Abschlussprüfung Winter 2023

Fachinformatiker für Anwendungsentwicklung
Dokumentation zur betrieblichen Projektarbeit

AO-ENTON

Elektronische Neubewerbungen Transferieren aus Online-Komponente

Abgabetermin: Vechta, den 06.12.2023

Prüfungsbewerber:
Christian Eirich
[redacted]

Ausbildungsbetrieb:
Alte Oldenburger Krankenversicherung AG
Alte-Oldenburger-Platz 1
49377 Vechta

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

AO	Alte Oldenburger Krankenversicherung AG
API	Application Programming Interface
CDI	Contexts and Dependency Injection
CD	Continuous Deployment
CI	Continuous Integration
CRUD	Create, Read, Update, Delete
DBMS	Datenbankmanagementsystem
DDD	Domain-driven Design
DTO	Data Transfer Object
ENTON	Elektronische Neubewerbungen Transferieren aus Online-Komponente
ERM	Entity-Relationship-Modell
JPA	Jakarta Persistence API
JSON	JavaScript Object Notation
Jakarta EE	Jakarta Enterprise Edition
ORM	Objekt-relationaler Mapper
PDF	Portable Document Format
PR	Pull Request
QS	Qualitätssicherungsumgebung
REST	Representational State Transfer
ROBBE	Routiniert Online-Bewerbungen Bearbeiten und Erfassen
SCM	Source-Code-Management
TDD	Test-Driven Development
UUID	Universally Unique Identifier

1 Einleitung

In der folgenden Projektdokumentation wird der Ablauf des IHK-Abschlussprojekts beschrieben, das im Rahmen der Ausbildung zum Fachinformatiker für Anwendungsentwicklung bei der Alte Oldenburger Krankenversicherung AG (AO) realisiert wurde.

1.1 Projektumfeld

Die AO ist eine private Krankenversicherung mit Sitz in Vechta. Sie bietet private Krankenversicherungen, Pflegezusatzversicherungen, Auslandsreisekrankenversicherungen und Zusatzversicherungen für gesetzlich Versicherte an. Ende 2022 beschäftigte die AO etwa 280 Mitarbeiter mit steigender Tendenz. Auftraggeber des Projekts sind verschiedene Beteiligte am Bewerbungsprozess der AO, darunter die Personalabteilung, Ausbilder, Abteilungsleiter, Abteilungsdirektoren und die Mitarbeitervertretung.

1.2 Projektziel

Die AO zielt darauf ab, den Bewerbungsprozess erheblich zu modernisieren und effizienter zu gestalten. Mit dem Projekt Elektronische Neubewerbungen Transferieren aus Online-Komponente (ENTON) beabsichtigt das Unternehmen, eine Brücke zwischen potenziellen Bewerbern und dem internen Bewerbungsmanagementsystem Routiniert Online-Bewerbungen Bearbeiten und Erfassen (ROBBE) zu schlagen.

Das Hauptziel ist die Einführung einer Online-Komponente, welche es Bewerbern erlaubt, ihre Bewerbungsunterlagen unkompliziert digital einzureichen. Nachdem eine Bewerbung eingereicht und erfolgreich mit dem internen Bewerbungsmanagementsystem ROBBE synchronisiert wurde, erhält der Bewerber eine Bestätigungs-E-Mail. Diese Bestätigung dient als Nachweis und erste Rückmeldung, dass die Bewerbung erfolgreich eingegangen ist. Zudem sollen Bewerber die Möglichkeit haben, den Status ihrer Bewerbung in Echtzeit zu überprüfen.

Darüber hinaus soll die Online-Komponente eine integrierte Terminverwaltungsfunktion enthalten. Bewerber erhalten einen Link per E-Mail, durch den sie zu einem Bewerbungsgespräch eingeladen werden. Über diesen Link können sie verfügbare Termine einsehen, diese nach Bedarf auswählen, bestätigen oder auch stornieren. Diese verfügbaren Termine werden von den Beteiligten des Bewerbungsprozesses in ROBBE regelmäßig gepflegt.

Für eine reibungslose Integration und Kommunikation zwischen dem neuen System ENTON und dem bestehenden Bewerbungsmanagementsystem ROBBE soll zudem eine Application Programming Interface (API) entwickelt werden. Diese API wird dazu beitragen, dass neue Daten regelmäßig und sicher von ENTON zu ROBBE übertragen werden.

Um den Datenschutz und die Sicherheit der Bewerberdaten zu gewährleisten, wird die Komponente nach aktuellen Sicherheitsstandards entwickelt, wobei sie außerhalb des firmeninternen Netzwerks betrieben wird und dennoch mit dem Intranet-System ROBBE kommuniziert.

Insgesamt soll durch das Projekt ENTON nicht nur die Effizienz im Bewerbungsmanagementprozess gesteigert werden, sondern auch die Benutzererfahrung für Bewerber erheblich verbessert werden, indem ihnen ein benutzerfreundlicher, transparenter und schneller Prozess geboten wird.

1.3 Projektbegründung

Der derzeitige Bewerbungsprozess der AO ist in vielerlei Hinsicht ineffizient und arbeitsintensiv. Insbesondere die manuelle Übertragung der Bewerberdaten aus externen Quellen in das interne Bewerbungsmanagementsystem ROBBE stellt einen bedeutenden Zeit- und Arbeitsaufwand dar. Die Notwendigkeit, Bewerbungsunterlagen an bestimmten Arbeitsplätzen herunterzuladen und sie zur Sicherheitsanalyse auf ein anderes System zu übertragen, ist nicht nur zeitaufwändig, sondern auch fehleranfällig. Hinzu kommt die zeitaufwändige Terminfindung, die sowohl für das Unternehmen als auch für die Bewerber oft mühsam und ineffizient ist. Diese umständlichen Schritte bergen das Risiko von Datenverlust oder -verfälschung. Zudem kann der aktuelle manuelle Prozess zu Verzögerungen bei der Bewerbungsabwicklung führen, was potenziell zu einer negativen Bewerbererfahrung und dem Verlust qualifizierter Kandidaten beiträgt.

1.4 Projektschnittstellen

Das ENTON-System wird eng mit dem aktuellen Bewerbermanagementsystem ROBBE zusammenarbeiten. Für diese Interaktion ist die Entwicklung einer API vorgesehen. Diese Schnittstelle wird gemäß den Prinzipien der Representational State Transfer (REST)-Architektur konzipiert und nutzt das Datenformat JavaScript Object Notation (JSON), um eine reibungslose und standardisierte Datenkommunikation zu gewährleisten. Bewerber werden die primären Nutzer der zu entwickelnden Anwendung ENTON sein. Sie können verfügbare Termine einsehen und auswählen, die zuvor von den Zuständigen einer ausgeschriebenen Stelle in ROBBE eingepflegt wurden. Andere Daten, wie der aktuelle Stand der Bewerbung, werden automatisiert aus ROBBE gelesen.

1.5 Projektabgrenzung

Das Hauptziel dieses Projekts ist die Entwicklung von ENTON als eigenständige Anwendung, die den Bewerbungsprozess der AO digitalisiert und optimiert. Ein wesentlicher Bestandteil des Projekts ist die Entwicklung einer API für ENTON, die von ROBBE genutzt wird, um eine reibungslose und sichere Kommunikation zwischen den beiden Systemen zu gewährleisten.

Darüber hinaus wird im Rahmen dieses Projekts eine Termin-Eintrage-Funktion sowie der E-Mail-Versand in ROBBE implementiert. Diese Funktion ermöglicht es den Beteiligten einer ausgeschriebenen Stelle, verfügbare Termine für Bewerbungsgespräche in das System einzupflegen. Bewerber können dann über einen ihnen per E-Mail zugesandten Link auf diese Termine zugreifen und einen passenden Termin buchen.

Explizit nicht Teil des Projekts sind die Überarbeitung und Weiterentwicklung von ROBBE abseits der genannten Funktionen. Das Hauptaugenmerk liegt auf der Entwicklung von ENTON und den damit verbundenen Funktionen und Schnittstellen.

2 Projektplanung

Im Folgenden wird die Planung des Projekts ENTON dargestellt, wobei zunächst die einzelnen Projektphasen festgelegt wurden. Anschließend wurden die benötigten Ressourcen identifiziert und der Entwicklungsprozess definiert.

2.1 Projektphasen

Der Entwicklungsprozess für dieses Projekt orientiert sich am Wasserfallmodell. Das Wasserfallmodell ist ein sequenzielles Vorgehensmodell, bei dem jede Phase erst beginnt, wenn die vorherige abgeschlossen ist. Dieses Modell ist besonders geeignet, wenn die Anforderungen im Voraus klar definiert sind, wie es bei diesem Projekt der Fall ist.[1]

Projektphase	**Geplante Zeit**
Analysephase	7 h
Entwurfsphase	11 h
Implementierungsphase	48 h
Abnahme und Einführung	7 h
Erstellen der Dokumentation	7 h
Gesamt	**80 h**

Tabelle 1: Grobe Zeitplanung

Die Projektlaufzeit beträgt insgesamt 80 Stunden. Tabelle 1 gibt eine Übersicht über die Verteilung dieser Stunden auf die einzelnen Phasen des Wasserfallmodells. Für eine präzisere Zeitaufschlüsselung inklusive Teilschritte, siehe Anhang A.1: Detaillierte Zeitplanung auf Seite i.

2.2 Ressourcenplanung

Um das Projekt erfolgreich durchzuführen, werden sowohl physische, als auch technische Ressourcen benötigt. Dies umfasst Räumlichkeiten, Hardware, Software und ggf. zusätzliches Personal. Bei der Softwareauswahl wurde besonderer Wert auf kostenfreie oder bereits in der AO verwendete Lösungen gelegt, um zusätzliche Kosten zu minimieren. Ein detaillierter Überblick über die verwendeten Ressourcen findet sich im Anhang A.2: Ressourcenplanung auf Seite ii.

2.3 Entwicklungsprozess

Im Projekt wurde besonderes Augenmerk auf die Nutzung moderner Entwicklungspraktiken gelegt. Hierzu gehört der Einsatz von Git als Versionskontrollsystem. Durch den Gebrauch von Pull Requests konnte sichergestellt werden, dass jede Änderung am Code vor ihrer Integration in den `Main`-Branch überprüft wurde. Dies gewährleistet eine hohe Codequalität und reduziert die Wahrscheinlichkeit von Fehlern.

Des Weiteren wurden für jedes neue Feature oder jeden Fehler, der im Laufe des Projekts identifiziert wurde, separate Feature-Branches erstellt. Dies erleichtert nicht nur die Nachverfolgung von Änderungen und die Zuordnung zu spezifischen Anforderungen oder Problemen, sondern ermöglicht auch eine parallele Entwicklung unterschiedlicher Funktionen ohne Beeinträchtigung des Hauptentwicklungszweigs (`Main`-Branch).

3 Analysephase

In der Analysephase wurde eine Ist-Analyse durchgeführt, um den aktuellen Prozess zu verstehen. Danach wurde die Wirtschaftlichkeit des Projekts nachgewiesen und die Bedürfnisse des Bewerbers sowie der Mitarbeiter in Anwendungsfällen erfasst. Schließlich wurden die Anforderungen an die zu entwickelnde Software im Lastenheft festgehalten.

[1] Vgl. Bogdan-Alexandru u. a. [2019].

3.1 Ist-Analyse

Die Ist-Analyse dient dazu, ein detailliertes Bild des aktuellen Bewerbungsprozesses bei der AO zu gewinnen. Dies ist notwendig, um zielgerichtete Verbesserungen und Modernisierungen durchführen zu können.

Aktueller Stand Das interne Bewerbungsmanagementsystem ROBBE ist die zentrale Schnittstelle für die Verwaltung von Bewerbungsprozessen innerhalb der AO. Es ermöglicht verschiedenen Stakeholdern, wie z. B. der Personalabteilung, den Ausbildern und Abteilungsleitern, Bewerberdaten zu erfassen und zu verwalten. Das System ermöglicht es, spezifische Workflow-Prozesse für jedes Stellenangebot zu definieren. Verfügbar ist dieses jedoch nur im Intranet der AO.

Arbeitsablauf und Beteiligte Bei Eingang einer Bewerbung per E-Mail oder Post übernimmt die Personalabteilung die initiale Verarbeitung. Dazu gehört im Falle der E-Mail ein aufwändiger Virenscan, bevor die Daten manuell in das ROBBE-System eingetragen werden. Der für die jeweilige Stelle verantwortliche Ansprechpartner (z. B. Ausbilder für die Ausbildung oder Abteilungsleiter für Festanstellungen) kann anschließend alle Bewerbungen für die betreffende Stelle einsehen und weitere Schritte einleiten, wie zum Beispiel die Einladung zu einem Vorstellungsgespräch. Am Vorstellungsgespräch sind im Durchschnitt vier Mitarbeiter beteiligt, inklusive des Ansprechpartners der Stelle.

Identifizierte Herausforderungen Der aktuelle Prozess ist in mehrfacher Hinsicht ineffizient:

- Die manuelle Übertragung von Bewerberdaten ist zeitaufwendig und fehleranfällig.
- Das Fehlen einer Möglichkeit für Bewerber, den Status ihrer Bewerbung einzusehen oder Termine online zu vereinbaren, erzeugt zusätzlichen Verwaltungsaufwand.
- Die Beteiligung mehrerer Mitarbeiter im Terminfindungsprozess erfordert einen hohen Koordinationsaufwand.

Verbesserungsbedarf Aus der Analyse ergibt sich folgender Bedarf an Optimierungen:

- Implementierung der Möglichkeit, Bewerbungen online einzureichen, um manuelle Prozesse zu reduzieren.
- Übertragung von in ENTON erfassten Bewerberdaten nach ROBBE, um manuelle Prozesse zu reduzieren.
- Einführung einer Terminbuchungs- und verwaltungsfunktion, durch die eingeladene Bewerber Termine selbständig auswählen oder stornieren können.

Insgesamt macht die Ist-Analyse deutlich, dass die AO erheblichen Nutzen aus der Modernisierung und Automatisierung des aktuellen Bewerbungsprozesses ziehen würde.

3.2 Wirtschaftlichkeitsanalyse

Die Wirtschaftlichkeitsanalyse dient dazu, die Rentabilität und die finanzielle Machbarkeit des Projekts ENTON zu bewerten. Sie bietet eine umfassende Übersicht über die anfallenden Kosten und den erwarteten Nutzen.

3.2.1 „Make or Buy"-Entscheidung

Der Ausbildungsleiter der AO hat sich eine Lösung gewünscht, die möglichst wenig Kosten verursacht und gleichzeitig in das bestehende System ROBBE eingebunden werden kann.

In Anbetracht dieser Vorgaben wurde entschieden, eine Eigenentwicklung zu realisieren. Dieser Entschluss basierte auf der Einschätzung, dass eine speziell für die AO entwickelte Lösung die effektivste Möglichkeit darstellt, die reibungslose Integration mit ROBBE zu gewährleisten. Zudem bietet die Eigenentwicklung eine größere Flexibilität und Anpassungsfähigkeit an spezifische Bedürfnisse und Prozesse der AO, was mit einer Standardsoftware nicht erreichbar wäre.

3.2.2 Projektkosten

Für eine ganzheitliche und transparente Bewertung des Projekts wurde eine genaue Übersicht über die Projektkosten erstellt. Die Kosten umfassen nicht nur die reinen Entwicklungskosten, sondern auch die Kosten für alle Ressourcen, die im Projektverlauf benötigt wurden.

Kostenkalkulation Bei der Kostenkalkulation für das Projekt ENTON standen sowohl die Personalkosten als auch die Kosten für die identifizierten Ressourcen (siehe 2.2) im Vordergrund. Aus Vertraulichkeitsgründen können die genauen Personalkosten nicht offengelegt werden. Aus diesem Grund hat die Personalabteilung vor einiger Zeit in Abstimmung mit der Geschäftsführung der AO spezifische Stundensätze[2] für Projekte erstellt. Der festgelegte Stundensatz für einen Mitarbeiter aus der Personal- oder IT-Abteilung beträgt dabei 35 €. Der Stundensatz für einen Auszubildenden liegt bei 10 €.

Des Weiteren wurden für die genutzten Ressourcen[3], einschließlich aller Lizenzkosten genutzter Software, zusätzliche 15 € pro Stunde veranschlagt. Um die Übersicht in Tabelle 2 zu erleichtern, wurden diese Ressourcenkosten bereits zu den jeweiligen Stundensätzen addiert. Dadurch ergibt sich ein angepasster Stundensatz von 25 € für einen Auszubildenden und 50 € für einen Mitarbeiter der Personal- oder IT-Abteilung. Zusätzlich zu den reinen Entwicklungskosten fallen jährliche Kosten in Höhe von 145 € an. In diesen jährlichen Kosten enthalten sind bspw. Kosten für Wartung, Hosting und Domain.

