BEWERBERMANAGEMENT SYSTEM

GFOS Innovations-Award 2021

Exposee

Dies ist die Dokumentation zu dem während des Wettbewerbs programmierten Bewerbermanagement-System. Sie erläutert die Projektdurchführung und die allgemeine Umsetzung. Des Weiteren sind eine Installationsanleitung sowie ein Benutzerhandbuch angefügt.

Florian Noje, Lukas Krinke & Simon Engel

mit Lehrer Michael Albrecht

Inhaltsverzeichnis

Inhaltsverzeichnis 1
1. Projekt 3
1.1 Vorbereitung 3
1.1.1 Aufgabenstellung 3
1.1.2 Erste Überlegungen 3
1.1.3 Aufgabenverteilung 3
1.2 Umsetzung 4
1.2.1 Verwendete Technologien 4
1.2.2 Durchführung 4
1.2.3 Generelle Software-Architektur 5
1.2.4 Backend 7
Architektur 7
Filter 8
Webservices 9
EJBs – Enterprise Java Beans 9
Services 10
Entity Classes 11
Datenbank 13
Speicherung von Dateien und Bildern 16
Externe API zum GeoCoding 16
Struktur der Teams und Personaler 17
1.2.5 Frontend 19
Architektur 19
Angular 19
State-Management 20
Viewports 20
Bekannte Bugs und fehlende Funktionen 21
1.2.6 Sicherheit 22
Hashing der Passwörter 22
Verifizierung der E-Mail 22
Passwort zurücksetzen 22
Authentifizierung & Sitzungsmanagement 23
2-Faktor-Authentifizierung 23
1.3 Nachbetrachtung 24

Welche Ziele wurden erreicht? 24
Wo besteht noch Verbesserungsbedarf? Welche Features hätten wir gerne noch eingebaut? 24
Fazit 25
2. Benutzerhandbuch 26
2.1 Installation 26
2.1.1 Installation mit Netbeans 26
Backend 26
Frontend 28
2.2 Quick-Start 29
Bewerber 29
Personaler 29

1. Projekt

1.1 Vorbereitung

1.1.1 Aufgabenstellung

Das Ziel des Projektes sollte es sein, ein browserbasiertes Bewerbermanagement-System zu konzipieren und zu programmieren. Das Backend sollte dabei in Java geschrieben werden; für das Frontend durfte ein beliebiges JavaScript Framework verwendet werden. Dieses Bewerbermanagement-System sollte übliche Funktionalitäten, wie zum Beispiel das Erstellen von Jobangeboten und das Verwalten von Bewerbungen seitens der Personaler eines Betriebes bieten sowie das Abschicken von Bewerbungen seitens eines Bewerbers.

1.1.2 Erste Überlegungen

Zunächst haben wir Pläne für die einzelnen Komponenten angefertigt. Diese umfassen folgenden Punkte:

- Frontend
  - Skizzen der wichtigsten Masken
  - Grobe Designentscheidungen, wie Farben, etc.
  - Navigation: Wie kann der User intuitiv zwischen den Seiten navigieren?
- Backend
  - Liste aller Funktionen, die essentiell sind
  - Struktureller Aufbau
  - Welche Technologien benötigen wir des Weiteren?
- Datenbank
  - Entwurf aller Tabellen und zugehöriger Attribute

Für alle wichtigen Angelegenheiten wurde ein Notizbuch erstellt, in das jeder von uns jederzeit wichtige Ideen oder existierende Fehler hineinschreiben konnte. So konnte eine reibungslose Kommunikation sichergestellt werden und keine Ideen, beziehungsweise Fehler in der Software, sind in Vergessenheit geraten.

1.1.3 Aufgabenverteilung

Nachdem alle wichtigen Vorbereitungen gemeinsam getroffen wurden, haben wir uns an die eigentliche Implementierung des Projektes gesetzt. Dafür haben wir uns nach unseren Stärken aufgeteilt. Simon hat sich mit der Umsetzung des Backends auseinandergesetzt, da er aufgrund des Informatik- Leistungskurses und eigener Projekte in diesem Bereich bereits einiges an Erfahrung sammeln konnte. Lukas und Florian haben sich um die Implementierung des Frontend gekümmert, da sie schon mehrere Anwendungen mit Angular programmiert haben und sich auf diesem Gebiet gut auskennen. Auf Grund der Vorabi- und Abiturklausuren war die Zeit sehr knapp, wodurch sich nur auf die essentiellen Funktionen konzentriert werden konnte.

1.2 Umsetzung

1.2.1 Verwendete Technologien

Für das Frontend haben wir Angular (Version 13.1.4) als Framework mit Taiga UI (Version 2.28.0) für das Design genutzt. So konnte einfach zwischen Komponenten und Logik kommuniziert werden. Alle benötigten Module mit zugehöriger Versionsnummer sind in der *package-lock.json* aufgelistet.

Das Backend wurde entsprechend der Aufgabenstellung in JavaEE 8 geschrieben. Als Datenbank haben wir die in NetBeans integrierte JavaDB (Version 10.14.2.0) genutzt. Bereitgestellt wird es mit dem GlassFish-Application Server (Version 5.1.0). Zur Realisierung einiger Features benötigten wir folgende Bibliotheken:

1.2.2 Durchführung

Das Frontend wurde zunächst mit statischen Beispiel-Datensätzen implementiert, um ohne ein funktionierendes Backend das Design zu entwickeln. Zudem konnten so Backend-Fehler ausgeschlossen werden und sich zunächst ausschließlich auf das Design konzentriert werden.

Währenddessen wurde zuerst die Datenbank erstellt. Anschließend wurden die einzelnen Routen geschrieben, die vorher beschlossen wurden. Während der Umsetzung wurden die Datenbanktabellen immer wieder um einzelne Attribute erweitert. Bei der Implementierung der einzelnen Routen und Methoden wurde nach dem „Trial and Error-Prinzip“ vorgegangen. Getestet wurden die Routen zunächst mit dem Tool „Postman“.

Auf Grund der geringen Zeit die uns zur Verfügung stand, war das Backend bereits einige Zeit vor dem Frontend fertig. Nachdem das Frontend ebenfalls alle grundlegenden Ansichten enthielt, wurden es nach und nach mit dem Backend verknüpft.

