dafür wird in der entsprechenden Stage das Playbook aufgerufen. Dieses muss jetzt noch um die Schritte erweitert werden, die notwendig sind, um die Container zu starten. Diese Schritte werden in die Datei docker.yml ausgelagert, welche über eine include

Anweisung in das Playbook eingebunden werden kann. Dadurch lassen sich die Anweisungen z.B. Komponentenweise trennen, wodurch das Playbook übersichtlich bleibt. In `docker.yml` wird unter anderem festgelegt, wo Dateien wie `docker-compose.yml` oder `.env` zu finden sind und in welches Verzeichnis sie auf der Test VM kopiert werden sollen. Alte Versionen dieser Dateien werden dabei überschrieben. Container, die eventuell noch von einem früheren Testdurchlauf gestartet sind, werden gelöscht. Anschließend werden über „docker-compose up“ wieder alle Container gestartet. Da in der Build-Stage die Images neu gebaut und in die Registry gepusht wurden, werden nun auch diese neuen Images zum Starten der Container verwendet.

#### **4.2.4 Nachweis/Bewertung der Funktionsfähigkeit**

In diesem Fall ist die Funktionsfähigkeit sehr leicht am Status der Testpipeline zu überprüfen. Sollte eine Komponente nicht starten oder keine Verbindung zu ihr möglich sein, würden einige Tests einen Fehler werfen und die Pipeline wäre „rot“. Zudem liefern die Monitoring Seiten einiger Komponenten Informationen dazu, welche Verbindungen bereit sind.

## 4.3 Aufgabe 3

Anpassung der Startseite von PKP

### 4.3.1 Aufgabenbeschreibung und Zielsetzung

Der Kunde von PKP hat sich ein neues Layout der Startseite gewünscht. Abbildung 9 und 10 zeigen das alte bzw. das neue Layout. Neben optischen Anpassungen sollen den einzelnen Kacheln, die jeweils ein Modul repräsentieren, sog. „Quicklinks“ hinzugefügt werden. Das sind Verknüpfungen auf Funktionen eines Moduls die direkt von der Startseite aus angesteuert werden können.