Vorgang	Mitarbeiter	Zeit	Kosten pro Stunde	Kosten
Erstellung des Lastenheftes	5 Mitarbeiter[4]	3 h	50 € * 5 = 250 €	750 €
Abnahme der Mockups	5 Mitarbeiter[4]	1 h	50 € * 5 = 250 €	250 €
Entwicklungskosten	1 Auszubildender der IT	80 h	25 €	2.000 €
Pull Requests (PRs) abnehmen	1 Mitarbeiter der IT-Abteilung	3 h	50 €	150 €
Abnahme und Erfolgskontrolle	5 Mitarbeiter[4]	3 h	50 € * 5 = 250 €	750 €
				3.900 €

Tabelle 2: Projektkosten

[2] Die Stundensätze setzen sich u.a. aus dem Gehalt und Sozialaufwendungen zusammen.
[3] z. B. genutzte Räumlichkeiten, Stromkosten, etc.
[4] Ein Ausbilder, ein Abteilungsleiter, ein Abteilungsdirektor, ein Mitglied der Mitarbeitervertretung, eine Mitarbeiterin der Personalabteilung.

3.2.3 Amortisationsdauer

Um die Amortisationsdauer berechnen zu können, mussten die Projektkosten aus Tabelle 2 in Höhe von 3.900 € den Zeiteinsparungen gegenübergestellt werden. Die hier verwendeten Zeitwerte basieren auf Erfahrungswerten, die vom Fachbereich zur Verfügung gestellt wurden.

Vor der Implementierung:

- Download und Virenscan: 11 Minuten
- Terminabstimmung (4 Mitarbeiter[5]): 20 Minuten
- Überschneidungen finden: 5 Minuten
- Endgültige Abstimmung: 8 Minuten

Das ergibt insgesamt 44 Minuten pro Bewerbung.

Nach der Implementierung:

- Download und Virenscan: 0 Minuten (automatisiert)
- Terminabstimmung (4 Mitarbeiter): 20 Minuten
- Überschneidungen finden: 0 Minuten (automatisiert)
- Endgültige Abstimmung: 0 Minuten (automatisiert)

Das ergibt eine Zeiteinsparung von 24 Minuten pro Bewerbung, wodurch nur noch 20 Minuten Aufwand pro Bewerbung anfallen, die zum Eintragen verfügbarer Termine genutzt wird. Obwohl einige Schritte automatisiert wurden, bleibt die Eintragung der Termine eine manuelle Aufgabe, da sich die freien Termine je nach Mitarbeiter unterscheiden.

Von der Personalabteilung wurden 80 Bewerbungen pro Jahr als Durchschnittsmenge vorgegeben. Die Zeiteinsparung pro Jahr liegt somit bei 1.920 Minuten.

Um die jährliche Kosteneinsparung zu berechnen, wird der vorherige Stundensatz i.H.v 50,00 € pro Stunde verwendet:

$$\frac{1.920 \text{ min/Jahr}}{60} \cdot 50\,€/\text{h} \approx 1.600,00\,€/\text{Jahr}. \tag{1}$$

Um die genaue Amortisationsdauer zu berechnen, werden die Projektkosten der jährlichen Kosteneinsparung abzüglich der jährlichen Betriebs- und Wartungskosten gegenübergestellt.

$$\frac{3.900{,}00\,€}{(1.600,00 - 145,00)\ €/\text{Jahr}} \approx 2,68 \text{ Jahre} \tag{2}$$

Die berechnete Amortisationsdauer beträgt etwa 2,68 Jahre. Dies deutet darauf hin, dass das Projekt nicht nur wirtschaftlich rentabel ist, sondern sich auch in einer relativ kurzen Zeitspanne amortisiert. Darüber hinaus sollte beachtet werden, dass die AO kontinuierlich neue Mitarbeiter einstellt und ein Sinken der Bewerberzahlen nicht absehbar ist, sondern sogar eher eine Steigerung. Dadurch bleibt das Projekt ENTON vermutlich langfristig relevant und im Einsatz.

[5] Durchschnittsmenge.

3.3 Anwendungsfälle

In der Analysephase wurde ein Anwendungsfall-Diagramm entwickelt, das eine grobe Darstellung der funktionalen Anforderungen bietet. Dieses Diagramm, abgebildet im Anhang A.3: Use-Case-Diagramm auf Seite iii, stellt die Funktionen dar, die den Bewerbern und den Mitarbeitern zur Verfügung stehen. Die Erstellung dieses Diagramms war wichtig, um sicherzustellen, dass das Endprodukt den Anforderungen und Erwartungen der Nutzer entspricht und alle notwendigen Funktionen abdeckt. Es diente als grundlegende Orientierungshilfe für die Entwicklung und half dabei, Missverständnisse im Team zu vermeiden.

3.4 Lastenheft

Den Abschluss der Analysephase markierte die gemeinsame Erstellung eines Lastenheftes mit den fünf beteiligten Mitarbeitern. Dieses Dokument fungiert als Grundlage für alle fachlichen Anforderungen und setzt den Rahmen für das anschließende Pflichtenheft, welches die technischen Spezifikationen der fachlichen Anforderungen definiert. Die Anforderungen wurden systematisch erfasst und mithilfe der MoSCoW-Methode priorisiert. In diesem Zusammenhang wurden Anforderungen als „muss“, „soll“, „kann“ oder „wird nicht“ klassifiziert.[6] Ein Auszug des Lastenheftes ist im Anhang A.4: Lastenheft auf Seite iii einzusehen. Auf dessen Grundlage wurde auch das Pflichtenheft in Abschnitt 4.8 Pflichtenheft entwickelt.

4 Entwurfsphase

Nach der vollständigen Analyse der Projektanforderungen begann die Entwurfsphase. In dieser Phase wurden die Architektur der Anwendung, die Wahl der Technologien und die Datenstrukturen definiert. Zudem wurden Maßnahmen zur Qualitätssicherung festgelegt und das Pflichtenheft erstellt, welches die technischen Spezifikationen detailliert ausführt.

4.1 Zielplattform

Den Architekturrichtlinien der AO folgend wurde entschieden, das Projekt als Webanwendung umzusetzen, um eine hohe Zugänglichkeit und Benutzerfreundlichkeit für Bewerber zu ermöglichen. Konform zu diesen Richtlinien wurde für die Entwicklung der Webanwendung Java 17 in Verbindung mit der Spezifikation Jakarta EE ausgewählt. Darüber hinaus wird Git als Versionsverwaltungssystem und Gradle als Build-Tool verwendet. Der Build-Prozess wird durch Jenkins automatisiert, welches auch durch eine Integration mit SonarQube die Einhaltung der Qualitätsstandards sichert.

4.2 Architekturdesign

Bewertung und Auswahl von Frameworks Quarkus wurde als Framework für die Entwicklung ausgewählt. Es bietet eine Reihe von Vorteilen wie die Implementierung der Spezifikation Jakarta Enterprise Edition (Jakarta EE), schnelle Startzeiten, niedrigen Speicherverbrauch und eine breite Palette von Funktionen.[7] MariaDB wurde als Datenbankmanagementsystem (DBMS) ausgewählt, da es eine hohe Leistung und Flexibilität bietet und sich bereits in der bestehenden IT-Infrastruktur der AO bewährt hat. Darüber hinaus wurde Vavr als Framework eingesetzt, welches die funktionalen Programmierkonzepte in Java erweitert. Es bietet bspw.

[6] Vgl. Schulz [2023].
[7] Vgl. Quarkus [2023].

Abstraktionen für Werteklassen oder eine funktionale Ausnahmebehandelung, die die Code-Qualität und Wartbarkeit verbessern.[8] Die Verwendung von Vavr ermöglicht eine eher deklarative Programmierweise, die unnötige Seiteneffekte vermeidet und die Lesbarkeit des Codes verbessert.[9]

Wahl der Anwendungsarchitektur Das Projekt setzt auf eine Schichtenarchitektur, die sich durch eine klare Trennung der Verantwortlichkeiten auszeichnet. Diese Architektur erleichtert die Wartbarkeit, Testbarkeit und Skalierbarkeit der Anwendung, indem sie die Kopplung zwischen den Komponenten minimiert und die Kohäsion innerhalb jeder Schicht verbessert. Auch unterstützt die klare Strukturierung die Erweiterbarkeit und Anpassungsfähigkeit der Anwendung für zukünftige Funktionalitäten oder Änderungen. Das im Anhang A.5: Komponentendiagramm auf Seite v dargestellte Komponentendiagramm visualisiert diese Architektur und zeigt die Aufteilung in die Hauptbereiche View, Domäne und Persistenz.

Domänen-Schicht Die Domänen-Schicht ist der zentrale Teil der Anwendung und umfasst:

- **Model**: Hier werden die Entitäten und die damit verbundene fachliche Logik im Sinne von Domain-driven Design (DDD) definiert.
- **Services**: Diese Klassen implementieren die Geschäftslogik sowie Use-Cases der Anwendung.
- **Repositorys**: Stellen Interfaces für Operationen zur Verfügung, die sich nach Create, Read, Update, Delete (CRUD) richten. Die eigentliche Implementierung erfolgt in der Persistenz-Schicht.
- **Data Transfer Objects (DTOs)**: Diese Klassen sind für den Datenaustausch zwischen der Benutzeroberfläche bzw. der API und den Service-Klassen verantwortlich.

View-Schicht Die View-Schicht ist für die Benutzeroberfläche verantwortlich. Sie verwendet Qute und Renarde, um dynamische Webseiten zu generieren. Controller-Klassen bereiten die Daten für die Templates vor und stellen sie zur Verfügung.

Ressourcen-Schicht Diese Schicht stellt die REST-API von ENTON zur Verfügung und enthält spezialisierte Klassen, mit der das System ROBBE interagiert.

Persistenz-Schicht Die Persistenz-Schicht ist die Schnittstelle zur Datenbank. Sie verwendet Jakarta Persistence API (JPA) als Objekt-relationalen Mapper (ORM). Die eigentliche Implementierung der Repository-Interfaces erfolgt hier. Ein JDBC-MariaDB-Treiber wird für die Kommunikation mit der MariaDB-Datenbank verwendet.

4.3 Entwurf der Benutzeroberfläche

Beschreibung des visuellen Entwurfs Wie bereits unter 4.1 genannt, wird das Projekt als Webanwendung entwickelt. Die Mockups für die Benutzeroberfläche wurden mit Balsamiq erstellt und basieren auf den Anforderungen des Lastenhefts. Bei Erstellung der Mockups wurde darauf geachtet, dass die geplante Benutzeroberfläche möglichst klar und verständlich ist, um sie für eine breite Zielgruppe zugänglich zu machen. Nach der Erstellung wurden sie den beteiligten Personen zur Abnahme vorgelegt. Zwei ausgewählte Mockups befinden sich im Anhang A.6: Mockups auf Seite vi.

[8] Vgl. Schmitz [2018].
[9] Vgl. Schmitz [2018].

Das erste Mockup zeigt das Erfassungsformular für eine neue Bewerbung. Es enthält alle erforderlichen und optionalen Felder, die für die Bewerbung notwendig sind. Das zweite Mockup präsentiert die Benutzeroberfläche, auf der Bewerber verfügbare Termine buchen können.

Usability und Corporate Design Die Mockups wurden unter Berücksichtigung des Corporate Designs der AO und allgemeiner Usability-Richtlinien gestaltet. Dies umfasst unter anderem die Verwendung des Firmenlogos und die Gestaltung der Navigationsleiste im Header.

Für eine optimale Funktionalität wurde geplant, die Weboberfläche für alle gängigen Browser zu optimieren. Auch die Verwendung auf mobilen Geräten wurde berücksichtigt, um die Zugänglichkeit zu verbessern. Da keine Bewerbungen aus dem Ausland erwartet werden, wurde keine mehrsprachige Oberfläche geplant.

4.4 Datenmodell

Die Entwicklung des Entity-Relationship-Modells (ERMs) basierte auf dem Lastenheft und den in Abschnitt 4.3 Entwurf der Benutzeroberfläche entworfenen Mockups. Das ERM befindet sich im Anhang A.7: Entity-Relationship-Model auf Seite vii.

Die zentrale Entität ist `Bewerbung`, die alle Informationen zu einer Bewerbung enthält. So beinhaltet sie bspw. die persönlichen Daten des Bewerbers, eine Referenz auf die Stelle, auf die sich der Bewerber bewirbt sowie die hochgeladene Bewerbungsdatei. Sie steht in einer 1:n-Beziehung zur Entität `Termin`, da eine Bewerbung mehrere Termine haben kann und ein Termin immer zu genau einer Bewerbung gehört.

Die Entität `Termin` wiederum enthält alle Informationen zu einem bestimmten Termin. Diese enthält bspw. den Zeitpunkt des Termins und eine Referenz auf die Bewerbung, zu der der Termin gehört. Ebenfalls steht sie in einer 1:n-Beziehung zur Entität `Stelle`, da ein Termin immer zu genau einer Stelle gehört und zu einer Stelle mehrere Termine gehören können.

Der `Terminstatus` stellt den Status eines Termins dar, wie bspw. Frei oder Gebucht. Er steht in einer 1:n-Beziehung zur Entität `Termin`, da ein Termin immer genau einen Status hat und ein Status mehreren Terminen zugeordnet sein kann.

Die Entität `Stelle` repräsentiert eine offene Stelle in der AO. Dort enthalten sind Informationen wie die Bezeichnung der Stelle oder auch eine Beschreibung. Sie steht in einer 1:n-Beziehung zur Entität `Bewerbung`, da es auf eine offene Stelle mehrere Bewerbungen geben kann.

4.5 Geschäftslogik

Im neuen System reichen Bewerber ihre Bewerbungen online ein, wobei die Daten in ENTON gespeichert und dann über die API von ROBBE abgerufen werden. Ein internes Proxy-System prüft dabei auf Schadsoftware, was den manuellen Prüf- und Dateneingabeaufwand eliminiert.

Die Pflege freier Termine erfolgt in ROBBE, von wo aus diese regelmäßig nach ENTON übertragen werden. Sobald die Terminfindung in ROBBE initiiert wird, erhält der Bewerber eine E-Mail mit einem Link zur Terminabstimmung in ENTON, was den Koordinationsaufwand reduziert. Außerdem können Bewerber den Status ihrer Bewerbung online einsehen, was den Verwaltungsaufwand verringert.

Ein Aktivitätsdiagramm für den Bewerbungsprozess und ein weiteres für den Terminfindungsprozess sind im Anhang zu finden (Anhang A.8: Aktivitätsdiagramm für den Bewerbungsprozess auf Seite viii und Anhang A.9: Aktivitätsdiagramm für den Terminfindungs- und Buchungsprozess auf Seite viii), die die Geschäftslogik visuell darstellen.

Ergänzend gibt es ein Sequenzdiagramm (Anhang A.10: Sequenzdiagramm für die stündliche Synchronisation der Termine auf Seite ix), das die stündliche Synchronisation der Termine zwischen ROBBE und ENTON zeigt, wobei jede Anfrage einen Basic-Auth-Header zur Sicherheitsgewährleistung beinhaltet. Dieser Prozess beinhaltet die Anforderung und Aktualisierung gebuchter Termine in ROBBE, basierend auf den Daten von ENTON. Bei Fehlern wird ein Protokoll erstellt und der Entwickler automatisch benachrichtigt. Nach der erfolgreichen Übertragung gebuchter Termine fordert ROBBE das Löschen aller Termindaten in ENTON an und überträgt dann die aktuellen freien Termine nach ENTON. Fehler in diesem Schritt führen ebenfalls zur Protokollierung und Benachrichtigung. Dieses Verfahren sorgt dafür, dass ENTON stets die neuesten Termininformationen enthält.

4.6 Maßnahmen zur Qualitätssicherung

Um die Qualität des Projektergebnisses zu gewährleisten, wurden verschiedene Maßnahmen ergriffen. Eine der Hauptstrategien war die Anwendung von Test-Driven Development (TDD). In diesem Ansatz wird zuerst ein Test geschrieben, der fehlschlägt und die gewünschte Funktionalität beschreibt. Anschließend wird der Code implementiert, um den Test erfolgreich durchlaufen zu lassen. Nach erfolgreicher Implementierung wird der Code refaktorisiert, um die Qualität zu verbessern.[10]

Während der Entwicklung wurde der Continuous Testing-Modus von Quarkus genutzt. Dieser Modus führt kontinuierlich Unit-Tests im Hintergrund aus und ermöglicht so eine schnellere Entwicklung und höhere Codequalität.

Für die statische Code-Analyse kam SonarQube zum Einsatz. Diese Software analysiert den Code nach von der AO festgelegten Kriterien wie Code-Smells, Duplikationen, Bugs, Sicherheitsrisiken und Code-Coverage. Die AO erwartet eine Mindest-Code-Coverage von 80 Prozent.