1.2.3 Generelle Software-Architektur

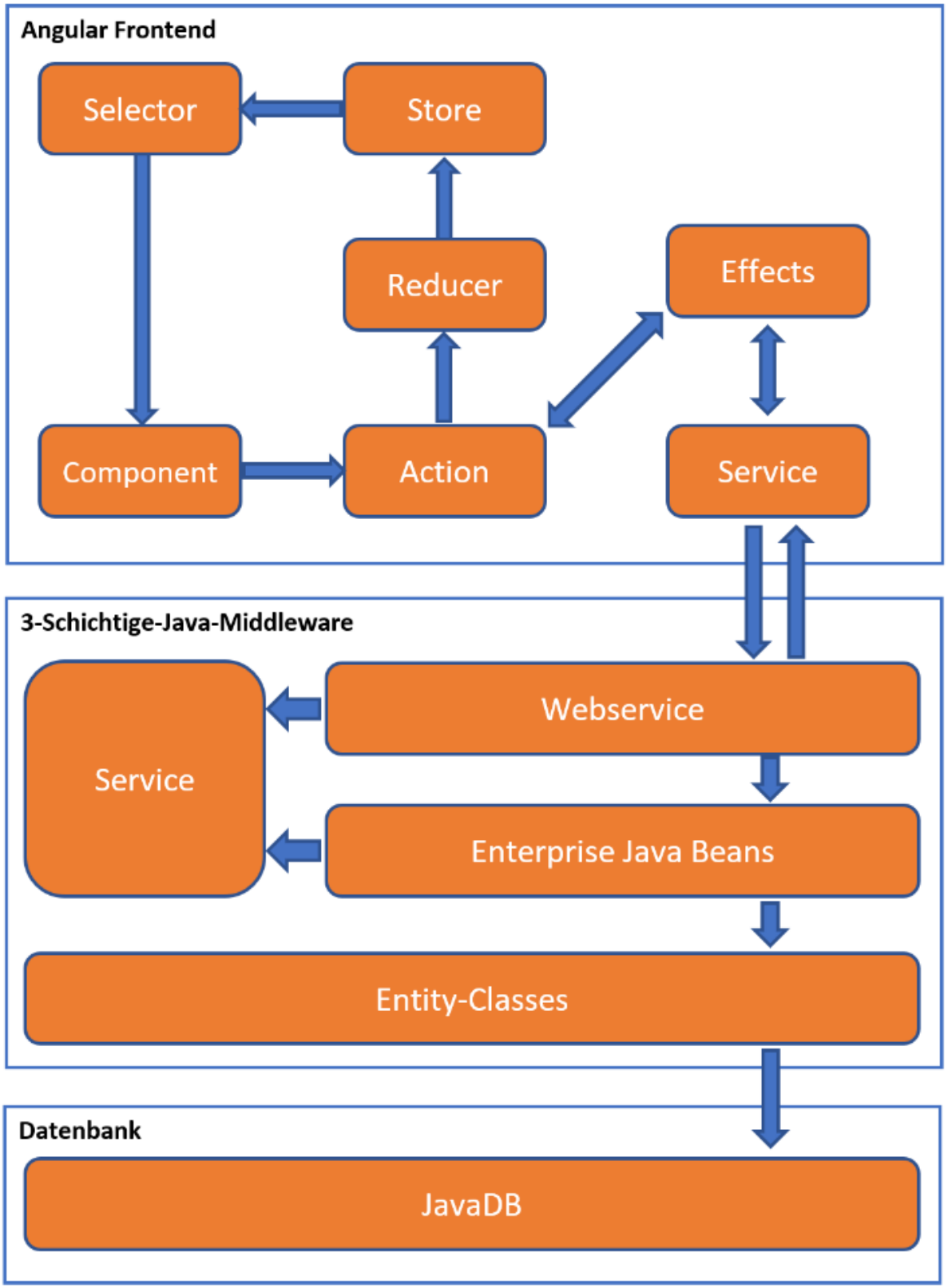


Abbildung 1: Software Architektur

Als generelle Struktur haben wir eine RESTful-Webservice-Anwendung gewählt. Diese besteht aus dem Frontend, welches der Nutzer im Browser aufruft. Dieses kommuniziert per HTTP über eine Middleware, in diesem Fall über eine dreischichtige JavaEE-Architektur, mit der Datenbank. Das nachstehende Schaubild zeigt diese Struktur detailliert auf. Die einzelnen Schichten werden im Folgenden genauer erläutert.

Folgendes Client-Server-Diagramm veranschaulicht noch einmal die Kommunikation zwischen Front- und Backend und den Datenfluss innerhalb des Backends.