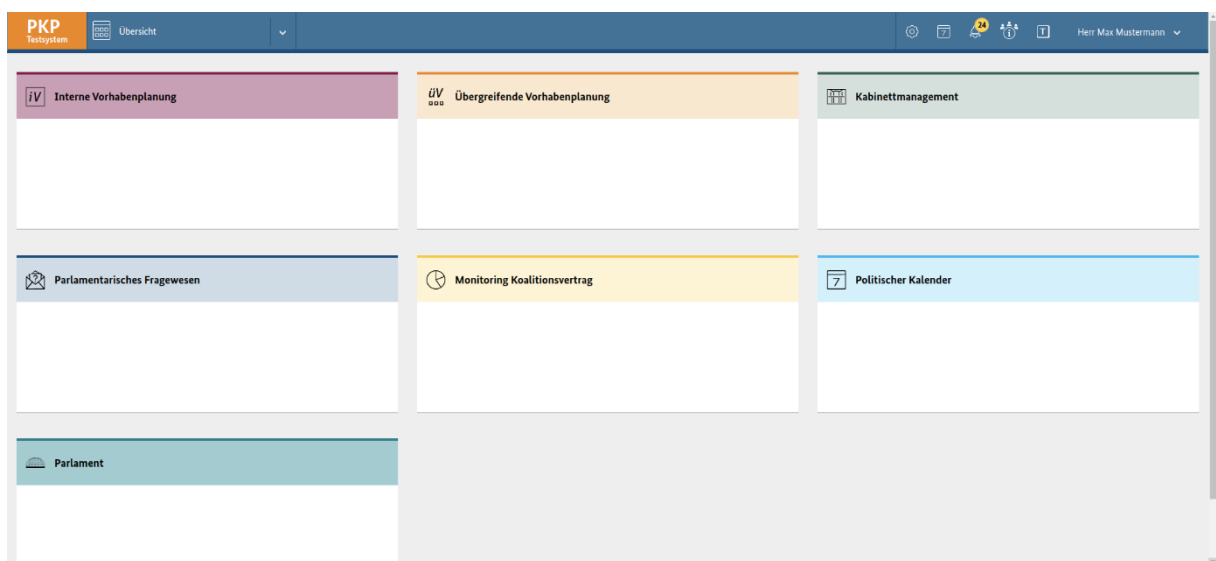


Abbildung 10 Das alte Layout der PKP Startseite

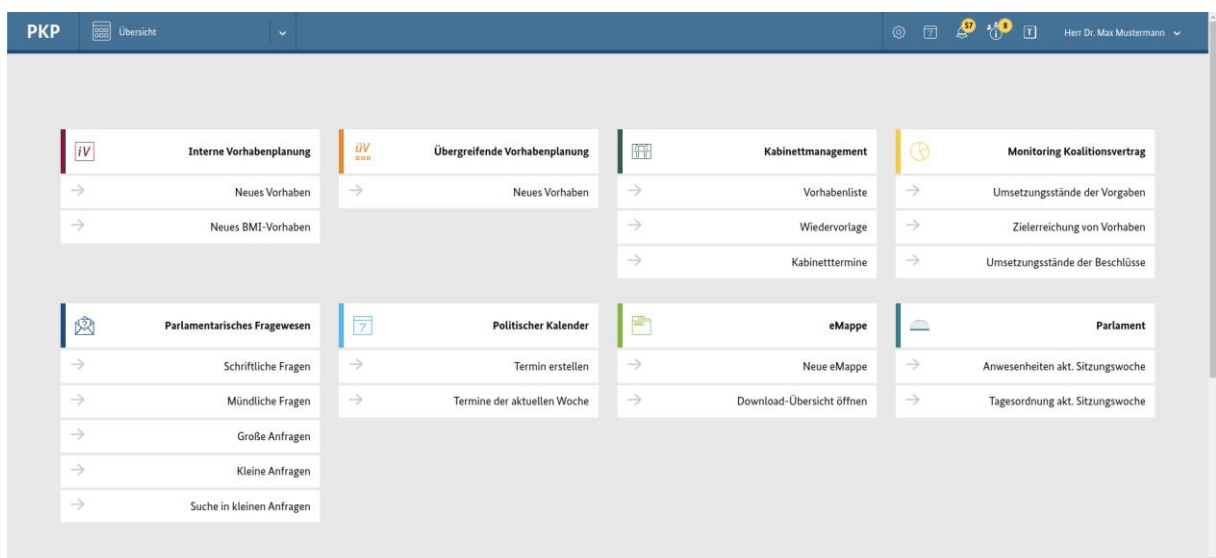


Abbildung 11 Das neue Layout der PKP Startseite



### 4.3.2 Theoretische Grundlage

Für das Frontend kommt die Template Engine Thymeleaf zum Einsatz. In Verbindung mit dem Framework Spring erleichtert es die Darstellung der angeforderten Daten. Im Java Code wird ein Model-Objekt mit den entsprechenden Daten befüllt, die beim Aufruf einer bestimmten URL benötigt werden. In den HTML-Dateien werden Variablen verwendet, die die Thymeleaf Engine dann durch die Daten aus dem Model-Objekt ersetzt.

### 4.3.3 Realisierung

Um die Anzahl der Probleme einzugrenzen, ist das erste Ziel das Layout so weit anzupassen, dass es der Desktop Version der Vorlage optisch ähnelt. Zunächst wird dafür das CSS angepasst, um die Farbe der Icons zu ändern, den farbigen Rand zu verschieben und die Hintergrundfarbe zu ändern. Das geschieht indirekt über die Anpassung von .less Dateien. Less ist eine Stylesheet-Sprache, die ähnlich zu benutzen ist wie css und auch in diese kompiliert wird.