Die Branching-Strategie sah vor, dass für jedes Ticket ein Feature-Branch erstellt wurde. Vor dem Merging eines Feature-Branches in den Branch `main` wurde ein umfassendes Code-Review durch einen erfahrenen Entwickler durchgeführt. Dieser Schritt diente der zusätzlichen Sicherstellung der Code-Qualität und der Einhaltung des in der AO erforderlichen Vieraugenprinzips. Erst nach erfolgreichem Review wurde der Branch in den Branch `main` gemerged. Jede Änderung im Branch `main` löst automatisch das Ausrollen einer neuen Qualitätssicherungsumgebung (QS) aus. Das Erstellen einer neuen Produktionsumgebung wird nur durch manuelles Hochsetzen der Versionsnummer ausgelöst.

4.7 Deployment

Ursprünglich war geplant, die Webanwendung mit Kubernetes zu deployen. Aufgrund von organisatorischen Einschränkungen und dem aktuellen Einsatzbereich von Kubernetes innerhalb der AO wurde jedoch entschieden, das Deployment mit einem Jenkins-Agent und Docker durchzuführen. Kubernetes wird derzeit nur für interne Anwendungen verwendet, und der Aufwand für die Einrichtung für dieses einzelne Projekt wäre unverhältnismäßig groß gewesen.

Die Webanwendung wurde nun auf einem bereits bestehenden Webserver in einem Docker-Container gehostet. Ein detailliertes Verteilungsdiagramm zur Darstellung der Deployment-Architektur ist im Anhang A.11: Deploymentdiagramm auf Seite x zu finden.

Ein spezialisierter Build-Server ist mit verschiedenen Tools ausgestattet: Jenkins für Continuous Integration (CI) und Continuous Deployment (CD), Gradle als Build-Tool, SonarQube für die Code-Analyse, und Artifactory für die Artefaktverwaltung.

[10] Vgl. Beck [2003].

Der Quellcode wird regelmäßig ins Source-Code-Management (SCM)-System übertragen. Jede Änderung im Branch `main` löst automatisch einen Build-Prozess in Jenkins aus. Dieser Prozess führt alle in der `build.gradle` definierten Schritte aus, einschließlich des Herunterladens von Abhängigkeiten und der Ausführung von Tests. Nach einem erfolgreichen Build wird ein Docker-Image erstellt und in Artifactory hochgeladen. Dieses Image wird dann automatisch auf dem bestehenden Webserver über eine Docker-Registry bereitgestellt und in Docker gestartet.

Die gesamte Jenkins-Pipeline ist in einem `Jenkinsfile` definiert, was eine effiziente und zuverlässige Bereitstellung der Anwendung sicherstellt.

4.8 Pflichtenheft

Das Pflichtenheft stellt die konkrete technische Umsetzung der im Lastenheft festgelegten Anforderungen dar (siehe Anhang A.4: Lastenheft auf Seite iii). Es dient als zentrales Dokument für die technische Planung und Implementierung der Anwendung. Bei Projektabschluss wird das Pflichtenheft als Referenzdokument verwendet, um sicherzustellen, dass alle vereinbarten Anforderungen technisch umgesetzt wurden. Ein Auszug des Pflichtenhefts ist im Anhang A.12: Pflichtenheft (Auszug) auf Seite xi zu finden.

5 Implementierungsphase

Die Implementierungsphase des Projekts umfasste die Ausarbeitung und Realisierung der Entwürfe aus Kapitel 4 (Entwurfsphase).

5.1 Aufsetzen des Projektes

Das Projekt wurde initial über die offizielle Quarkus-Webseite „code.quarkus.io“ erstellt. Dies diente als Grundgerüst und beinhaltete bereits die Kernabhängigkeiten für ein Quarkus-Projekt. Zusätzliche Abhängigkeiten wurden in der `build.gradle`-Datei ergänzt. Ein Auszug dieser befindet sich im Anhang A.13: `build.gradle` (Auszug) auf Seite xii.

Die `application.properties`-Datei von Quarkus wurde entsprechend konfiguriert, um bspw. Datenbankverbindungen und Authentifizierungsmechanismen zu konfigurieren. Ein Auszug dieser Datei ist im Anhang A.14: `application.properties` (Auszug) auf Seite xii zu finden.

Für den produktiven Betrieb der Webanwendung in einer Docker-Umgebung wurde ein spezielles `Dockerfile` erstellt. Dieses `Dockerfile` enthält Anweisungen für den Bau eines Docker-Images der Anwendung und ist im Anhang A.15: `Dockerfile` auf Seite xiii zu finden.

5.2 Implementierung der Datenstrukturen

Nach der Projektinitialisierung wurde die Implementierung der Datenstrukturen vorgenommen. In der Komponente Model befinden sich die Modelle, die die Entitäten der Domänenschicht repräsentieren. Die Entitäten wurden entsprechend dem Anhang A.7: Entity-Relationship-Model auf Seite vii erstellt. Eine abstrakte Basisklasse namens `Entitaet` wurde eingeführt, die das gemeinsame Attribut id enthält. Da dieses Attribut als Schlüssel dient, wurde es mit der JPA-Annotation `@EmbeddedId` gekennzeichnet. Als Datentyp wird eine Universally Unique Identifier (UUID) verwendet. Dies ermöglicht die Arbeit mit gültigen Entitäten ab dem Zeitpunkt ihrer Instanziierung, ohne auf eine Datenbankpersistenz angewiesen zu sein. Alle Entitäten wurden mit der Annotation `@Entity` annotiert, um sie als Entitätstyp für das Objekt-relationale Mapping (ORM) zu kennzeichnen.

Für jede Entität wurde eine statische Fabrikmethode erzeuge() implementiert, die ein Validation-Objekt von Vavr zurückgibt. Dieses Objekt enthält im Erfolgsfall die erstellte Entität und im

Fehlerfall ein Meldungen-Objekt, das alle aufgetretenen Fehlermeldungen kapselt. Der Standardkonstruktor jeder Entität wurde auf *private* gesetzt, um die Erstellung von Entitäten ausschließlich über die erzeuge()-Methode zu steuern und die Vavr-Bibliothek für robuste Fehlerbehandlung und Validierung zu nutzen.

Wertobjekte wie `Vorname`, `Nachname` und `Email` sind als Objekte modelliert, die sich über ihre Attribute definieren.

Beziehungen zwischen den Entitäten wurden mit der JPA-Annotation `ManyToOne` umgesetzt. Enum-Typen wie `Geschlecht` und `Terminstatus` wurden implementiert, um eine feste Menge an gültigen Werten zu definieren.

Zur Verdeutlichung ist Bewerbung.java im Anhang A.16: `Bewerbung.java` (Auszug) auf Seite xiii zu finden.

5.3 Implementierung der Geschäftslogik

Die Geschäftslogik des Projekts wurde nach der erfolgreichen Implementierung der Datenstrukturen entwickelt. Als Leitfaden dienten die Anwendungsfälle, die im Abschnitt 3.3 Anwendungsfälle behandelt wurden.

Um die Persistenzschicht sauber von der Geschäftslogik zu trennen, wurden Repository-Interfaces definiert. Diese wurden in der Persistenzschicht umgesetzt und mittels Contexts and Dependency Injection (CDI) und der `@Inject`-Annotation in die Services injiziert.

Ein Beispiel für die Umsetzung der Geschäftslogik ist der BewerbungService, welcher im Anhang A.17: `BewerbungService.java` auf Seite xiii zu finden ist. Diese Klasse ist speziell für die Verwaltung von Bewerbungen zuständig. Sie enthält Methoden, die die Anwendungsfälle in diesem Bereich abdecken. Die Methode erzeuge(NeueBewerbung neueBewerbung) nimmt das DTO NeueBewerbung entgegen, das die Eingaben des Benutzers aus der Oberfläche enthält. Diese Methode ist transaktional und nutzt ebenfalls die Vavr-Bibliothek für eine robuste Fehlerbehandlung und Validierung. Sollte die E-Mail-Adresse im DTO fehlen, wird die Methode sucheEmailInPdf(byte[] pdfInhalt) aufgerufen. Sie extrahiert den Text aus dem PDF und sucht mittels eines regulären Ausdrücks nach einer gültigen E-Mail-Adresse.

Die Architektur folgt dem Prinzip der *Dependency Inversion*, was die Testbarkeit des Codes erhöht und die Abhängigkeiten zwischen den verschiedenen Schichten minimiert. Durch die Verwendung spezifischer DTOs wird die klare Trennung von Datenübertragung und Geschäftslogik eingehalten, wodurch die Architektur übersichtlicher und wartungsfreundlicher wird.

5.4 Implementierung der Benutzeroberfläche

Die Benutzeroberfläche wurde mit der Template-Engine Qute und dem Web-Framework Renarde realisiert. Die Entscheidung für die Template-Engine Qute fiel, da Qute extra für Quarkus entwickelt wurde und somit eine sehr gute Integration bietet. Ein weiterer Grund für Qute waren die Typsicherheit, die in den Templates gewährleistet wird, und die Möglichkeit Kontrollstrukturen zu verwenden, um dynamische Inhalte zu rendern. Ein Beispiel hierfür ist im Anhang A.18: `terminauswahl.html` auf Seite xv zu sehen. Renarde wurde wegen des automatischen Transaktionsmanagements in Controller-Methoden verwendet, was bspw. eine einfache HTTP-Weiterleitung von POST-Anfragen auf GET-Anfragen ermöglicht.

Das Corporate Design der Anwendung wurde durch die Verwendung einer spezifischen CSS-Datei sichergestellt. Zusätzlich wurde das CSS-Framework Bootstrap für die allgemeine Gestaltung der Benutzeroberfläche eingesetzt.

Die Controller-Klassen, die die Logik der Benutzeroberfläche steuern, befinden sich in der View-Komponente. Diese Klassen sind für die Verarbeitung der Benutzereingaben und die Interaktion mit den Services verantwortlich. Die Controller verwenden die Renarde-Annotation `@Controller` und je Methode die `@Path`-Annotation, um die Pfade der Endpunkte zu definieren. Die Annotationen `@Produces` und `@Consumes` werden verwendet, um die unterstützten Medientypen zu definieren. Um die CRUD-Operationen zu implementieren, wurden die `@GET`, `@POST`, `@PUT` und `@DELETE`-Annotationen verwendet.

Screenshots der fertigen Benutzeroberfläche sind im Anhang A.19: Screenshots der Webanwendung auf Seite xvi zu finden. Diese Screenshots entsprechen den im Abschnitt 4.3 Entwurf der Benutzeroberfläche erstellten Mockups.

5.5 Implementierung der APIs

Die Implementierung der APIs in ENTON und ROBBE erfolgte in Quarkus als REST Services, die den Zalando RESTful API and Event Guidelines[11] entsprechen. Die Authentifizierung für die API-Zugriffe wurde über Basic-Auth realisiert, wobei die Zugangsdaten in der `application.properties` hinterlegt wurden, die im Anhang A.14: `application.properties` (Auszug) auf Seite xii zu finden ist.

Die Synchronisation der Daten zwischen ENTON und ROBBE wird dabei durch den Aufruf eines spezifischen Endpunkts in ROBBE ausgelöst. Dieser Endpunkt wird durch das in der AO verwendete Cron-Job-System *Enterprise Manager Cloud Control 13c* aufgerufen, um die Synchronisation auszulösen. Dieser Job wird stündlich ausgeführt und initiiert API-Aufrufe zu ENTON. Ein Ablauf dieser Synchronisation ist am Beispiel der Terminsynchronisation im Anhang A.10: Sequenzdiagramm für die stündliche Synchronisation der Termine auf Seite ix dargestellt.

Die Datenübertragung zwischen den Endpunkten erfolgt im JSON-Format. Im Falle eines Fehlers sendet die API eine entsprechende HTTP-Statusmeldung zurück.

5.6 Testen der Anwendung

Die Unit-Tests fokussierten sich auf die Modelle und die Geschäftslogik. Diese Tests wurden mit JUnit durchgeführt und nutzten Constructor-Injection in den Serviceklassen, um Mock-Objekte mit Mockito zu erstellen. Dadurch wurden die Tests von der Datenbank entkoppelt. Ein Beispiel eines solchen Tests ist im Anhang A.20: `TerminServiceSollte.java` (Auszug) auf Seite xvii zu finden.

Integrationstests kamen außerdem in den Bereichen Persistenz, Benutzeroberfläche und API zum Einsatz. Hierbei wurde die Dev-Services-Funktionalität von Quarkus verwendet. Diese nutzt intern Testcontainers, um beim Ausführen der Tests eine isolierte Datenbank in einem Docker-Container bereitzustellen. Zudem wurde als Test-Bibliothek RestAssured verwendet. RestAssured ist eine Java-Bibliothek, die für das Testen von REST-Services verwendet wird. Im Anhang A.21: `TerminControllerSollte.java` (Auszug) auf Seite xviii ist eine solcher Integrationstest am Beispiel des TerminController zu finden.

6 Abnahme- und Einführungsphase

In dieser Phase wurde das Deployment eingerichtet und das System gemäß dem Pflichtenheft überprüft. Die beteiligten Personen[12] führten umfassende Tests durch, um sicherzustellen, dass alle Anforderungen erfüllt wurden.

[11] https://opensource.zalando.com/restful-api-guidelines/

[12] Ein Ausbilder, ein Abteilungsleiter, ein Abteilungsdirektor, ein Mitglied der Mitarbeitervertretung, eine Mitarbeiterin der Personalabteilung.

6.1 Deployment der Anwendung

Nach der Implementierungsphase wurde die CI/CD-Pipeline mittels eines `Jenkinsfile` (siehe Anhang A.22: `Jenkinsfile` (Auszug) auf Seite xix) eingerichtet. Es wird für jede Änderung am `main`- oder `qs`-Branch ein neues Docker-Image gebaut und in das Artefakten-Repository *Artifactory* gepusht. Abhängig vom Branch wird der Container in der internen QS oder der Produktivumgebung auf dem externen Webserver deployt.

6.2 Abnahme durch den Fachbereich

Die Abnahme erfolgte in mehreren Schritten. Zunächst wurde die neue Anwendung ENTON sowie die Änderungen an ROBBE den beteiligten Personen vorgestellt. Anschließend trugen alle Beteiligten ihre derzeit freien Termine in ROBBE ein und ein Beispielprozess aus der Perspektive eines Bewerbers in ENTON wurde durchgespielt. Zudem wurde das Anstoßen der Termineinladung in ROBBE getestet.

Nach der Freigabe des letzten PRs wurde ein Jenkins-Build automatisch ausgelöst. Dieser deployte die Anwendung wie unter Abschnitt 4.7 Deployment beschrieben. Nach erfolgreicher Aktivierung des Cron-Jobs im Enterprise Manager Cloud Control 13c wurde die Synchronisation zwischen ENTON und ROBBE gestartet und die Anwendung ging in den Produktivbetrieb über.

7 Dokumentation

Die Dokumentation des Projekts wurde für verschiedene Zielgruppen erstellt. Die Zielgruppen sind Bewerber, die beteiligten Mitarbeiter und Entwickler.

7.1 Benutzerhandbuch für Bewerber

Ein Benutzerhandbuch für Bewerber wurde erstellt, das in leicht verständlicher Sprache die Nutzung der Anwendung erklärt. Es bietet eine Anleitung zur Nutzung der Anwendung und ist mit Screenshots angereichert. Dieses Handbuch ist als Portable Document Format (PDF)-Datei verfügbar und kann über die Hilfe-Sektion der Weboberfläche abgerufen werden. Ein Auszug ist im Anhang A.23: Benutzerhandbuch für Bewerber (Auszug) auf Seite xx zu finden.

7.2 Benutzerhandbuch für Mitarbeiter

Für die Mitarbeiter wurde ein separates Benutzerhandbuch erstellt. Es enthält detaillierte Informationen zur Nutzung der neuen Terminverwaltung in ROBBE. Dieses Handbuch ist ebenfalls als PDF-Datei verfügbar und ein Auszug ist im Anhang A.24: Benutzerhandbuch für Mitarbeiter (Auszug) auf Seite xxi zu finden.

7.3 Entwicklerdokumentation

Die Entwicklerdokumentation wurde umfassend gestaltet und enthält:

- *JavaDoc* für alle Klassen und Methoden (Anhang A.25: JavaDoc (Auszug) auf Seite xxii)
- Ein UML-Klassendiagramm, das die Struktur des Codes visualisiert (Anhang A.26: Klassendiagramm (Auszug) auf Seite xxiii)
- Eine Dokumentation des Datenbank-Tabellenmodells (Anhang A.27: Tabellenmodell (Auszug) auf Seite xxiii)
- Eine Dokumentation der REST-API (Anhang A.28: API-Dokumentation (Auszug) auf Seite xxiv)

- Eine `README`-Datei, die eine Übersicht über das Projekt, die lokale Einrichtung und die Verwendung der Anwendung bietet (Anhang A.29: `README.md` (Auszug) auf Seite xxv)

8 Fazit

Das Projekt ENTON wurde erfolgreich umgesetzt und stellt eine Verbesserung für die AO dar. Die erfolgreiche Implementierung und Abnahme legen den Grundstein für zukünftige Erweiterungen und Optimierungen.