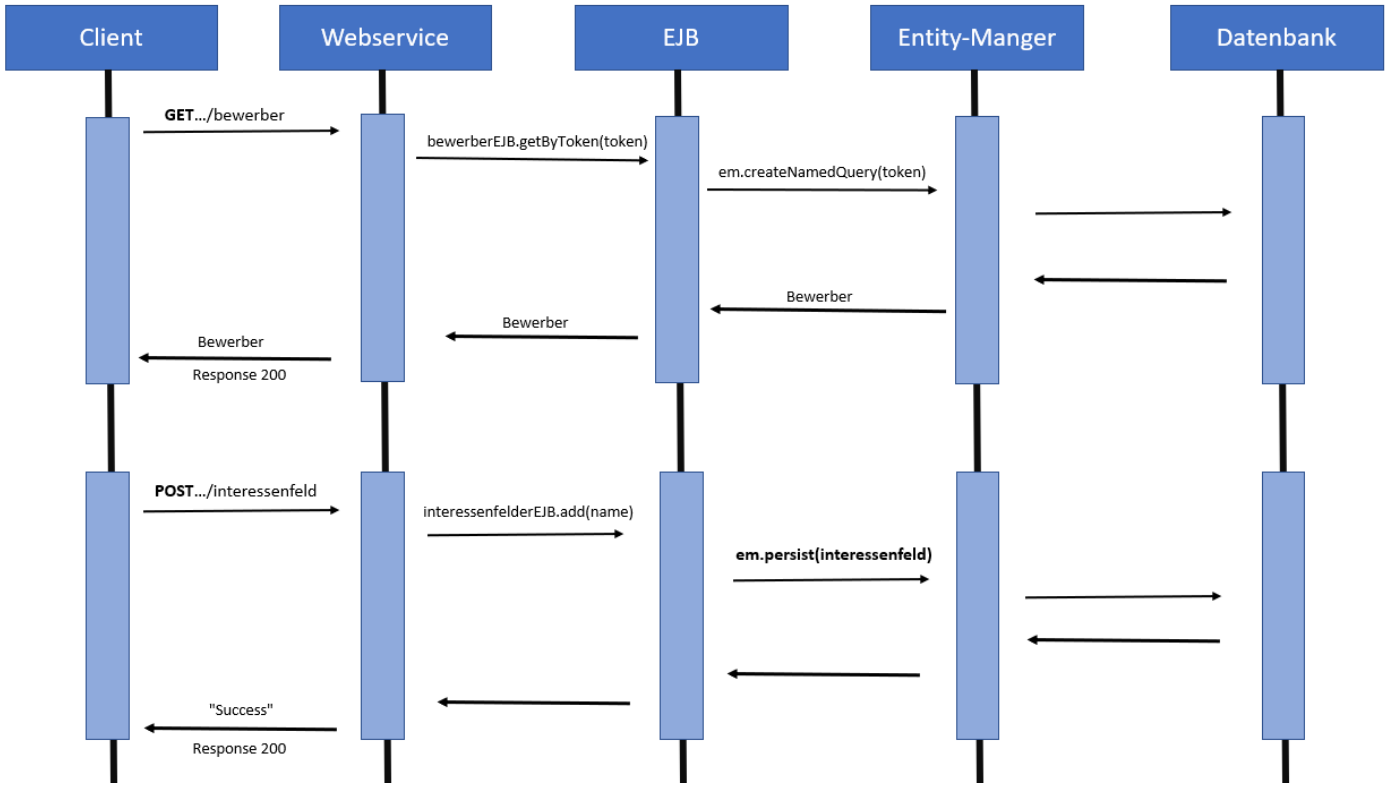


Abbildung 2: Client-Server Kommunikation

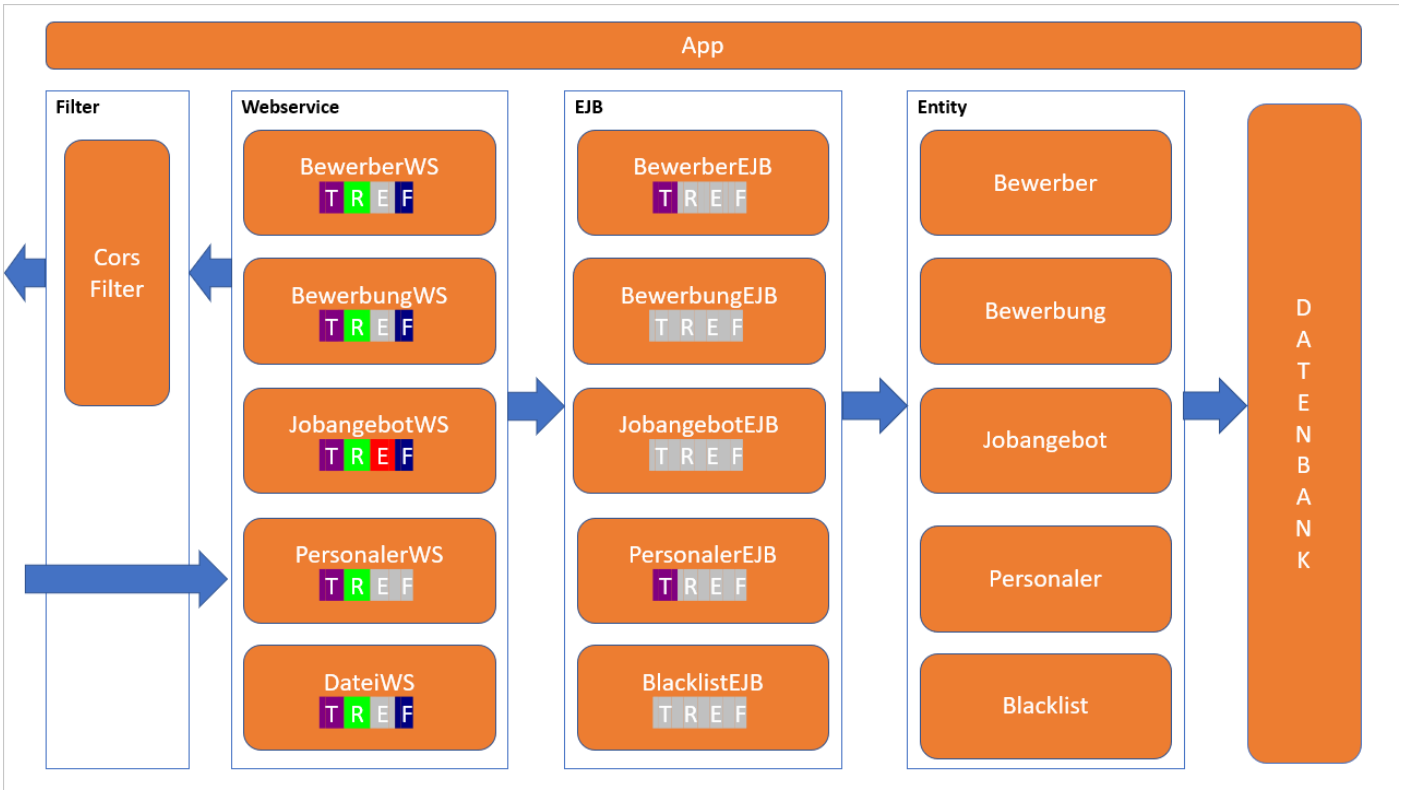
Wie zu erkennen ist, sendet der Client eine Anfrage an das Backend. Dieses verarbeitet dann die übergebenen Daten oder sucht angeforderte Ressourcen und schickt schließlich eine Antwort an den Client zurück, welche vom Frontend entsprechend visualisiert wird.

1.2.4 Backend

Architektur

Wie bereits erwähnt, gliedert sich unser Backend in eine dreischichtige Java-Architektur, welche sich in Webservices, EJBs und Entitätsklassen unterteilt. Diese Struktur verbessert sowohl die Übersicht über das gesamte Projekt, als auch die Flexibilität und das Debugging. Änderungen können dank klarer Unterteilung einfach und gezielt vorgenommen werden, ohne eventuell Bugs in anderen Teilen der Anwendung zu verursachen. Um diese Vorteile noch besser auszunutzen, haben wir uns zudem an dem modernen Prinzip der Microservices orientiert und einzelne Funktionen, nach ihrer Zugehörigkeit geordnet, auf viele Webservices aufgeteilt.

Zur Verknüpfung mit der Datenbank nutzen wir die Java Persistence API, kurz JPA, mit dem EntityManager. Sie ermöglicht es, die Datenbankeinträge als Java-Objekte bereitzustellen und zudem im Persistence-Context zu speichern, sodass nicht immer neue Datenbankabfragen getätigt werden müssen.



Tokenizer (T), ResponseService(R); EntfernungsService(E); FileService(F)

Abbildung 3: Architektur des Backends anhand ausgewählter Klassen

Das obenstehende Schaubild verdeutlicht die Beziehungen zwischen den Klassen und Schichten innerhalb des Backends. Eingehende Requests werden zunächst auf ein gültiges Token überprüft; einige Routen wie beispielsweise das Registrieren oder der Login sind davon ausgenommen. Ist ein gültiges Token enthalten, ruft der Webservice die entsprechende Methode in dem EJB auf, welche über den EntityManager mit den Entitätsklassen interagieren. Zusätzlich können auch Methoden aus Service-Klassen aufgerufen werden, welche zur Vermeidung von Redundanzen in eigene Klassen ausgelagert wurden. Mit den angeforderten Daten wird schlussendlich ein Response-Objekt erzeugt, in welchem im CORS-Filter noch HTTP-Header für die Erlaubnis des *Cross Origin Ressource Sharing* gesetzt sind. Das vollständige Response-Objekt wird dann an den Client zurückgesandt.

GFOS_2022 1 API

Packages

Package	Description
Application	Grundbaustein der Application
EJB	EJBs mit der Geschäftlogik
Entitiy	Automatisch generierte Entitätsklassen der Datenbank
Filter	Filter zum hinzufügen von CORS-Headern
Service	Services der Application
WS	WebServices für das Entgegennehmen der Anfragen

Abbildung 4: Alle Packages des Backends

Filter

In unserer Architektur sind neben den drei erwähnten Schichten auch noch ein CORS-Filter enthalten. Sie implementieren das ContainerResponseFilter- oder ContainerRequestFilter-Interface. Diese erlauben es, die Response beziehungsweise die Request „abzufangen“ und eventuell zu modifizieren. Beim CORS-Filter nutzen wir dies, um jeder ausgehenden Response die Header anzufügen, die CORS erlauben. Dies war nötig, um auch den von Angular gesendeten Preflight-Requests die nötigen Header hinzuzufügen.

Package Filter

Filter zum hinzufügen von CORS-Headern

See: Description

Class Summary

Class	Description
Filter	CORS-Filter Dieser Filter fügt jeder Request Header hinzu, die CORS erlauben.

Abbildung 5: Package Filter

Webservices

Die Webservices, welche wir wie bereits erläutert in viele Microservices unterteilt haben, nehmen die HTTP-Requests an die unterschiedlichen URIs an. Wir verwenden hierfür die passenden Protokoll-Methoden von HTTP (POST, GET, PUT, DELETE) und orientieren uns an dem CRUD-Prinzip (Create, Read, Update, Delete). Der Webservice ruft anschließend die zugehörige Methode in den EJBs auf. Die Rückgabe der EJBs gibt er in Form eines Response-Objektes an den Client zurück. Im Falle eines Fehlers wird eine Response mit dem passenden HTTP-Status-Code und der Fehlermeldung zurückgegeben, damit das Frontend diese visualisieren kann.