Als nächstes wird mit Thymeleaf ein Quicklink-„Fragment“ definiert, zu sehen in Abbildung 11. Die URL des Links sowie die Bezeichnung wird dabei als Parameter definiert. Dieses Fragment kann dann in den Kacheln mehrfach eingefügt werden. Da noch keine Links vorhanden sind, werden diese Fragmente zunächst mit Dummy-Daten in die Kacheln eingesetzt.

```
<!--/* einzelner quick-link */-->
<div th:fragment="quick-link(bezeichnung, link)">
  <a class="quick-link" th:href="@{${link}}">
    <span class="ql-icon">
      <div th:replace="components/svg :: svg (icon='icon_arrow_right')"></div>
    </span>
    <span class="ql-text" th:text="#{${bezeichnung}}"></span>
  </a>
</div>
```

Abbildung 12 Thymeleaf Fragment, welches einen Quicklink darstellt

Wie in Abbildung 10 zu sehen ist, soll außerdem ein kleiner Pfeil anzeigen, dass es sich um einen Link handelt. Dafür wird eine Scalable Vector Graphic (SVG) verwendet. In einem XML-Dokument wird hierbei das Icon definiert. Durch die Kommandos M (move) und L (line) bewegt sich ein virtueller Stift auf einem Koordinatenfeld oder zeichnet Linien, die sich zu einem Polygon verbinden lassen. Abbildung 12 zeigt, wie der Pfad in der XML Datei bzw. das fertige Icon aussieht.

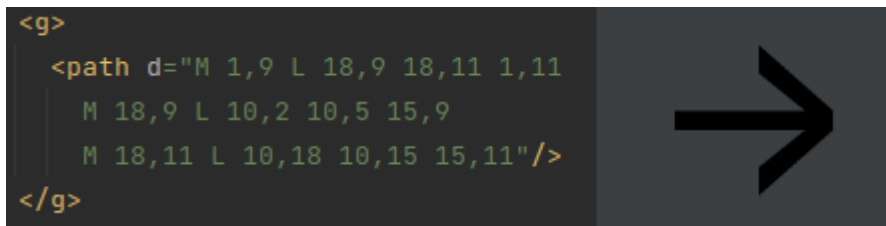


Abbildung 13 Ausschnitt aus der SVG Datei (Links) mit den Kommandos die zu dem Pfeil Icon (Rechts) führen

Nun müssen im Backend die Daten gesammelt werden, die für die Quicklinks benötigt werden. Dafür wird dem Model-Objekt eine HashMap hinzugefügt, die als Key die Kurzbezeichnung der einzelnen Module enthält und als Wert eine Liste, die die eigentlichen Links und deren Bezeichnungen enthält. Im HTML Dokument besteht dann die Möglichkeit, mittels Thymeleaf Variablen auf diese zuzugreifen. Beim Befüllen der HashMap wird zudem der AccessContext des angemeldeten Benutzers als Parameter mitgegeben. Dieses Objekt enthält unter anderem die Rollen, die dem Nutzer zugeordnet sind. Diese können dann beim Hinzufügen eines jeden Quicklinks abgeglichen werden. Links auf Funktionen, auf die der Nutzer keine Rechte hat, werden gar nicht erst in das Model-Objekt eingefügt und können auch nicht angezeigt werden.

Damit die Anzeigenamen der Links leicht geändert werden können, werden diese in eine „message.properties“ Datei ausgelagert. Diese enthält Schlüssel-Wert Paare, die mittels Spring-Funktionen im Front- und Backend ausgelesen werden können. Durch Verwendung des Schlüssels lässt sich ein Anzeigetext in allen möglichen grafischen Oberflächen wiederverwenden. Auch kann er an einer zentralen Stelle geändert werden.