8.1 Soll-/Ist-Vergleich

Im Projektverlauf wurden alle Muss-Anforderungen und die einzige Soll-Anforderung erfolgreich umgesetzt. Die Kann-Anforderungen wurden aufgrund von Zeitmangel nicht umgesetzt.

Darüber hinaus wurden folgende Abweichungen von der Projektplanung festgestellt:

- **Analysephase:** Dauerte eine Stunde länger als geplant, aufgrund einer detaillierteren Untersuchung der Anwendungsfälle.
- **Entwurfsphase:** Eine zusätzliche Stunde wurde für das Erstellen eines Sequenzdiagramms aufgewendet.
- **Abnahme- und Einführungsphase:** Eine Stunde eingespart durch den Verzicht auf Kubernetes.
- **Dokumentationsphase:** Zeitersparnis durch Generierung der API-Dokumentation mit OpenAPI und Swagger.

Projektphase	Geplant	Tatsächlich	Differenz
Analysephase	7 h	8 h	+1 h
Entwurfsphase	11 h	12 h	+1 h
Implementierungsphase	48 h	48 h	0 h
Abnahme und Einführung	7 h	6 h	-1 h
Dokumentation	7 h	6 h	-1 h
Gesamt	**80 h**	**80 h**	

Tabelle 3: Soll-/Ist-Vergleich

8.2 Lessons Learned

Die Projektumsetzung bot zahlreiche Lernmöglichkeiten. Die Kommunikation mit den Stakeholdern des Projektes war besonders wertvoll, da sie direktes Feedback und Einblicke in die tatsächlichen Anforderungen und Erwartungen lieferten. Dies ermöglichte es, das Projekt besser an die Bedürfnisse der Anwender anzupassen und mögliche Missverständnisse frühzeitig zu klären. Technische Aspekte wie TDD und Jakarta EE wurden vertieft und erweitert.

8.3 Ausblick

Die erfolgreiche Umsetzung des Projekts ENTON öffnet Möglichkeiten für zukünftige Erweiterungen. Insbesondere könnten die nicht umgesetzten Kann-Anforderungen aus dem Lastenheft (siehe Anhang A.4: Lastenheft auf Seite iii) wie Status-Benachrichtigungen per E-Mail, iCalendar-Export und der Dark Mode in Betracht gezogen werden. Diese Erweiterungen würden das Nutzererlebnis weiter verbessern und den Funktionsumfang von ENTON erweitern.

Literaturverzeichnis

Beck 2003
Beck, Kent: *Test-Driven Development: By Example*. Addison-Wesley, 2003 (The Addison-Wesley Signature Series)

Bogdan-Alexandru u. a. 2019
Bogdan-Alexandru, Andrei ; Casu-Pop, Andrei-Cosmin ; Sorin-Catalin, Gheorghe ; Boiangiu, Costin-Anton: A study on using waterfall and agile methods in software project management. In: *Journal of Information Systems & Operations Management* (2019), S. 125–135

Quarkus 2023
Quarkus: *What is Quarkus?* https://quarkus.io/about/. Version: 2023, Abruf: 28.09.2023

Schmitz 2018
Schmitz, David: *Vavr – Objekt-funktionale Programmierung leichtgemacht*. https://senacor.blog/vavr-objekt-funktionale-programmierung-leichtgemacht/. Version: 2018, Abruf: 29.09.2023

Schulz 2023
Schulz, Dr. C.: *Die MoSCoW Methode – Themen mit 4 Klassen priorisieren*. https://www.consulting-life.de/moscow-methode/. Version: 2023, Abruf: 27.09.2023

A Anhang

A.1 Detaillierte Zeitplanung

Analyse	**7h**
Durchführen der Ist-Analyse	2h
Durchführen der Wirtschaftlichkeitsanalyse und Amortisationsrechnung	1h
Ermitteln von Use-Cases inkl. Erstellung eines Anwendungsfall-Diagramms	1h
Erstellen des Lastenhefts (Unterstützung des Fachbereiches)	3h
Entwurf	**11h**
Entwerfen eines Verteilungsdiagramms	1h
Entwerfen eines Aktivitätsdiagramms	2h
Entwerfen der Benutzungsoberflächen inkl. Erstellung von Mock-Ups	1h
Planen der Architektur inkl. Erstellung eines Komponentendiagramms	2h
Entwerfen der Domäne inkl. Erstellung eines ER-Modells	2h
Erstellen des Pflichtenhefts	3h
Implementierung inkl. UI und Tests	**48h**
Erstellen des Java-Projekts	1h
Erstellen des Gradle-Builds inkl. der Anbindung an die statische Code-Analyse-Software	2h
Erstellen des Dockerfiles	1h
Implementieren der Domänenklassen	8h
Implementieren des Einreichens einer Bewerbung	6h
Implementieren der Terminabstimmung durch den Bewerber (Auswahl, Bestätigung, Stornierung)	8h
Implementieren des Anlegens von Terminen in ROBBE	8h
Implementieren des E-Mail-Versands in der bestehenden ROBBE-Anwendung	6h
Implementieren der REST-API in beiden Anwendungen inkl. Tests	8h
Abnahme und Einführung	**7h**
Code-Review durch erfahrenen Entwickler	3h
Abnahme durch den Fachbereich	1h
Continuous Integration & Continuous Deployment	2h
Einrichten des Jenkins-Builds	1h
Anlegen der Kubernetes-Dateien	1h
Deployment der Anwendung	1h
Dokumentation	**7h**
Erstellen der Benutzerhandbücher	2h
Erstellen der Entwicklerdokumentation mit JavaDoc, Klassendiagramm, Tabellenmodell, API-Dokumentation und README	5h
Gesamt	**80h**

A.2 Ressourcenplanung

Räumlichkeiten

- Ein Büro, in dem gearbeitet wird.
- Höhenverstellbarer Tisch mit ergonomischem Stuhl.

Hardware

- Ein Laptop mit einer Docking-Station.
- Ergonomische Eingabegeräte: Tastatur und Maus.
- Zwei 24-Zoll-Monitore für produktive Arbeitsumgebung.

Software

- Betriebssystem: Windows 10
- Build-Management: Gradle
- Containerisierung: Docker
- Continuous Delivery: Jenkins
- Datenbanksystem: MariaDB
- Datenbankverwaltungssoftware: DBeaver
- Entwicklungsumgebung: IntelliJ IDEA Ultimate
- Java-Testing: JUnit
- Mockup-Design: Balsamiq
- Modellierung: Draw.io
- Projektmanagement: Redmine
- Statische Code-Analyse: SonarQube
- Textsatz: MiKTeX (für LaTeX)
- UML-Modellierung: PlantUML
- Versionsverwaltung: Git

Personal

- Ein Azubi im Bereich Fachinformatik für Anwendungsentwicklung, der für die Projektumsetzung verantwortlich ist.
- Ein Anwendungsentwickler für das Review der Pull-Requests.
- Vier Mitarbeiter und der Ausbilder, die für die Definition der Anforderungen, dem Entwurf der Benutzeroberfläche und die Abnahme der Anwendung verantwortlich sind.

A.3 Use-Case-Diagramm

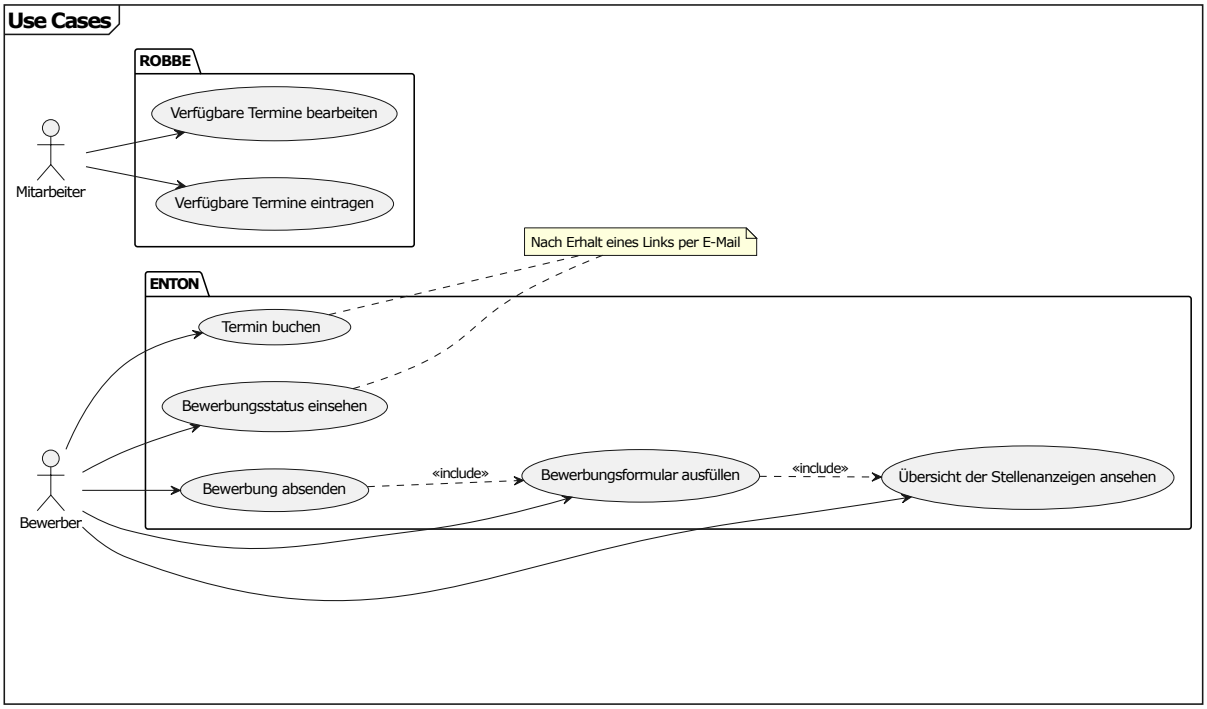


Abbildung 1: Umzusetzende Anwendungsfälle in den Anwendungen ENTON und ROBBE

A.4 Lastenheft

Für das Projekt ENTON werden folgende Anforderungen festgehalten, sortiert nach ihrer Wichtigkeit für den Bewerbungsprozess:

Muss-Anforderungen Die Anwendung muss...

1. ...die Möglichkeit bieten, sich auf Stellenangebote zu bewerben.
   Zentraler Bestandteil des Bewerbungsprozesses. Vom Ausbildungsleiter gefordert.
2. ...erfasste Bewerberdaten stündlich nach ROBBE übertragen.
   Wichtig für die Aktualität und Konsistenz der Bewerberdaten. Gewünscht von der Personalabteilung.
3. ...existierende Stellenangebote regelmäßig aus ROBBE übertragen.
   Erforderlich, um aktuelle Stellenangebote anzuzeigen. Von der Personalabteilung gefordert.
4. ...existierende Stellenangebote anzeigen können.
   Notwendig für Bewerber, um passende Stellenangebote zu finden. Vom Ausbildungsleiter gefordert.
5. ...die erfassten Daten von Bewerbern sicher speichern.
   Zum Schutz der personenbezogenen Daten. Vom Abteilungsdirektor gefordert.
6. ...den Status der Bewerbung für die Bewerber per Link anzeigen können.
   Bietet den Bewerbern Transparenz. Von der Personalabteilung gewünscht.
7. ...existierende Quellen regelmäßig aus ROBBE übertragen.
   Wichtig für die Auswertung der Bewerberquellen. Von der Personalabteilung gefordert.

8. . . . freie Termine regelmäßig aus ROBBE übertragen.
   Ermöglicht Bewerbern die Terminbuchung. Vom Ausbildungsleiter gefordert.
9. . . . die Möglichkeit bieten, Termine zu buchen, zu stornieren und zu verschieben.
   Erhöht die Flexibilität für Bewerber. Von der Personalabteilung gefordert.
10. . . . gebuchte Termine regelmäßig nach ROBBE übertragen.
    Stellt sicher, dass gebuchte Termine im System aktualisiert werden. Vom Ausbildungsleiter gefordert.
11. . . . in allen gängigen Webbrowsern lauffähig sein.
    Um eine breite Zugänglichkeit für alle Bewerber zu gewährleisten. Vom Ausbildungsleiter gefordert.
12. . . . für den sicheren Umgang mit Daten TLS-Verschlüsselung nutzen.
    Standardverfahren zur sicheren Datenübertragung. Vom Ausbildungsleiter gefordert.

Soll-Anforderungen Die Anwendung soll. . .

1. . . . eine mobile Ansicht für Smartphones und Tablets bieten.
   Erhöht die Zugänglichkeit für Bewerber. Von der Personalabteilung gewünscht.

Kann-Anforderungen Die Anwendung kann. . .

1. . . . per E-Mail über den Status der Bewerbung informieren.
   Erhöht die Transparenz für Bewerber. Von der Personalabteilung gewünscht.
2. . . . einen gebuchten Termin für externe Kalenderanwendungen mit dem Standard iCalendar exportieren.
   Ermöglicht eine bessere Terminplanung für Bewerber. Vom Ausbildungsleiter gewünscht.
3. . . . einen „Dark Mode“ anbieten.
   Komfortfunktion für Nutzer, die eine dunkle Benutzeroberfläche bevorzugen. Vom Ausbildungsleiter gewünscht.

Wird-nicht-Anforderungen Die Anwendung wird nicht. . .

1. . . . auf veralteten Browsern laufen (z. B. Internet Explorer).
   Fokus liegt auf der Unterstützung moderner Browser. Von allen Beteiligten entschieden.