Package WS

WebServices für das Entgegennehmen der Anfragen

See: Description

Class Summary

Class	Description
AdresseWS	Webservice für die Adresse Diese Klasse stellt Routen bezüglich der Adresse der Bewerber bereit.
AnmeldeWS	Webservice für die Anmeldung Diese Klasse stellt Routen bezüglich der Anmeldung bereit.
BewerbereinstellungenWS	Webservice für die Bewerbereinstellungen Diese Klasse stellt Routen bezüglich der Einstellungen der Bewerber bereit.
BewerberWS	Webservice für Bewerber Diese Klasse stellt Routen bezüglich der Bewerber bereit.
BewerbungsnachrichtWS	Webservice für Bewerbungsnachrichten Diese Klasse stellt Routen bezüglich der Bewerbungsnachrichten bereit.
BewerbungWS	Webservice für Bewerbungen Diese Klasse stellt Routen bezüglich der Bewerbungen bereit.
DateiWS	Webservice für Dateien Diese Klasse stellt Routen bezüglich der Dateien bereit.
FachgebietWS	Webservice für Fachgebiete Diese Klasse stellt Routen bezüglich der Jobangebote bereit.
FotoWS	Webservice für Fotos Diese Klasse stellt Routen bezüglich der Fotos bereit.
InteressenfelderWS	Webservice für Interessenfelder Diese Klasse stellt Routen bezüglich der Interessenfelder bereit.
JobangebotWS	Webservice für Jobangebote Diese Klasse stellt Routen bezüglich der Jobangebote bereit.
LebenslaufstationWS	Webservice für Lebenslaufstationen Diese Klasse stellt Routen bezüglich der Lebenslaufstationen bereit.
PersonalerWS	Webservice für Personaler Diese Klasse stellt Routen bezüglich der Personaler bereit.
ToDoWS	

Abbildung 6: Package WebService

EJBs – Enterprise Java Beans

In den EJBs befindet sich die Geschäftslogik, die vom Webservice aufgerufen wird. Mithilfe einer Instanz des Entity-Managers aus der JPA werden hier Daten abgefragt, aktualisiert, hinzugefügt oder gelöscht. Er stellt zudem durch das Object-Relational-Mapping (ORM) die Datenbankeinträge als Java-Objekte zur Verfügung, sodass mit diesen auch Überprüfungen vorgenommen werden können. Somit stellen die EJBs die direkte Schnittstelle mit der Datenbank dar. Um auf dem Server Ressourcen zu schonen, sind sie zustandslos deklariert und werden im Webservice mit der Annotation @EJB instanziiert.

Package EJB

EJBs mit der Geschäftlogik

See: Description

Class Summary

Class	Description
AdresseEJB	EJB für Adressen Diese Klasse stellt Methoden bezüglich Adressen bereit.
BewerbereinstellungenEJB	EJB für Bewerbereinstellungen Diese Klasse stellt Methoden bezüglich Bewerbereinstellungen bereit.
BewerberEJB	EJB für Bewerbern Diese Klasse stellt Methoden bezüglich Bewerbern bereit.
BewerbungEJB	EJB für Bewerbungen Diese Klasse stellt Methoden bezüglich Bewerbungen bereit.
BewerbungsnachrichtEJB	EJB für Bewerbungsnachrichten Diese Klasse stellt Methoden bezüglich Bewerbungsnachrichten bereit.
BewerbungstypEJB	EJB für Bewerbungstypen Diese Klasse stellt Methoden bezüglich Bewerbungstypen bereit.
BlacklistEJB	EJB für die Blacklist der Tokens Diese Klasse stellt Methoden bezüglich der Blacklist für noch gültige Tokens bereit.
FachgebietEJB	EJB für Fachgebiete Diese Klasse stellt Methoden bezüglich Fachgebieten bereit.
InteressenfelderEJB	EJB für Interessenfelder Diese Klasse stellt Methoden bezüglich Interessenfelder bereit.
JobangebotEJB	EJB für Jobangebote Diese Klasse stellt Methoden bezüglich Jobangeboten bereit.
LebenslaufstationEJB	EJB für Lebenslaufstationen Diese Klasse stellt Methoden bezüglich Lebenslaufstationen bereit.
PersonalerEJB	EJB für Personaler Diese Klasse stellt Methoden bezüglich Personalern bereit.
ToDoEJB	EJB für Fotos Diese Klasse stellt Methoden bezüglich Fotos bereit.

Abbildung 7: Package EJB

Services

Einige Methoden, die in mehreren Klassen benötigt werden, haben wir aufgrund der Vermeidung von Redundanzen und zur Übersichtlichkeit des Codes in Services ausgelagert. Sie sind nach den Funktionalitäten der enthaltenen Methoden geordnet.

Ein Beispiel hierfür ist der AntwortService. Er stellt eine Methode zur Konstruktion eines Response-Objektes mit einem JSON-String an Daten und eine Methode zum Bauen eines Response-Objektes mit einem Fehler bereit. Somit muss dieses Objekt nicht in jeder Methode jedes Webservices konstruiert werden, sondern ein Methodenaufruf genügt. Mit „response.build(JSON-String)“ wird am Ende jeder Webservice-Methode dann ein Response-Objekt erstellt.

Package Service

Services der Application

See: Description

Class Summary

Class	Description
EntfernungsService	Service zum Berechnen von Entfernungen
FileService	Die Klasse zum Verwalten von Dateien, die von Nutzern gesendet werden.
GeocodingService	Service für Geocoding Diese Klasse stellt Methoden zum Geocoding bereit
Hasher	Die Java-Klasse zum Hashen der Passwörter.
MailService	Die Java-Klasse zum Versenden von E-Mails.
ResponseService	Service für das Erstellen von Response-Objektes Diese Klasse erstellt Response-Objekten.
Tokenizer	Die Java-Klassse zur Identifizierung und Authentifizierung der Nutzer.

Abbildung 8: Package Service

Entity Classes

Die Entitätsklassen werden vom EntityManager verwendet, um die Datenbankeinträge zu sogenannten Plain Old Java Objects, kurz POJOs, umzuwandeln. Dadurch kann in den EJBs wie mit normalen Java-Objekten gearbeitet werden. Aufgrund des Persistence-Contexts werden Änderungen an diesen direkt in die Datenbank übernommen, wodurch Daten beispielsweise sehr unkompliziert und schnell aktualisiert werden können.

Um bei einigen Routen jedoch nur relevante Attribute an den Client zu geben, die dort auch wirklich angezeigt werden, müssen diese aus dem Persistence-Context entfernt werden. Ansonsten würden durch das Setzen auf null auch die Datenbankeinträge gelöscht. Theoretisch kann dafür die Funktion „detach()“ des EntityManagers genutzt werden, jedoch gab uns das Implementieren einer eigenen clone()-Methode, welche im Prinzip selbiges tätigt, mehr Freiraum, da die Attribute der Objekte beliebig modifiziert werden können. Zudem wird der JSON-String, welcher später an das Frontend gesendet wird, dadurch erheblich verkürzt. Attribute, die in der clone()-Methode auf null gesetzt werden, werden von dem GSON-Parser nicht übernommen, während EntityManager.detach() auch die „leeren“ Attribute mit in den JSON-String schreibt.