Um nun in jedem Modul die richtigen Links anzuzeigen, wird das Thymeleaf Attribut „th:each“ verwendet, welches es ermöglicht, in der HTML Datei über Java-Collections zu iterieren, die sich im Model-Objekt befinden. Durch den Key des jeweiligen Moduls kann auf die Quicklinks zugegriffen werden.

Da sich ein Teil der grafischen Oberfläche verändert hat, müssen bei dieser Story bestehende Visual Regression Tests (VRT) angepasst werden. Bei diesen werden, nach dem Einspielen von Testdaten, verschiedene URLs aufgerufen. Anschließend wird ein Screenshot erstellt und dieser dann mit einer Vorlage abgeglichen. Dafür wird bei PKP BackstopJS eingesetzt, welches das automatische Erstellen und Vergleichen von Screenshots ermöglicht. Am Ende des Durchlaufs wird automatisch ein Bericht erstellt, der genau zeigt, wo sich der Test Screenshot und die Vorlage unterscheiden. Da der VR Test für die Startseite schon existiert, muss nun lediglich die Vorlage aktualisiert werden. Dafür wird das Szenario, welches die Startseite beinhaltet, mit der „reference“ Option aufgerufen. BackstopJS macht nun einen Screenshot und ersetzt damit die alte Vorlage.

#### **4.3.4 Nachweis/Bewertung der Funktionsfähigkeit**

Der Erfolg dieser Story wird zum einen wieder am Status der Test Pipeline festgemacht. Zum anderen wird die Startseite aber auch noch manuell getestet. Dabei wird auf Anzeigefehler geachtet und die Links werden ausprobiert. Außerdem loggt man sich mit verschiedenen Berechtigungen ein, um zu sehen, ob alle Elemente ausgeblendet werden, die nicht zu sehen sein sollen.

## 5 Persönliche Eindrücke und Erfahrungen

Während der Einarbeitungsphase wurde ich in kurzer Zeit mit sehr vielen neuen Informationen konfrontiert. Ich musste nicht nur lernen mich im Quellcode der Anwendungen zurechtzufinden, sondern auch mit den eingesetzten Werkzeugen wie Jira oder Jenkins zurechtkommen. Auch das Aufsetzen der Entwicklungsumgebung stellte trotz Dokumentation eine kleine Herausforderung dar. Nachdem diese Hürden überwunden waren, fand ich aber zunehmend Spaß an den Aufgaben. Da das Team für alles rund um das Projekt zuständig ist, hatte ich die Möglichkeit alles näher kennenzulernen, was zur Umsetzung eines solchen Projektes dazugehört. So konnte ich mein bisher erworbenes Wissen vor allem im Java Backend einsetzen, darüber hinaus aber auch viel Neues lernen, wie z.B. den Umgang mit Thymeleaf oder Details über die Infrastruktur der automatisierten Tests oder des Deployments. Vor allem die Arbeit an der Dockerisierung der Anwendung DokV war sehr interessant. Auch hier war der Einstieg etwas schwierig, da ich von Docker lediglich gehört, aber keinerlei praktische Vorerfahrung hatte. Dafür hatte die Anwendung fast schon „Tutorial Charakter“. Ein der beiden Komponenten wurde zuvor schon dockerisiert, so dass ich mich daran orientieren konnte. Zudem mussten jeweils ca. 5 Module in Container verpackt werden, weshalb ich das Gelernte öfter wiederholen konnte.

Die Atmosphäre im Team war hervorragend. Ich wurde von Tag eins an super aufgenommen und in alles mit einbezogen. Bei Schwierigkeiten konnte ich mich jederzeit an die anderen Teammitglieder wenden, egal ob vor Ort im Büro oder remote über den Chat. Im Büro war wegen Corona meist eher wenig los, die Stimmung war aber nicht zuletzt wegen bereitgestellten Getränken und Snacks immer gut. Ein Highlight war in dieser Hinsicht definitiv das Sommerfest, bei dem fast das ganze Team zusammen kam.