A.5 Komponentendiagramm

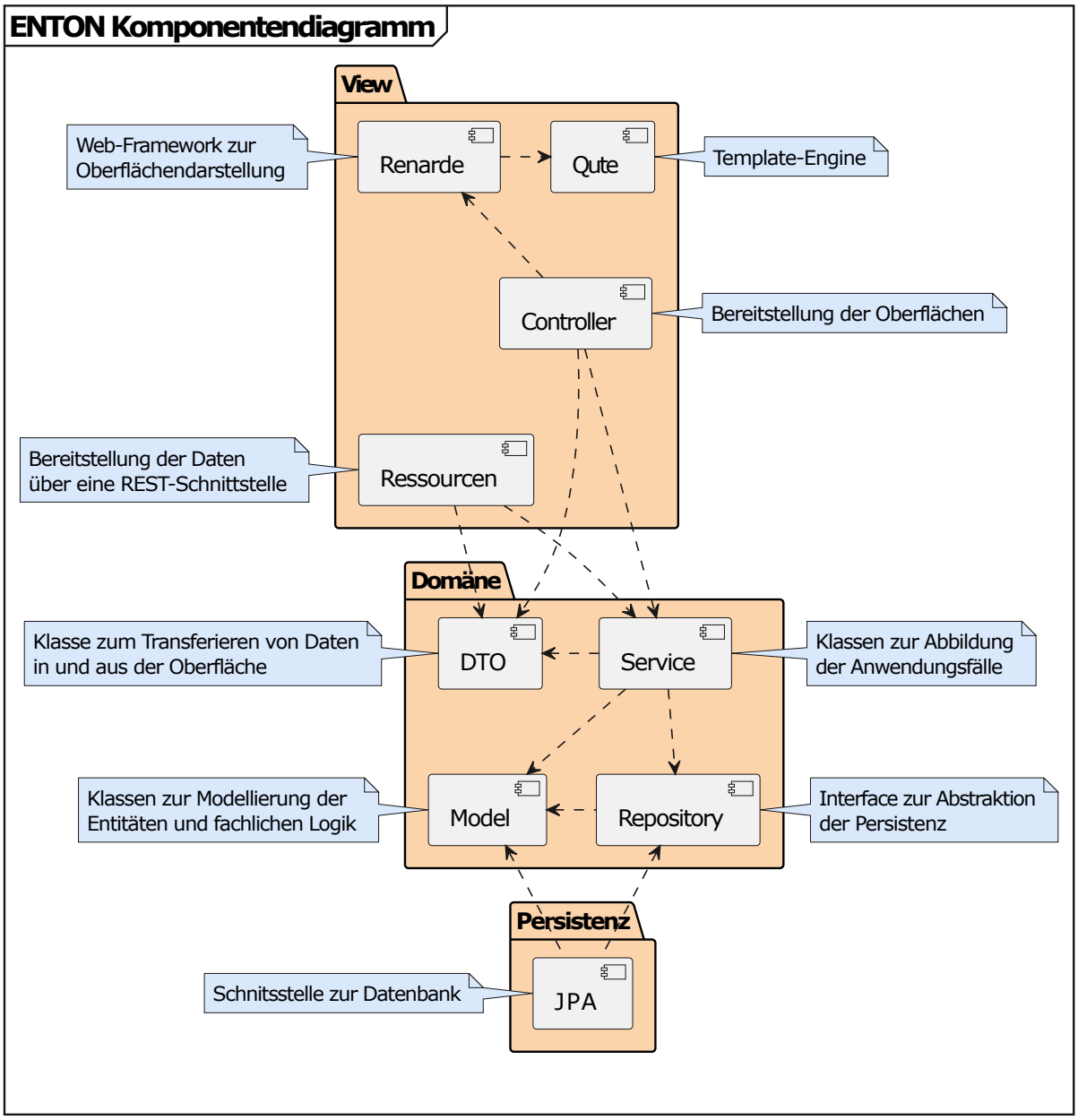


Abbildung 2: Komponentendiagramm der zu entwickelnden Anwendung ENTON

A.6 Mockups

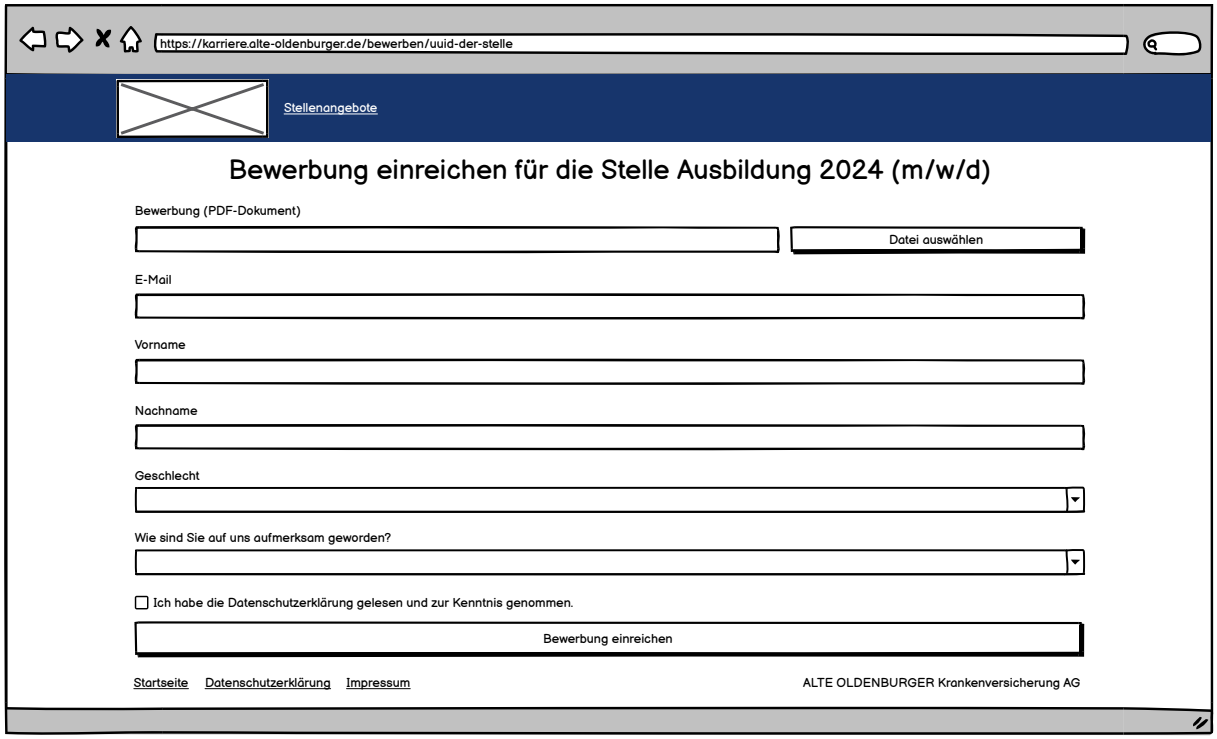


Abbildung 3: Mockup des Formulars zur Erfassung einer neuen Bewerbung

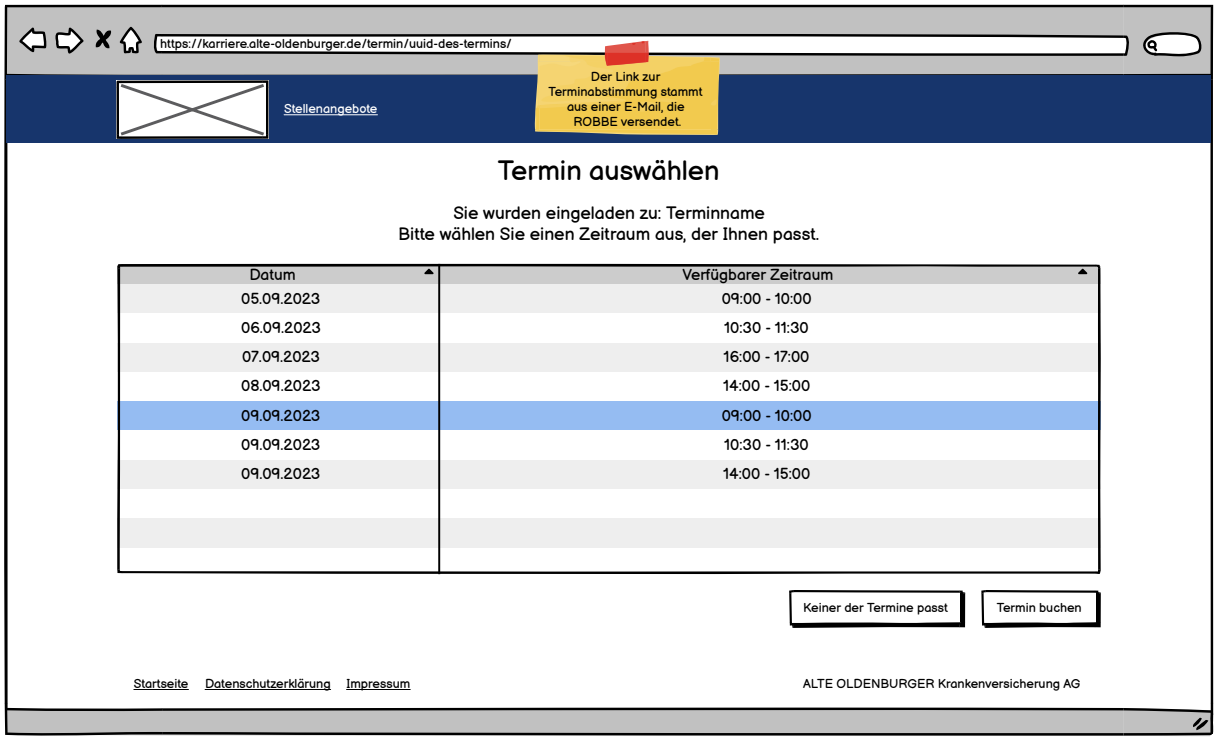


Abbildung 4: Mockup der Benutzeroberfläche zur Terminbuchung

A.7 Entity-Relationship-Model

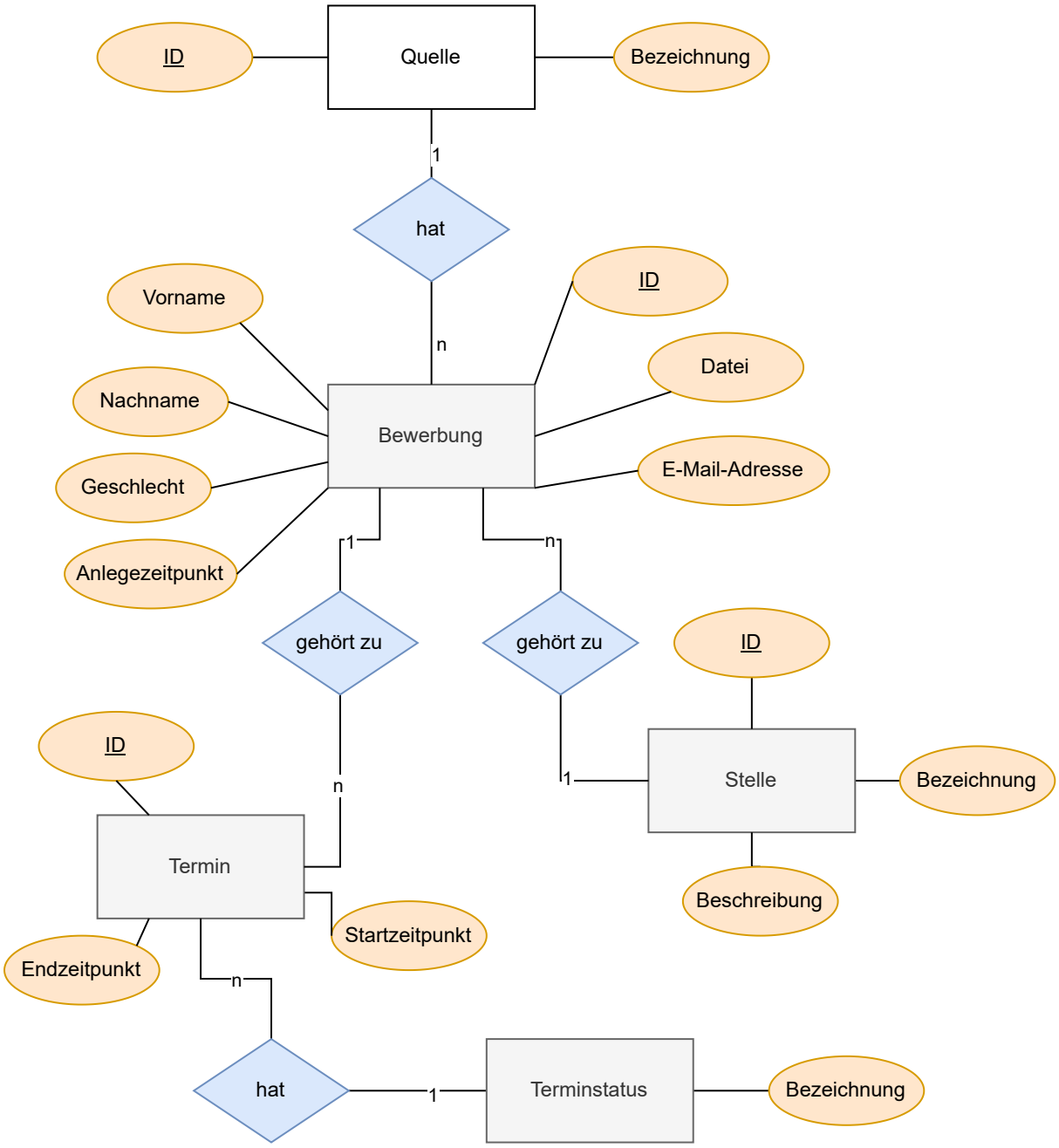


Abbildung 5: Entity-Relationship-Model (ENTON)