Package Entitiy

Automatisch generierte Entitätsklassen der Datenbank

See: Description

Class Summary

Class	Description
Adresse	Automatisch generierte Entitätsklassen der Tabelle Adresse
Bewerber	Automatisch generierte Entitätsklassen der Tabelle Bewerber
Bewerbereinstellungen	Automatisch generierte Entitätsklassen der Tabelle Bewerbereinstellungen
Bewerbung	Automatisch generierte Entitätsklassen der Tabelle Bewerbung
Bewerbungsnachricht	Automatisch generierte Entitätsklassen der Tabelle Bewerbungsnachricht
Bewerbungstyp	Automatisch generierte Entitätsklassen der Tabelle Bewerbungstyp
Blacklist	Automatisch generierte Entitätsklassen der Tabelle Blacklist
Fachgebiet	Automatisch generierte Entitätsklassen der Tabelle Fachgebiet
Interessenfelder	Automatisch generierte Entitätsklassen der Tabelle Interessenfelder
Jobangebot	Automatisch generierte Entitätsklassen der Tabelle Jobangebot
Lebenslaufstation	Automatisch generierte Entitätsklassen der Tabelle Lebenslaufstation
Personaler	Automatisch generierte Entitätsklassen der Tabelle Personaler
Todo	Automatisch generierte Entitätsklassen der Tabelle Todo

Abbildung 9: Package Entity

Datenbank

Alle Nutzer erhalten nach der Registrierung ein eigenes Nutzerprofil. Dabei wird eine Unterteilung zwischen Bewerbern und Personalern vorgenommen. Bewerber können sich selbständig registrieren, wohingegen Personaler-Accounts nur von einem anderen Personaler, beispielsweise dem Team-Chef, erstellt werden können. Fast alle weiteren Tabellen beziehen sich mithilfe von Fremdschlüsseln auf die Nutzerprofile, wodurch auch der EntityManager auf alle Objekte und somit Attribute eines Nutzers zugreifen kann. Jede Tabelle wird um eine automatisch generierte ID in Form eines Integers erweitert, um einen eindeutigen Primärschlüssel zu gewährleisten.

Um atomare Datensätze zu erhalten und Redundanzen zu vermeiden, haben wir uns bei der Planung und Erstellung der Datenbank an den ersten drei Normalisierungen orientiert. Größtenteils werden so auch funktionale und transitive Abhängigkeiten vermieden, sodass wir eine möglichst gute Performance der Datenbank sicherstellen können. Bei einigen Tabellen haben wir uns jedoch auch bewusst dagegen entschieden, um gerade während des Entwicklungsprozesses die Übersichtlichkeit der Tabellen zu gewährleisten.

Das folgende vereinfachte ER-Diagramm veranschaulicht die Beziehungen der Tabellen und somit auch der Entitätsklassen zueinander. Zur Übersichtlichkeit wurden die Attribute außenvorgelassen, sie werden anschließend für jede Tabelle einzeln dargestellt.

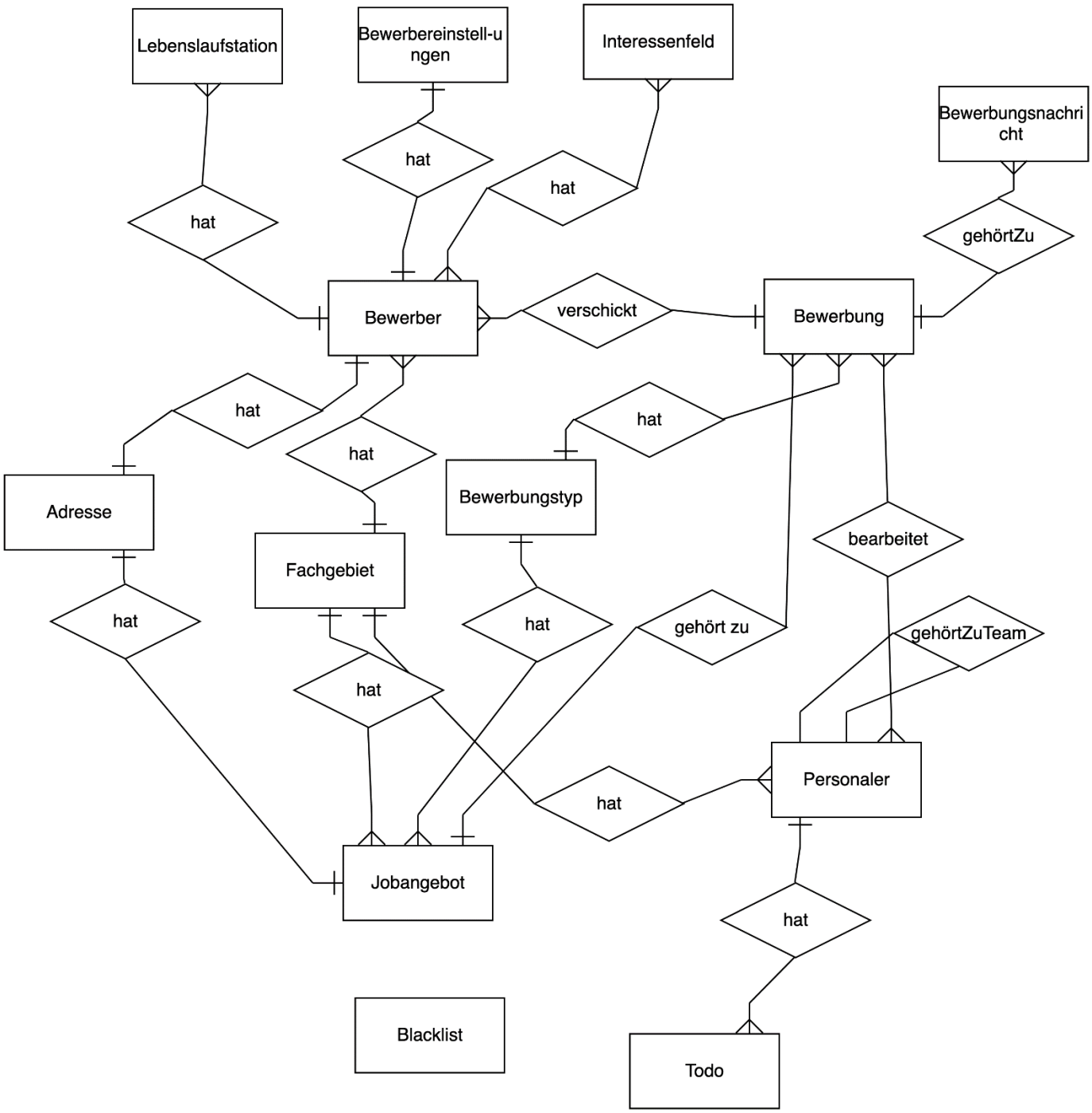


Abbildung 10: Vereinfachtes ER-Diagramm zur Veranschaulichung der Entitätsbeziehungen

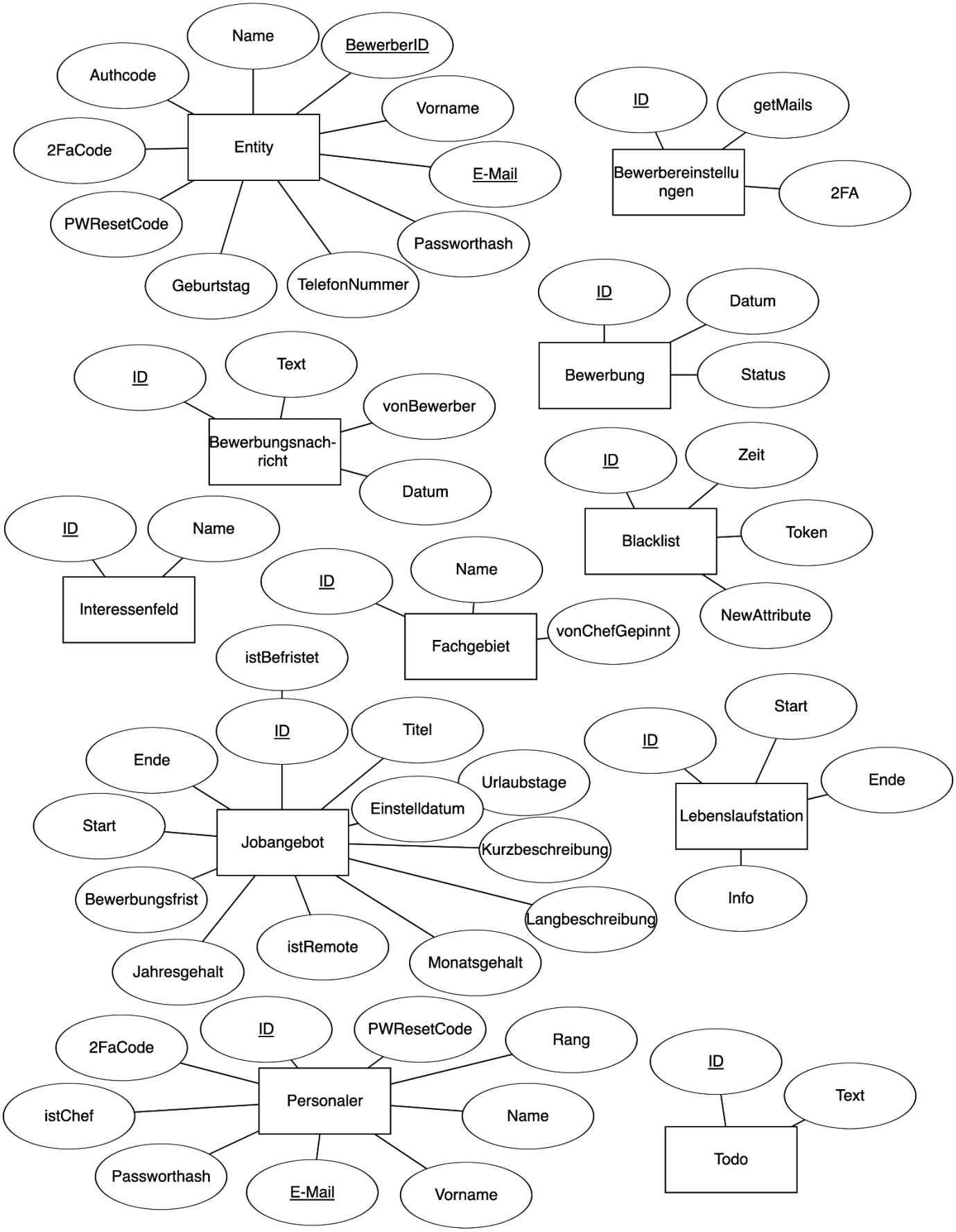


Abbildung 11: Datenbank Tabellen

Speicherung von Dateien und Bildern

Da wir uns bewusst dagegen entschieden haben, Dateien und Bilder in der Datenbank als Base64-String zu speichern, haben wir eine Alternative gesucht. Die Lösung war es die Dateien im Dateisystem des Glassfish-Servers zu speichern. Dafür wurde unter dem Pfad „GlassFish_Server/glassfish/domains/domain1/config“ der Ordner „projectFiles“ erstellt, indem alle Dateien abgespeichert werden. Es gibt für jede der vier Dateiarten (Bewerbungsschreiben, Lebenslauf, Referenz und Profilbilder) einen separaten Ordner, sodass als Dateiname die automatisch generierte ID des zugehörigen Datenbankeintrags genutzt wird.

Dateien vom Frontend werden generell als Base64-String an das Backend geschickt. Um diese Dateien nun zu speichern, muss in der zugehörigen Webservice-Route der Dateiname auf unterschiedliche Weise bestimmt. Bei der Speicherung eines Profilbildes ist zum Beispiel die ID des Bewerbers der Dateiname. Die Webservice-Methode ruft daraufhin eine der Speicherungsmethoden des Fileservices auf, die den Dateipfad generiert und den Base-64-String zur Speicherung vorbereitet. Für die unterschiedlichen Dateiarten gibt es jeweils unterschiedliche Speicherungsmethoden, damit der Pfad richtig generiert wird. Ein wichtiger Schritt ist es dabei, dass das Präfix des Base64-String (data:application/pdf;base64,) entfernt wird, da sonst die Umwandlung in eine Datei nicht funktioniert. Um die Speicherung der Dateien abzuschließen, wird die interne Methode „saveFile“ aufgerufen, die den übergegebenen Base64-String unter dem gegebenen Pfad als Datei abspeichert.

Das Abrufen von Dateien und zurücksenden ans Frontend funktioniert auf umgekehrte Weise. Dabei wird die Datei über den richtigen Pfad aus dem Dateisystem gelesen, in einen Base64-Stirng umgewandelt und ans Frontend gesendet.

Zum Löschen von Dateien wird Datei aus dem Dateisystem geladen und über die java-interne Methode „.delete()“ gelöscht.

Wenn man eine Datei aktualisieren möchte, wird die Methode zum Speichern mit der neu übergebenen Datei erneut aufgerufen. Die alte Datei wird dabei einfach überschrieben.

Externe API zum GeoCoding

Da wir uns dazu entschieden haben, auch eine Entfernungssuche in unsere Bewerbermanagement-System einzubauen, um eine Umkreissuche von Jobs in der Nähe des Bewerbers zu realisieren, benötigen wir eine externe API zum GeoCoding. GeoCoding beschreibt den Prozess der Umwandlung von einer Adresse in Koordinaten. In unserem Beispiel werden die Koordinaten eines Jobangebotes benötigt, um die Entfernung zu einem Bewerber zu bestimmen. Bei der Entfernungssuche wird eine JavaScript-Methode genutzt, um die aktuellen Koordinaten des Bewerbers zu ermitteln. Aus diesen beiden Koordinatensätzen lässt sich dann mühelos die Entfernung berechnen.

Um aus Java heraus Http-Aufrufe senden zu können, müssen die zugehörigen Zertifikate manuell installiert werden, wie im Nutzerhandbuch beschrieben ist. Zum Zeitpunkt der Abgabe haben wir nur einen Weg gefunden, das Zertifikat auf UNIX-basierten System zu installieren. Auf Windows geht dies leider nicht, weswegen die Entfernungssuche auf Windows nicht aktiviert ist.