A.8 Aktivitätsdiagramm für den Bewerbungsprozess

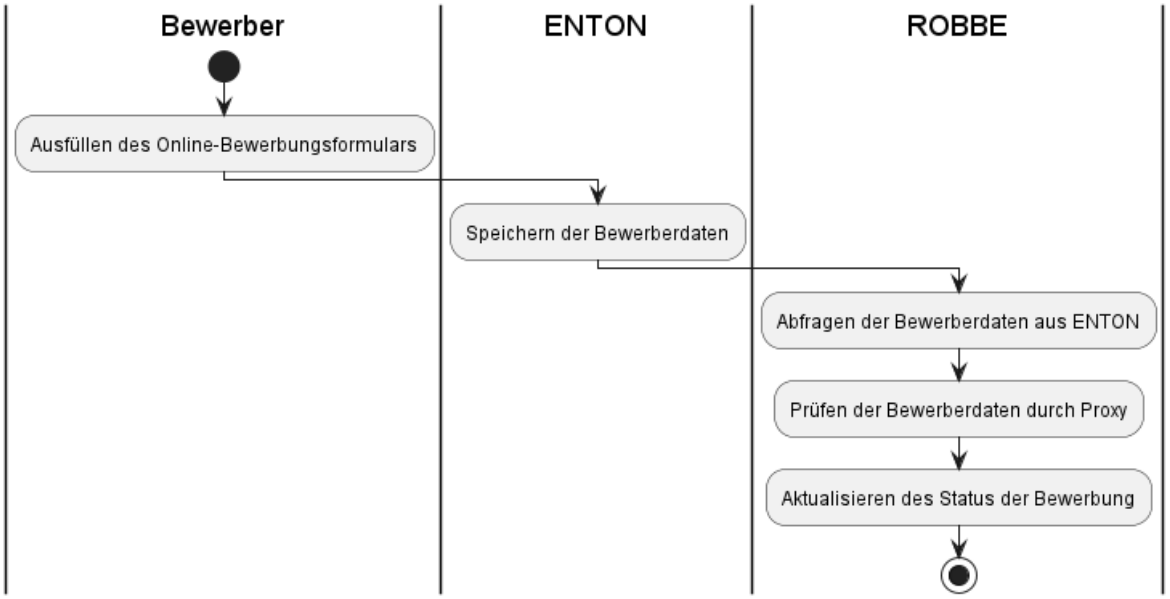


Abbildung 6: Aktivitätsdiagramm für den Bewerbungsprozess

A.9 Aktivitätsdiagramm für den Terminfindungs- und Buchungsprozess

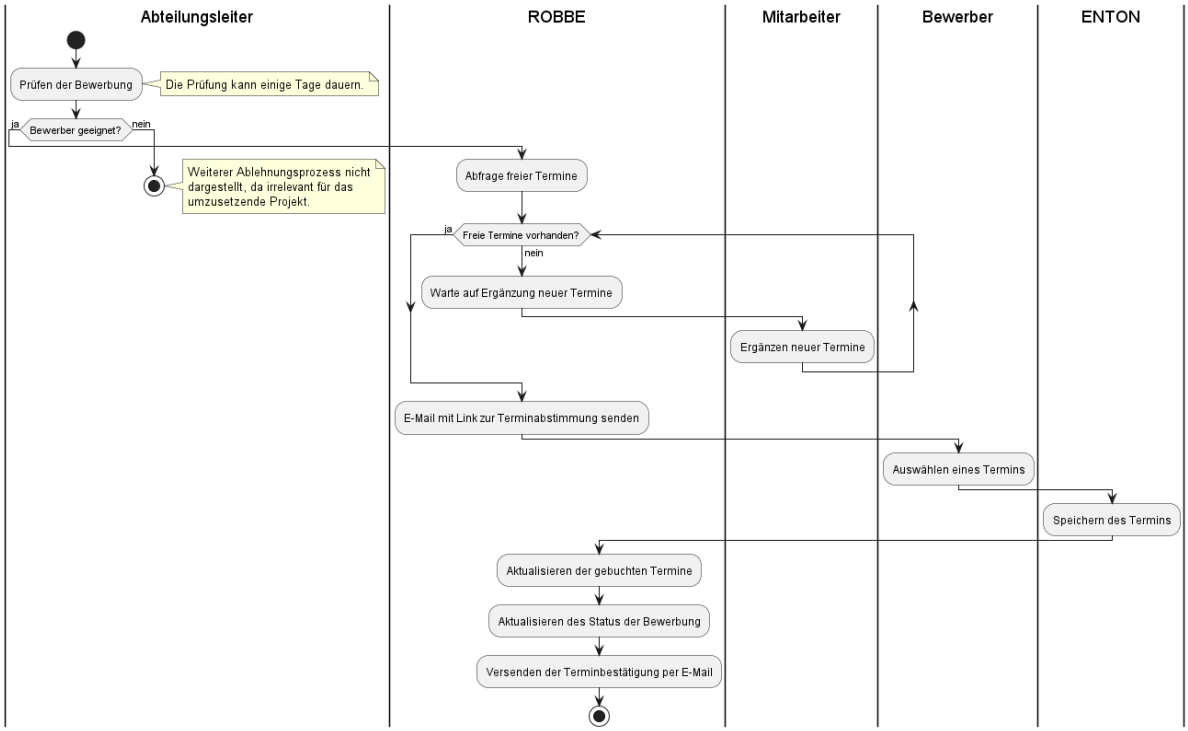


Abbildung 7: Aktivitätsdiagramm für den Terminfindungs- und Buchungsprozess

A.10 Sequenzdiagramm für die stündliche Synchronisation der Termine

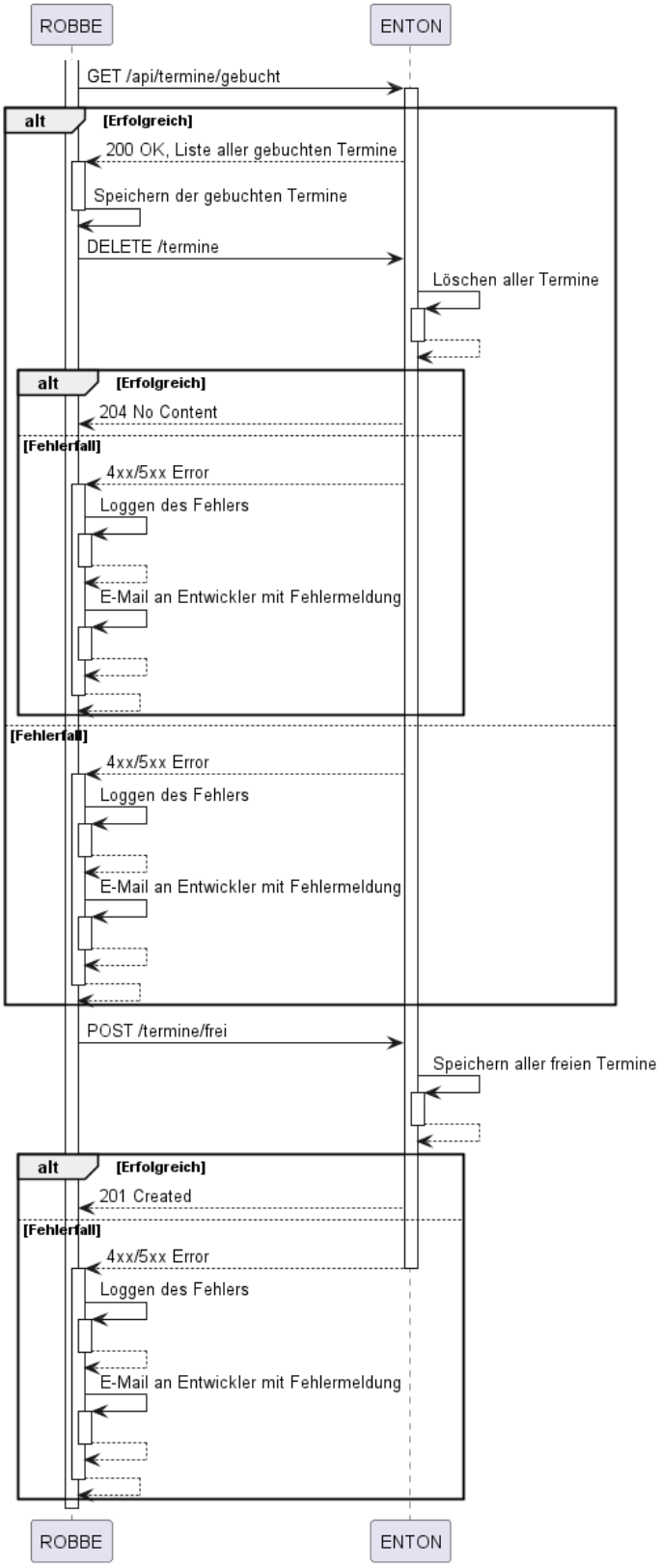


Abbildung 8: Sequenzdiagramm für die stündliche Synchronisation der Termine

A.11 Deploymentdiagramm

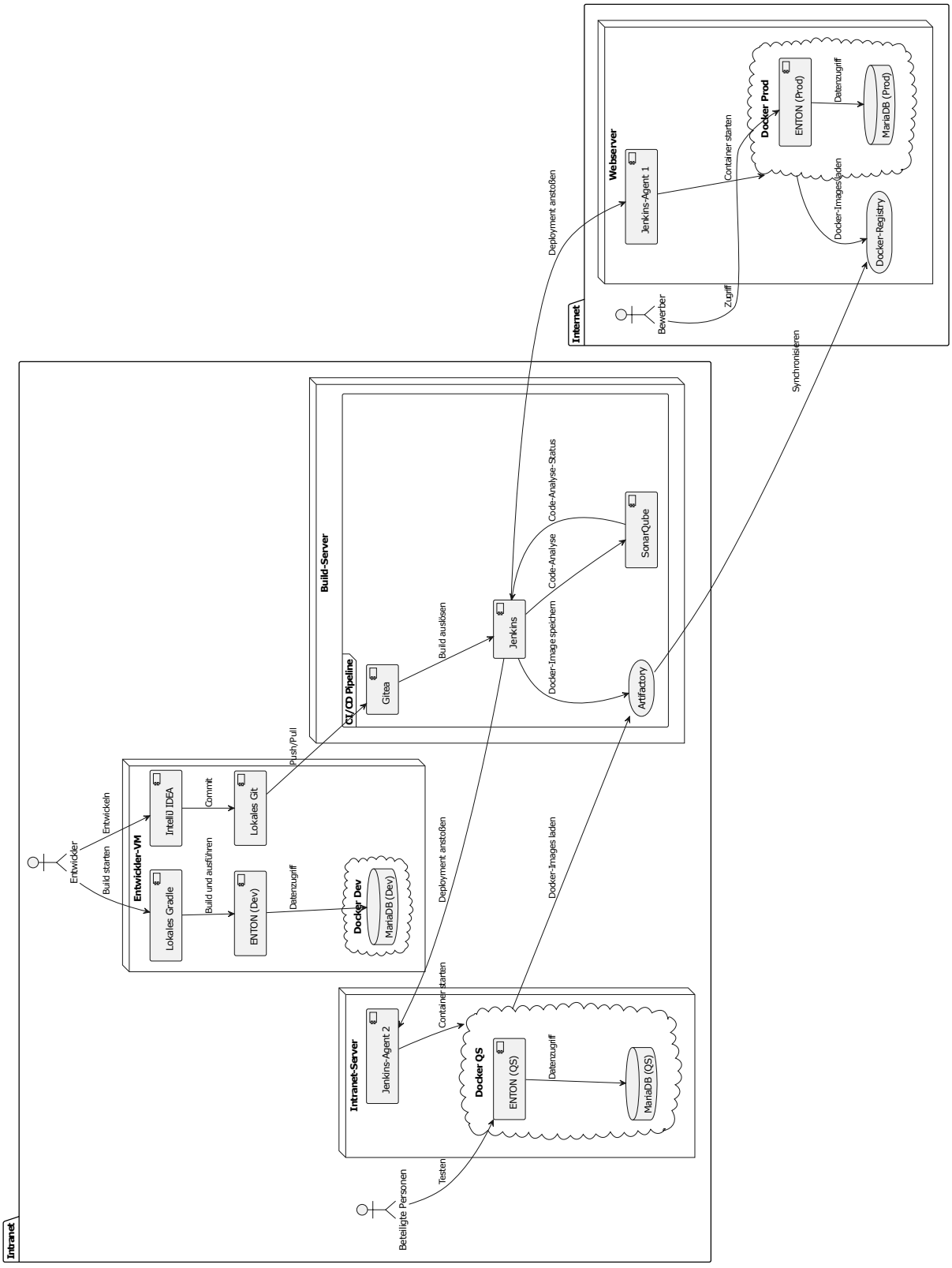


Abbildung 9: Deploymentdiagramm für ENTON

A.12 Pflichtenheft (Auszug)

1. **Plattform**
   1.1. Java (Version 17.0.2 mit Long-Term-Support) wird als Programmiersprache eingesetzt.
   1.2. Quarkus (Version 3.3.0) wird als Java-Framework eingesetzt, welches die Jakarta EE-Spezifikationen implementiert.
   1.3. Git (Version 2.35.1) zur Versionskontrolle des Quellcodes und Jenkins (Version 2.414.1) zur Automatisierung der Build- und Deployment-Prozesse.
   1.4. RESTEasy (Version 5.0.8.Final) zur Implementierung der REST-APIs.
   1.5. Docker (Version 24.0.5) dient zur Containerisierung und Vereinfachung des Deployments der gesamten Anwendung.
2. **Datenbank**
   2.1. MariaDB (Version 11) als relationales DBMS, ausgewählt für seine Leistungsfähigkeit und Zuverlässigkeit.
   2.2. Hibernate (Version 6), eine Implementierung von JPA, als Objekt-relationaler Mapper (ORM).
3. **Benutzeroberflächen**
   3.1. Die Benutzeroberflächen werden mit Qute als Template-Engine und Renarde als Web-Framework implementiert. Renarde agiert dabei als Aufsatz auf Qute und bietet zusätzliche Funktionalitäten, wie bspw. vereinfachte Formularverarbeitung.
   3.2. Bootstrap für das responsive Layout, ergänzt durch eine spezifische CSS-Datei, die das Corporate Design der AO umsetzt.
4. **Geschäftslogik**
   4.1. JUnit (Version 5) und Mockito (Version 5.5.0) für umfassende automatisierte Tests, um die Funktionalität und Zuverlässigkeit der Anwendung zu gewährleisten.
   4.2. Apache PDFBox (Version 3.0) für das Extrahieren von E-Mail-Adressen aus PDF-Dateien.
5. **Sicherheitsmaßnahmen**
   5.1. Einsatz von HTTPS zur Sicherstellung der Datenübertragungssicherheit.
   5.2. Basic-Auth für die Authentifizierung und Zugriffskontrolle der API.
6. **Dokumentation**
   6.1. Erstellung detaillierter Benutzerhandbücher für Bewerber und Mitarbeiter, die die Nutzung und Funktionalitäten der Anwendung erläutern.
   6.2. Entwicklerdokumentation mit JavaDoc, Klassendiagrammen, einem detaillierten Tabellenmodell und API-Dokumentation (visualisiert mit Swagger).
7. **Qualitätssicherung**
   7.1. Anwendung von TDD und die Verwendung von der statischen Codeanalysesoftware SonarQube, um die Fehlerfreiheit und die Einhaltung der Codequalitätsstandards zu gewährleisten.

A.13 `build.gradle` (Auszug)

```
dependencies {
    implementation enforcedPlatform("${quarkusPlatformGroupId}:${
        quarkusPlatformArtifactId}:${quarkusPlatformVersion}")
    implementation 'io.quarkus:quarkus-hibernate-orm',
            'io.quarkus:quarkus-resteasy-reactive',
            'io.quarkus:quarkus-resteasy-reactive-jackson',
            'io.quarkus:quarkus-resteasy-reactive-qute',
            'io.quarkus:quarkus-rest-client-reactive-jackson',
            'io.quarkus:quarkus-arc',
            'io.quarkus:quarkus-agroal',
            'io.quarkus:quarkus-jdbc-mariadb',
            'io.quarkus:quarkus-undertow',
            'io.quarkus:quarkus-elytron-security-properties-file',
            'io.quarkus:quarkus-security-jpa',
            'io.quarkus:quarkus-webjars-locator',
            'io.quarkiverse.renarde:quarkus-renarde:3.0.4',
            'org.webjars:bootstrap:5.3.2',
            'io.vavr:vavr:0.10.4',
            'org.apache.pdfbox:pdfbox:3.0.0'
    testImplementation 'io.quarkus:quarkus-junit5',
            'io.quarkus:quarkus-jacoco',
            'io.quarkiverse.renarde:quarkus-renarde-test:3.0.4',
            'io.rest-assured:rest-assured:5.3.2',
            'org.mockito:mockito-core:5.5.0',
            'org.assertj:assertj-core:3.24.2',
            'org.assertj:assertj-vavr:0.4.3'
    compileOnly 'org.projectlombok:lombok:1.18.28'
    annotationProcessor 'org.projectlombok:lombok:1.18.28'
}
```

Listing 1: `build.gradle` zur Konfiguration des Gradle-Builds (Auszug)

A.14 `application.properties` (Auszug)

```
# -- Allgemein --
quarkus.http.auth.basic=true
quarkus.http.auth.permission.admin-permission.paths=/api/*
quarkus.http.auth.permission.admin-permission.policy=authenticated
quarkus.security.users.embedded.enabled=true
quarkus.security.users.embedded.plain-text=true
quarkus.security.users.embedded.users.robbe=***
# -- Produktion --
%prod.quarkus.datasource.username=enton
%prod.quarkus.datasource.password=***
%prod.quarkus.datasource.jdbc.url=jdbc:mariadb://localhost:3306/enton
# -- Build --
%build.quarkus.hibernate-orm.database.generation=drop-and-create
%build.quarkus.hibernate-orm.sql-load-script=import.sql,importDev.sql
%build.quarkus.datasource.devservices.enabled=true
%build.quarkus.datasource.devservices.image-name=artifacts.ao/docker/mariadb:11
```

Listing 2: `application.properties` zur Konfiguration von Quarkus (Auszug)

A.15 `Dockerfile`

```
FROM artifacts.ao/docker-local/alteoldenburger/java-build:17
ENV LANG="de_DE.UTF-8" LANGUAGE="de_DE:de"
ENV JAVA_OPTIONS="-Dquarkus.http.host=0.0.0.0 -Djava.util.logging.manager=org.jboss.
    logmanager.LogManager"
COPY  build/quarkus-app/lib/ /deployments/lib/
COPY  build/quarkus-app/*.jar /deployments/
COPY  build/quarkus-app/app/ /deployments/app/
COPY  build/quarkus-app/quarkus/ /deployments/quarkus/
EXPOSE 8080
ENTRYPOINT [ "java", "-jar", "/deployments/quarkus-run.jar" ]
```

Listing 3: `Dockerfile` für die Erstellung des Image für ENTON

A.16 `Bewerbung.java` (Auszug)

```
@Entity
@NoArgsConstructor(access = AccessLevel.PRIVATE)
@AllArgsConstructor(access = AccessLevel.PRIVATE)
public class Bewerbung extends Entitaet {
    @ManyToOne private Stelle stelle;
    @ManyToOne private Quelle quelle;
    ...

    public static Validation<Meldungen, Bewerbung> erzeuge(
            Stelle stelle, Datei datei, Vorname vorname, Nachname nachname,
            Geschlecht geschlecht, Email emailAdresse, Quelle quelle) {
        return Validation.combine(
                        validiereAttribut(stelle, Meldung.STELLE_LEER),
                        validiereAttribut(datei, Meldung.DATEI_UNGUELTIG),
                        Validation.valid(vorname), Validation.valid(nachname),
                        Validation.valid(geschlecht), Validation.valid(emailAdresse),
                        Validation.valid(quelle), Validation.valid(LocalDateTime.now()))
                .ap(Bewerbung::new)
                .mapError(Meldungen::aus);
    }
}
```

Listing 4: Entitätsklasse `Bewerbung.java` (Auszug)

A.17 `BewerbungService.java`

```
public class BewerbungService extends Service {
    @Inject BewerbungRepository bewerbungRepository;
    @Inject StelleRepository stelleRepository;
    @Inject QuelleRepository quelleRepository;

    private <T> Validation<Meldungen, T> validiereUndErzeuge(
            String formularfeld, Function<String, Validation<Meldungen, T>>
                erzeugeMethode) {
        return formularfeld == null || formularfeld.isBlank()
                ? Validation.valid(null)
                : erzeugeMethode.apply(formularfeld);
    }

    public Option<String> sucheEmailInPdf(byte[] pdfInhalt) {
```

```
        return Try.of(() -> Loader.loadPDF(pdfInhalt))
                .mapTry(pdf -> new PDFTextStripper().getText(pdf))
                .mapTry(gefundenerText -> Pattern
                        .compile("[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}")
                        //RegEx für E-Mails nach RFC 5322
                        .matcher(gefundenerText))
                .filter(Matcher::find)
                .map(Matcher::group)
                .toOption();
    }

    @Transactional
    public Validation<Meldungen, Bewerbung> erzeuge(NeueBewerbung neueBewerbung) {
        if (neueBewerbung.getDatenschutzhinweiseGelesen() == null) {
            return Validation.invalid(Meldungen.erzeugeFehler(
                    Meldung.DATENSCHUTZHINWEISE_NICHT_GELESEN));
        }
        var stelle = findeEntitaet(neueBewerbung.getStelle(),
                stelleRepository::findeMit, Meldung.STELLE_EXISTIERT_NICHT);
        var bewerbung = Datei.erzeuge(neueBewerbung.getDatei());
        Validation<Meldungen, Quelle> optionaleQuelle = neueBewerbung.getQuelle()
                .isBlank()
                ? Validation.valid(null)
                : findeEntitaet(neueBewerbung.getQuelle(), quelleRepository::findeMit,
                Meldung.QUELLE_EXISTIERT_NICHT);
        if (neueBewerbung.getEmail().isBlank()) {
            sucheEmailInPdf(neueBewerbung.getDatei())
                    .peek(neueBewerbung::setEmailAusPdf);
        }
        return Validation.combine(stelle,
                        bewerbung,
                        validiereUndErzeuge(neueBewerbung.getVorname(),
                                Vorname::erzeuge),
                        validiereUndErzeuge(neueBewerbung.getNachname(),
                                Nachname::erzeuge),
                        validiereUndErzeuge(neueBewerbung.getGeschlecht(),
                                Geschlecht::erzeuge),
                        validiereUndErzeuge(neueBewerbung.getEmail(), Email::erzeuge),
                        optionaleQuelle)
                .ap(Bewerbung::erzeuge)
                .mapError(Meldungen::aus)
                .flatMap(Function.identity())
                .peek(bewerbungRepository::speichere);
    }

    public List<GespeicherteBewerbung> findeAlle() {
        return bewerbungRepository.findeAlle()
                .stream()
                .map(bewerbung -> new GespeicherteBewerbung(...))
                .toList();
    }

    public Validation<Meldungen, Bewerbung> findeMit(String bewerbungSchluessel) {
        return findeEntitaet(bewerbungSchluessel, bewerbungRepository::findeMit,
                Meldung.BEWERBUNG_EXISTIERT_NICHT);
    }
```

```
70
71      public Validation<Meldungen, GefundeneEmail> setzeGefundeneEmail(
72              GefundeneEmail gefundeneEmail) {
73          var bewerbungSchluessel = gefundeneEmail.getBewerbungSchluessel();
74          var emailAusPdf = gefundeneEmail.getEmail();
75          return Validation.combine(
76                          Email.erzeuge(emailAusPdf),
77                          findeEntitaet(bewerbungSchluessel, bewerbungRepository::findeMit,
78                                  Meldung.BEWERBUNG_EXISTIERT_NICHT))
79                  .ap((valideEmail, valideBewerbung) -> {
80                      valideBewerbung.setEmail(valideEmail);
81                      bewerbungRepository.speichere(valideBewerbung);
82                      return gefundeneEmail;
83                  })
84                  .mapError(Meldungen::aus);
85      }
86  }
```