Die API ist unter dem Link „https://api.mapbox.com/geocoding/v5/mapbox.places/“ zu erreichen. Das Access-Token ist „pk.eyJ1Ijoic2lyc2ltb24wNCIsImEiOiJja3h1anRzY3YweDE1Mm9vNW4xbmY2dGN1In0.1LfSFOli4OfQCqaz9qjwrg“. Mit diesem Token können pro Monat 100000 Anfragen gesendet werden, was für unsere Anwendung mehr als ausreichend ist. Ein Beispiel API-Call ist:

„https://api.mapbox.com/geocoding/v5/mapbox.places/9%20Am%20Lichtbogen%2045141%20Essen.json?access_token=pk.eyJ1Ijoic2lyc2ltb24wNCIsImEiOiJja3h1anRzY3YweDE1Mm9vNW4xbmY2dGN1In0.1LfSFOli4OfQCqaz9qjwrg“

Struktur der Teams und Personaler

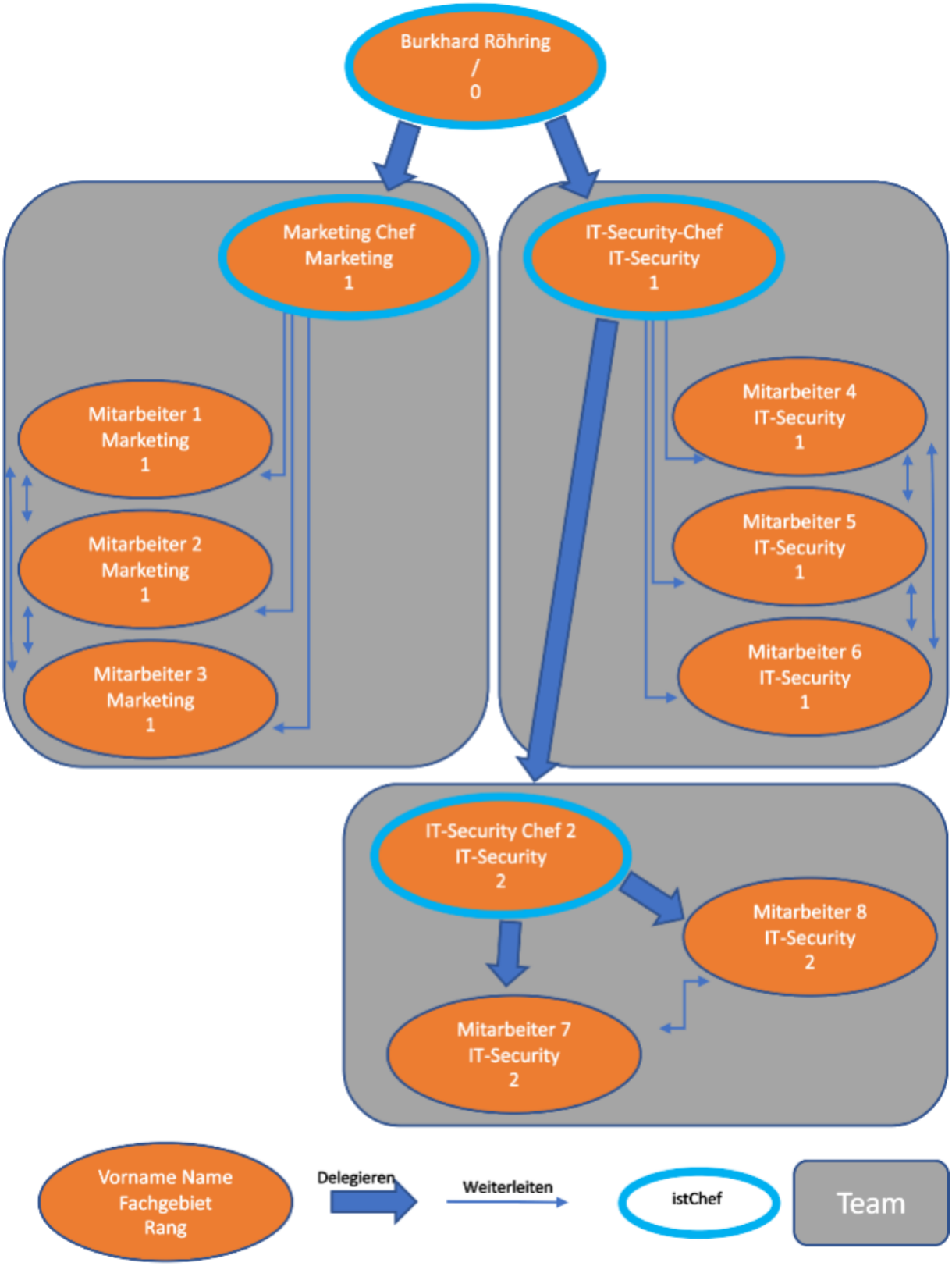


Abbildung 12 Beispiel-Hierarchie

Dieses Diagramm veranschaulicht die Team- und Personalstruktur.
Jeder Personaler hat einen Rang und ein Fachgebiet. Je kleiner der Rang ist, desto höher ist der Personaler in der Firma.
Deshalb ist der Ober-Chef des ganzen Unternehmens Burkhard Röhrig mit dem Rang 0. Der Ober-Chef des Unternehmens hat außerdem kein Fachgebiet.
Generell sind immer Personaler, die sich auf einer Ebene befinden und das gleiche Fachgebiet haben in einem Team. Ein Team arbeitet immer zusammen, hat aber bei der Umsetzung des Systems keine sehr große Bedeutung, Teams sind vordergründig für die Benutzung und Organisation der Personaler untereinander wichtig.
Es gibt verschiedene Möglichkeiten, Bewerbungen an andere Personaler zu verteilen. Die erste Möglichkeit ist es, Bewerbungen zu delegieren. Diese Möglichkeit besteht nur für Chefs des jeweiligen Teams (blau umrandet). Mit der Delegierung einer Bewerbung wird diese an ein anderes
Team niedrigeren Ranges abgebeben, deswegen kann die Bewerbung nur an den Chef des Teams darunter delegiert werden. Wenn dieser Prozess abgeschlossen ist, wurde die Bewerbung jedem Personaler, der dem Team des Chefs angehört, entzogen und sie arbeiten nicht mehr daran.
Außerdem besteht die Möglichkeit des Weiterleitens einer Bewerbung. Das ist für jeden Personaler möglich, aber nur innerhalb des gleichen Teams. Nach dem Weiterleiten arbeitet der Personaler, der die Anfrage gesendet hat, immer noch an der Bewerbung, dazu zusätzlich aber auch der Personaler, an den die Bewerbung weitergeleitet wurde.

1.2.5 Frontend

Architektur

Wie bei aktuellen Web-Applikationen üblich, wurde das Frontend in einem JavaScript-Framework programmiert. Wir haben uns für Angular mit TypeScript entschieden. Es handelt sich um eine SPA, eine Single-Page-Application. Die einzelnen Komponenten werden in der App-Komponente per Angular-Routing initialisiert. Dies bedeutet nach einmaliger minimal längerer Ladezeit beim ersten Aufruf der Seite deutlich kürzere Ladvorgänge beim Wechseln zwischen den Ansichten, was bei einem Messenger häufig der Fall ist.

Identisch zum Backend haben wir uns auch hier für eine mehrschichtige Architektur entschieden, wobei die einzelnen Schichten bestimmte Aufgabenbereiche abdecken. Auch hier kann die Software dadurch einfach erweitert oder Fehler behoben werden.

Das folgende Schaubild veranschaulicht die Struktur inklusive der Kommunikation allgemein. Im Folgenden werden die einzelnen Bestandteile genauer erläutert.