Listing 5: Service-Klasse `BewerbungService.java`

A.18 `terminauswahl.html`

```
1  {#include includes/main}
2  {#title}Termin auswählen{/title}
3  <section class="container mt-5">
4      <div class="row">
5          <div class="col">
6              <h1 class="fw-bold">Termin auswählen</h1>
7              <h3>Bitte wählen Sie aus der folgenden Liste einen Termin, der Ihnen passt.
                   </h3>
8              <p>Sie wurden eingeladen zu <strong>{termin.bezeichnung}</strong>.<br/>
9                  Sie haben sich auf die Stelle <strong>{bewerbung.stelle.bezeichnung}
10                 </strong> beworben.</p>
11         </div>
12     </div>
13     <form accept-charset="UTF-8" action="{uri:TerminController.terminBuchen}"
14           enctype="multipart/form-data" method="post">
15         <fieldset>
16             <legend>Verfügbare Termine</legend>
17             {#authenticityToken/}
18             <table class="table table-striped">
19                 <thead>
20                 <tr>
21                     <th>Datum</th>
22                     <th>Zeitraum</th>
23                     <th>Auswählen</th>
24                 </tr>
25                 </thead>
26                 <tbody>
27                 {#for termin in termine}
28                 <tr class="clickable-row" data-radio-id="radio-{termin.schluessel}">
29                     <td>{termin.datum}</td>
30                     <td>{termin.start} - {termin.ende} Uhr</td>
31                     <td><input id="radio-{termin.schluessel}" name="gewaehlterTermin"
32                                required="required" type="radio"
33                                value="{termin.schluessel}"></td>
34                 </tr>
```

```
                {/for}
                </tbody>
            </table>
            <button class="btn btn-secondary" type="submit" value="keinTerminPasst">
                Keiner der Termine passt</button>
            <button class="btn btn-primary float-end" type="submit" value="buchen">
                Termin buchen</button>
        </fieldset>
    </form>
</section>
{/include}
```

Listing 6: Qute-Template `terminauswahl.html`

A.19 Screenshots der Webanwendung

Abbildung 10: Benutzeroberfläche der Eingabemaske zur Erfassung einer neuen Bewerbung

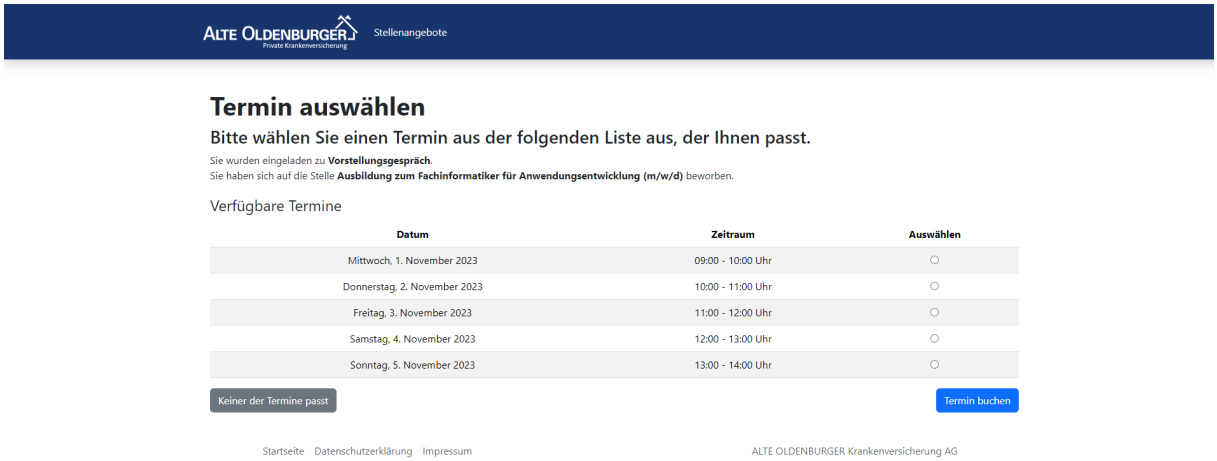


Abbildung 11: Benutzeroberfläche zur Auswahl eines Termins

A.20 `TerminServiceSollte.java` (Auszug)

```
1  @DisplayName("TerminService sollte")
2  class TerminServiceSollte {
3      private static final Termin FREIER_TERMIN = Termin.erzeuge("", DATUM, START, ENDE,
           Terminstatus.FREI, STELLE).get();
4      ...
5      private TerminService sut;
6      private TerminRepository terminRepository;
7
8      @BeforeEach
9      void setup() {
10         terminRepository = mock(TerminRepository.class);
11         sut = new TerminService(terminRepository);
12     }
13
14     @Test
15     @DisplayName("alle gespeicherten Termine finden")
16     void test01() {
17         when(terminRepository.findeAlle()).thenReturn(List.of(FREIER_TERMIN,
18                 GEBUCHTER_TERMIN));
19         var ergebnis = sut.findeAlle();
20         assertAll(() -> assertThat(ergebnis).isNotEmpty(),
21                 () -> assertThat(ergebnis).containsExactlyInAnyOrder(FREIER_TERMIN_DTO,
22                         GEBUCHTER_TERMIN_DTO));
23     }
24
25     @Test
26     @DisplayName("alle Termine speichern")
27     void test02() {
28         sut.speichereAlle(List.of(FREIER_TERMIN, GEBUCHTER_TERMIN));
29         verify(terminRepository).speichereAlle(List.of(FREIER_TERMIN,
30                 GEBUCHTER_TERMIN));
31     }
32
33     @Test
34     @DisplayName("einen freien Termin buchen")
35     void test03() {
```

```
36          when(terminRepository.findeMit(FREIER_TERMIN_SCHLUESSEL)).thenReturn(Option.of(
                FREIER_TERMIN));
37          var ergebnis = sut.bucheTermin(ZU_BUCHENDER_TERMIN);
38          assertAll(() -> org.assertj.vavr.api.VavrAssertions.assertThat(ergebnis)
39                          .isValid(),
40                  () -> assertThat(ergebnis.get().getStatus()).isEqualTo(
41                          Terminstatus.GEBUCHT));
42      }
43
44      @Test
45      @DisplayName("einen nicht existierenden Termin nicht buchen")
46      void test04() {
47          when(terminRepository.findeMit(any(Schluessel.class))).thenReturn(
48                  Option.none());
49          var ergebnis = sut.bucheTermin(new ZuBuchenderTermin(
50                  UUID.randomUUID().toString(), UUID.randomUUID().toString(),
51                  Aktion.ANWENDEN.toString()));
52          assertAll(() -> assertThat(ergebnis.isInvalid()).isTrue(),
53                  () -> assertThat(ergebnis.getError()).isEqualTo(Meldungen.erzeugeFehler(
54                          Meldung.TERMIN_EXISTIERT_NICHT)));
55      }
56      ...
57  }
```

Listing 7: Test der Klasse `TerminService.java`

A.21 `TerminControllerSollte.java` (Auszug)

```
1   @QuarkusTest
2   @DisplayName("TerminController sollte")
3   public class TerminControllerSollte {
4       @Test
5       @DisplayName("bei gültiger Bewerbung eine Terminseite anzeigen")
6       public void test01() {
7           given()
8                   .when()
9                   .get("/termin/{bewerbungsschluessel}",
10                          "a243c4ee-fa42-4ad8-8784-076e0687dc6a")
11                  .then()
12                  .statusCode(200)
13                  .contentType(ContentType.HTML)
14                  .body(contains("Termin auswählen"));
15      }
16
17      @Test
18      @DisplayName("bei nicht vorhandener Bewerbung auf die Startseite weiterleiten und
            eine Fehlermeldung anzeigen")
19      public void test02() {
20          given()
21                  .when()
22                  .get("/termin/{bewerbungsschluessel}",
23                          UUID.randomUUID().toString())
24                  .then()
25                  .statusCode(302)
26                  .header("Location", equalTo("/startseite"))
27                  .body(contains("Die Bewerbung existiert nicht!"));
28      }
```

```
29      ...
30
31      @Test
32      @DisplayName("einen Termin erfolgreich buchen")
33      public void test04() {
34          given()
35                  .formParam("bewerbungsschluessel",
36                         "a243c4ee-fa42-4ad8-8784-076e0687dc6a")
37                  .formParam("schluessel", "8684e2c0-be61-4027-923d-d4532a42cdf4")
38                  .formParam("aktion", "buchen")
39                  .when()
40                  .post("/termin")
41                  .then()
42                  .statusCode(201)
43                  .header("Location", equalTo("/startseite"))
44                  .body(contains("Der Termin wurde erfolgreich gebucht! Sie erhalten in
45                         Kürze eine Bestätigung ihres Termins per E-Mail."));
46      }
47      ...
48  }
```

Listing 8: Integratiosntest der Klasse `TerminController.java`

A.22 `Jenkinsfile` (Auszug)

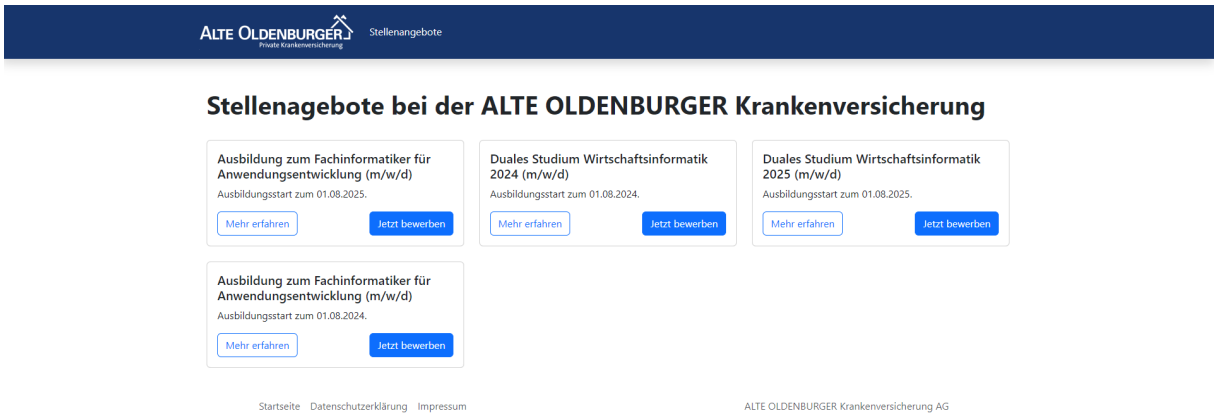
```
1       stages {
2           stage('Build') {
3               steps {
4                   checkout scm
5                   sh 'chmod u+x gradlew'
6                   sh './gradlew build check'
7                   aosonar.analysiereUndBreche(this)
8               }
9           }
10          stage('Erstelle Dockerimage') {
11              when {
12                  anyOf {
13                      branch 'main'
14                      branch 'qs'
15                  }
16              }
17              steps {
18                  erstelleDockerImage(imageName: 'ao-enton', tag: ${env.BRANCH_NAME})
19              }
20          }
21          stage('Deployment auf QS') {
22              when {
23                  branch 'qs'
24              }
25              agent {
26                  label 'Jenkins-Agent-2'
27              }
28              steps {
29                  deployeDockerImage(imageName: 'ao-enton', tag: qs,
30                  containerName: 'ao-enton', port: 8080)
31              }
32          }
```

```
33        stage('Deployment auf Prod') {
34            when {
35                branch 'main'
36            }
37            agent {
38                label 'Jenkins-Agent-1'
39            }
40            steps {
41                deployeDockerImage(imageName: 'ao-enton', tag: main,
42                containerName: 'ao-enton', port: 8080)
43            }
44        }
45    }
46    post {
47        failure {
48            script {
49                aomail.versendeFehlerMail(this)
50            }
51        }
52    }
```

Listing 9: Auszug aus dem `Jenkinsfile`

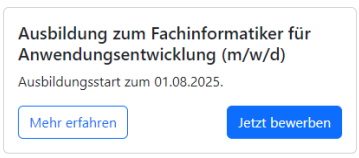
A.23 Benutzerhandbuch für Bewerber (Auszug)

Stellenangebote



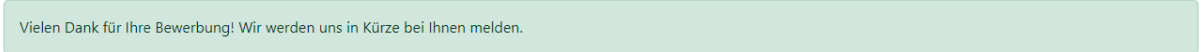
Unsere Startseite ist zugleich die Übersicht der aktuellen Stellenangebote.

Für weiterführende Informationen zu einer Stelle klicken Sie bitte auf den **Mehr erfahren**-Button. Haben Sie eine passende Stelle gefunden, nutzen Sie den **Jetzt bewerben**-Button, um mit dem Bewerbungsprozess zu starten.

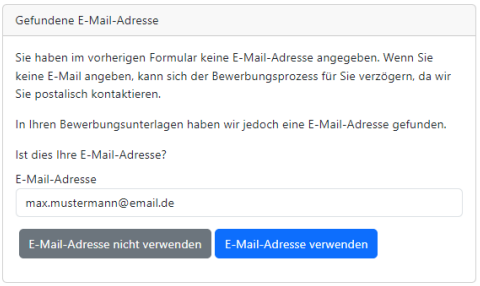


Nach Einreichung der Bewerbung

Nach Einreichung Ihrer Bewerbung, sehen Sie folgende Meldung über der Startseite:



Sollten Sie in unserem Bewerbungsformular keine E-Mail-Adresse angegeben haben, aber in Ihren Bewerbungsunterlagen, sehen Sie zusätzlich zur vorherigen Meldung eine weitere Meldung. Sie haben hier die Möglichkeit die gefundene E-Mail-Adresse zu verwenden, sodass wir Sie per E-Mail kontaktieren können. Alternativ haben Sie die Möglichkeit, die E-Mail-Adresse im Feld zu korrigieren. Sollten Sie eine Kontaktaufnahme per Post wünschen, wählen Sie einfach **E-Mail-Adresse nicht verwenden**.

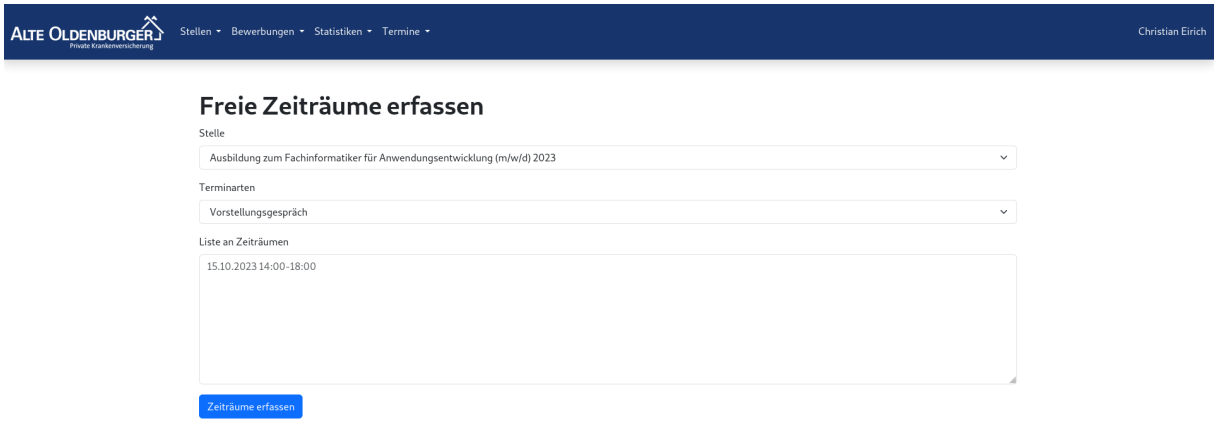


A.24 Benutzerhandbuch für Mitarbeiter (Auszug)

Zugriff auf das Terminplanungssystem

1. Melden Sie sich wie gewohnt im Bewerbungsmanagementsystem ROBBE an.
2. Gehen Sie im Hauptmenü mit der Maus über den Punkt **Termine** und wählen Sie dann den Unterpunkt **Freie Zeiträume erfassen**.

Erfassen neuer Zeiträume



1. Wählen Sie aus dem Dropdown-Menü die Stelle aus, für die Sie Termine anbieten möchten.
2. Wählen Sie die Art des Termins, beispielsweise *Vorstellungsgespräch*.
3. Tragen Sie die freien Zeiträume in das dafür vorgesehene Textfeld ein. Formatieren Sie die Eingabe im Format `TT.MM.JJJJ HH:MM-HH:MM`, z. B. `15.10.2023 14:00-18:00`.
4. Klicken Sie auf **Zeiträume erfassen**, um die eingegebenen Zeiten zu speichern.

Wöchentliche Aktualisierung

1. Es ist erforderlich, dass Sie Ihre verfügbaren Zeiträume wöchentlich aktualisieren, um eine effiziente Terminplanung zu gewährleisten.
2. Sollten sich Änderungen an bereits erfassten Zeiträumen ergeben, können Sie diese im Hauptmenü unter **Termine** und dann **Erfasste Zeiträume bearbeiten** ändern.

A.25 JavaDoc (Auszug)

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method	Description
io.vavr.control.Validation<Meldungen,Bewerbung>	erzeuge(NeueBewerbung neueBewerbung)	Erstellt eine neue Bewerbung und speichert sie in der Datenbank.
List<GespeicherteBewerbung>	findeAlle()	Ruft alle gespeicherten Bewerbungen aus der Datenbank ab.
io.vavr.control.Validation<Meldungen,Bewerbung>	findeMit(String bewerbungSchluessel)	Sucht eine Bewerbung in der Datenbank anhand ihres eindeutigen Schlüssels.
io.vavr.control.Validation<Meldungen,GefundeneEmail>	setzeGefundeneEmail(GefundeneEmail gefundeneEmail)	Aktualisiert die E-Mail-Adresse einer Bewerbung in der Datenbank.
io.vavr.control.Option<String>	sucheEmailInPdf(byte[] pdfInhalt)	Durchsucht den Inhalt eines PDF-Dokuments nach einer E-Mail-Adresse.

Methods inherited from class net.aokv.enton.domain.service.Service

findeEntitaet, findeMit

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Method Details

sucheEmailInPdf

public io.vavr.control.Option<String> sucheEmailInPdf(byte[] pdfInhalt)

Durchsucht den Inhalt eines PDF-Dokuments nach einer E-Mail-Adresse. Verwendet RegEx zur Identifizierung der E-Mail.

Parameters:
pdfInhalt - Byte-Array, das den Inhalt des PDF-Dokuments enthält.

Returns:
Eine Option, die die gefundene E-Mail-Adresse enthält, oder None, wenn keine gefunden wurde.

erzeuge

public io.vavr.control.Validation<Meldungen,Bewerbung> erzeuge(NeueBewerbung neueBewerbung)

Erstellt eine neue Bewerbung und speichert sie in der Datenbank. Diese Methode führt auch Validierungen durch und setzt die E-Mail-Adresse aus dem PDF, falls nicht angegeben.

Parameters:
neueBewerbung - Das DTO, das die Daten für die neue Bewerbung enthält.

Returns:
Eine Validation-Instanz, die entweder die erfolgreich gespeicherte Bewerbung oder eine Liste von Meldungen enthält.

findeAlle

public List<GespeicherteBewerbung> findeAlle()

Ruft alle gespeicherten Bewerbungen aus der Datenbank ab.

Returns:
Eine Liste von GespeicherteBewerbung-DTOs, die die Daten aller gespeicherten Bewerbungen enthalten.

findeMit

public io.vavr.control.Validation<Meldungen,Bewerbung> findeMit(String bewerbungSchluessel)

Sucht eine Bewerbung in der Datenbank anhand ihres eindeutigen Schlüssels.

Parameters:
bewerbungSchluessel - Der eindeutige Schlüssel der Bewerbung als String.

Returns:
Eine Validation-Instanz, die entweder die gefundene Bewerbung oder eine Liste von Meldungen enthält.

Abbildung 12: Mit JavaDoc erstellte Dokumentation der Klasse BewerbungService (Auszug)

A.26 Klassendiagramm (Auszug)

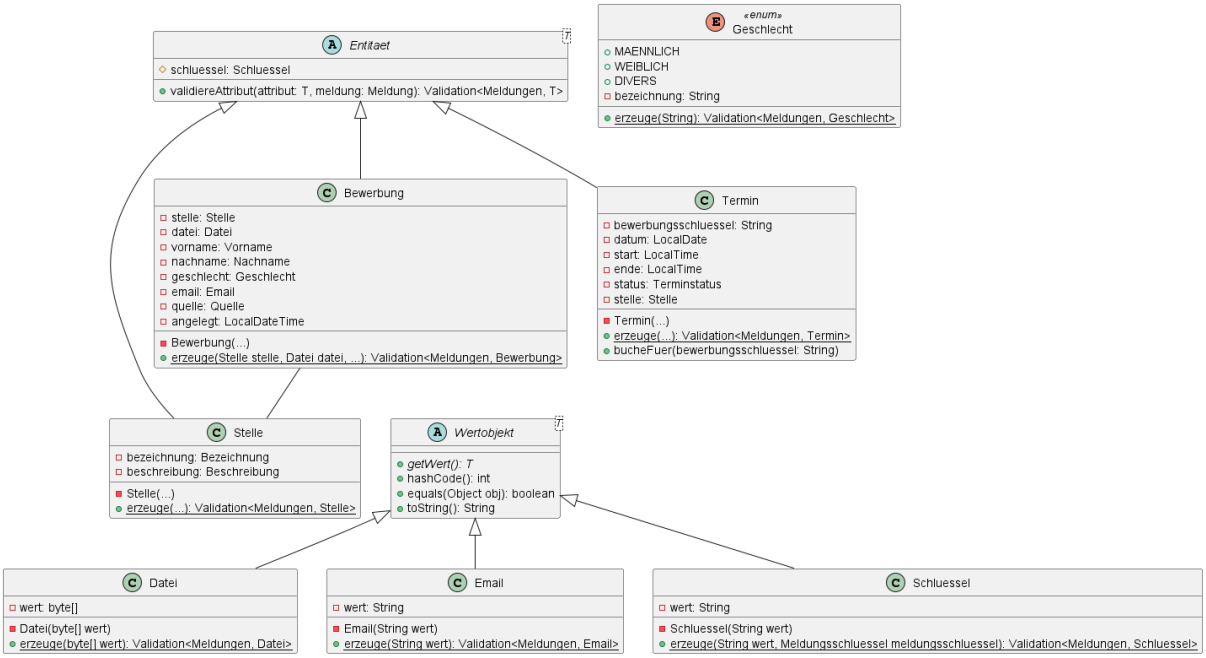


Abbildung 13: Klassendiagramm der Entitäten und Wertobjekte (gekürzter Auszug)

A.27 Tabellenmodell (Auszug)

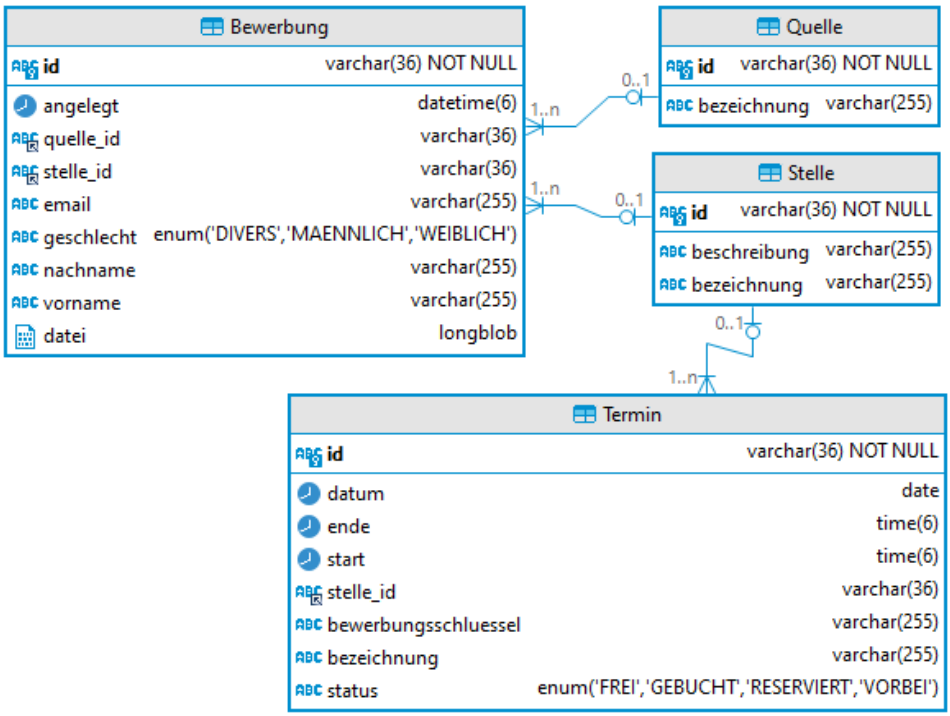


Abbildung 14: Tabellenmodell der Datenbank

A.28 API-Dokumentation (Auszug)

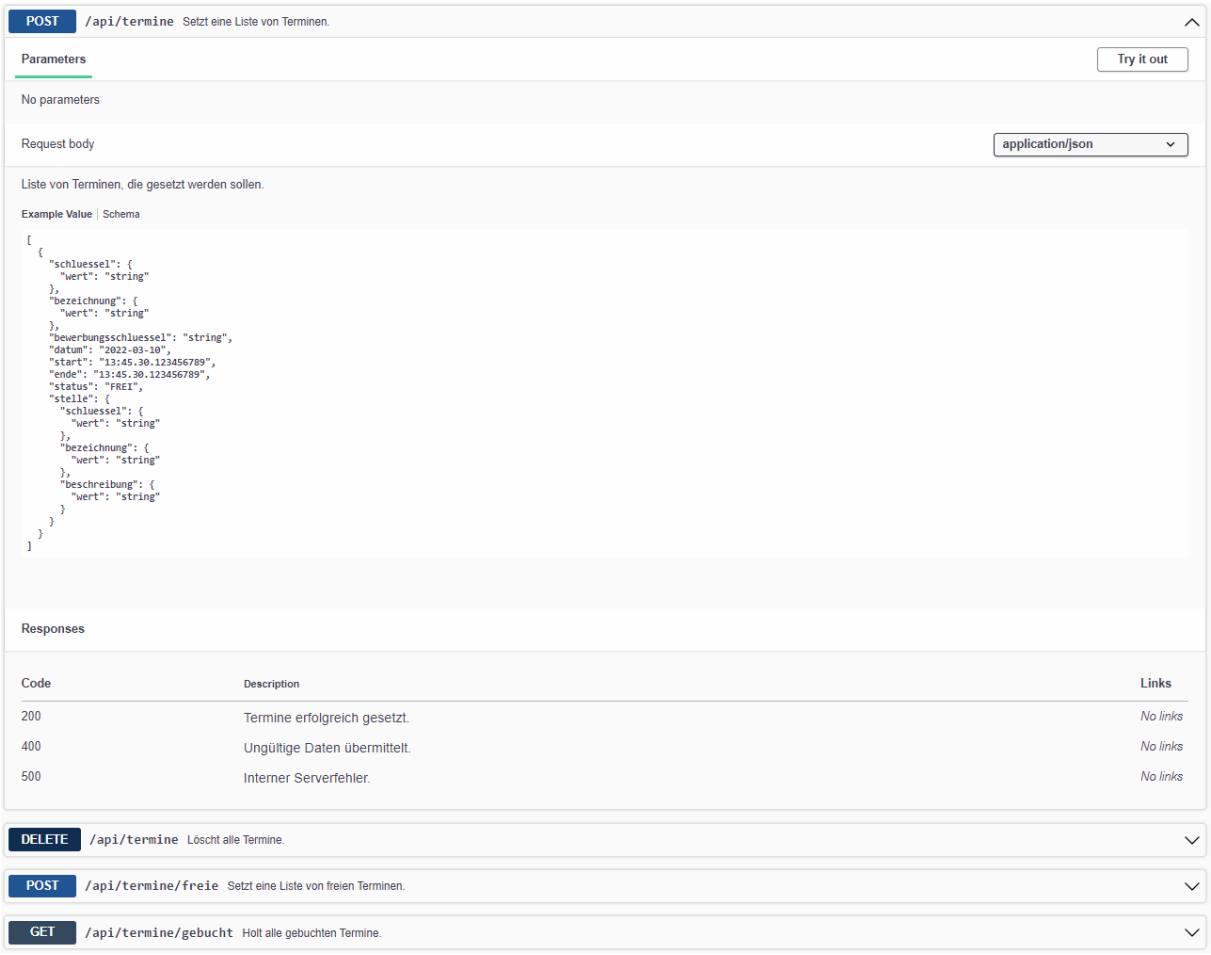


Abbildung 15: Mit OpenAPI und Swagger erstellte API-Dokumentation (Auszug)

A.29 `README.md` (Auszug)

ENTON

ENTON stellt eine Online-Komponente zum bereits bestehenden Bewerbungsmanagementsystem ROBBE dar. Es ermöglicht Bewerbern, sich online zu bewerben und nach Aufforderung Termine zu vereinbaren.

Lokale Entwicklung

1. Repository klonen: `git clone https://scm.ao-devnet/Java/ao-enton.git`
2. In das Projektverzeichnis navigieren: `cd ENTON`
3. Entwicklungsmodus starten: `./gradlew quarkusDev`
4. Eine Datenbank wird nicht benötigt, da die Anwendung im Entwicklungsmodus eine In-Memory-Datenbank verwendet.
5. Die Anwendung ist lokal unter folgender URL erreichbar: `http://localhost:8080/`

Die API-Dokumentation wird automatisch von Quarkus generiert und bei jeder Änderung aktualisiert. Sie ist unter folgender URL erreichbar:
`http://localhost:8080/q/swagger-ui/`

Architektur

ENTON setzt auf eine Schichtenarchitektur, die eine klare Trennung der Verantwortlichkeiten gewährleistet. Dies erleichtert die Wartbarkeit, Testbarkeit und Skalierbarkeit der Anwendung.

Frameworks und Technologien: Quarkus, MariaDB, Vavr.

Die Domänen-Schicht ist der zentrale Teil der Anwendung. Hier werden die Entitäten und die Geschäftslogik definiert.

- **Model**: Definition der Entitäten und der fachlichen Logik.
- **Services**: Implementierung der Geschäftslogik und Use-Cases.
- **Repositories**: Interfaces für CRUD-Operationen. Die Implementierung erfolgt in der Persistenz-Schicht.
- **DTOs**: Für den Datenaustausch zwischen UI/API und den Service-Klassen.

Die View-Schicht ist zuständig für die Benutzeroberfläche. Sie verwendet Qute und Renarde für dynamische Webseiten.

Die Ressourcen-Schicht stellt die REST-API von ENTON bereit und enthält spezialisierte Klassen für die Interaktion mit ROBBE.

Die Persistence-Schicht ist die Schnittstelle zur Datenbank. Sie nutzt JPA für die Objekt-Relationale Abbildung.

Glossar

- **Bewerbung**: Repräsentiert eine Bewerbung inklusive der zugehörigen Bewerberdaten.
- **Quelle**: Die Quelle von der aus der Bewerber auf die Stelle aufmerksam geworden ist.
- **Stelle**: Die spezifische Position, für die sich ein Bewerber bewirbt.
- **Termin**: Ein Zeitraum für einen Termin, der entweder frei oder gebucht sein kann.
- **ROBBE**: Das bestehende Bewerbungsmanagementsystem, das von ENTON erweitert wird.

Abbildung 16: `README.md` des Git-Repositorys (Auszug)