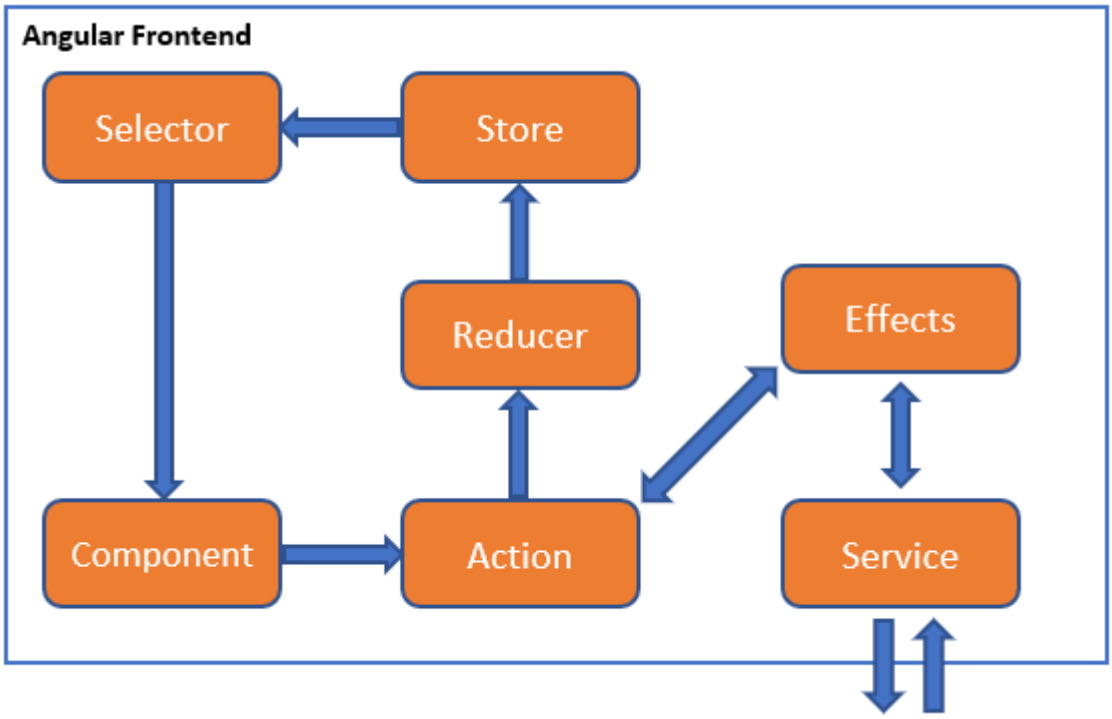


Abbildung 13: Architektur des Frontends

Angular

Neben der Vorerfahrung von Florian und Lukas haben wir uns auch für Angular entschieden, da es mit TaigaUI eine Lösung für das Design unserer App bietet, die sehr gut in das restliche Framework zu integrieren ist. Überzeugt hat uns außerdem der Komponenten-basierte Aufbau. Die Komponenten können dank zahlreicher Möglichkeiten einfach untereinander kommunizieren und fördern somit eine noch bessere Projektstruktur.

Die einzelnen Components sind zudem leicht verständlich, da im Gegensatz zu anderen Frameworks das Template (HTML), Stylesheet (SCSS) und die eigentliche TypeScript-Klasse getrennt voneinander vorliegen. TypeScript hilft uns dabei, die App einfacher zu debuggen, da die meisten Fehler durch das strict-type-checking direkt vom Compiler erkannt werden und behoben werden können.

State-Management

Um eine möglichst unkomplizierte Kommunikation zwischen den Components untereinander sowie Components und Services sicherzustellen, haben wir uns für eine State-Management-Lösung mit NgRx entschieden. Dies wirkt zunächst aufwendig und umständlich, bietet aber zahlreiche Debugging Hilfen und garantiert eine reibungsfreie Nutzung, wegen dem „Single-Point-of-Truth“.

Vereinfacht lässt es sich wie folgt erklären: Für die gesamte App gibt es einen Store, in dem alle relevanten Daten geladen sind. Sämtliche Components können (gleichzeitig) auf die Daten zugreifen, um sie zum Beispiel anzuzeigen. Es können aber auch Daten aus einem Dialog im Store abgelegt werden und zur Weiterverarbeitung in einem anderen Component verwendet werden.

Gesteuert wird der Ablauf über Actions. Diese können entweder vom User durch das Klicken eines Buttons, einer Eingabe in ein Textfeld oder andere Action selbst, ausgelöst werden. Die Actions lösen dann Reducer aus, die den State manipulieren. Über Selectors können die Components auf die Daten zugreifen und sie anzeigen. Da die Daten als Observables geladen werden, sind sämtliche Änderungen des Stores direkt im Component sichtbar.

Die Kommunikation mit dem Backend ist wie im obigen Schaubild zu sehen über Effects geregelt. Diese werden im Hintergrund wiederum durch Actions ausgelöst und führen dann die entsprechende Http-Request aus. Damit theoretisch User-Feedback möglich ist, handhaben wir vom Server angefragte Daten mit RemoteData-Objekten (importiertes Package). Diese haben neben den eigentlichen Daten, wie zum Beispiel Bewerbungen, auch ein Tag mit dem Zustand der Request (not-asked, in-progress, success, failure). Über entsprechende Pipes kann im Template je nach Zustand zum Beispiel ein Loading-Spinner, eine Fehlermeldung oder bei Success die richtigen Daten angezeigt werden.

Da wir leider zu wenig Zeit hatten, konnten wir dies nicht überall und umfangreich implementieren.

Viewports

In der Regel werden Bewerbungen an einem Desktop bearbeiten. Aufgrund dessen wählten wir für das Design der App den Desktop-First-Approach und priorisierten damit den Desktop als Endgerät. Dank dieses Ansatzes ist es leichter, auch die anderen Viewports zu gestalten. Allerdings ist die Bildschirmgröße eines Smartphones zu klein, um zum Beispiel die vielen Informationen eines Bewerberprofils anzuzeigen. Deshalb haben wir uns dazu entschieden, auf kleineren Geräten einige Ansichten zu entfernen und uns auf die essentiellen Informationen zu beschränken. Insgesamt haben wir für folgende Gerätetypen und Ausrichtungen einen Breaktpoint über CSS-MediaQueries konfiguriert:

- Smartphone Hoch- und Querformat
- Tablet Hoch- und Querformat
- Computermonitore

Dabei gibt es im Endeffekt zwei Grundformen des Designs, welche bei den unterschiedlichen Gerätetypen nochmals angepasst wurden. Ein Beispiel ist zum Beispiel die Sidebar, welche man als Personaler sieht. Diese ist in der Desktopansicht auf der linken Seite und auf kleineren Geräten nimmt sie den oberen Bereich der Seite in Anspruch. Dadurch ist die Oberfläche insgesamt immer aufgeräumt.

Bekannte Bugs und fehlende Funktionen

Auf Grund der fehlenden Zeit konnten einige Bugs nicht mehr behoben werden, folgende sind bekannt:

- Loggen Sie sich als Personaler ein, und kriegen keinen Fehler, dass der 2FA-Code falsch war, so hat das Routing zur Landing-Page versagt. Geben Sie bitte in der URL „/jobs“ ein. Das Token wurde trotzdem automatisch gespeichert.
- Wenn man als Bewerber eine seiner Bewerbungen löscht, wird man einmal auf die Details-Seite dieses Jobs weitergeleitet. Die Bewerbung wurde trotzdem gelöscht.
- Sämtlich Zurück-Navigation muss über den Browser passieren.
- Als Personaler kann man bei sämtlichen Bewerbungskarten nicht den Lebenslauf und das Anschreiben ansehen. Dies ist lediglich auf der Bewerber-Detail-Seite möglich.
- Die Funktion des Passwort-Resets ist leider noch nicht mit dem Backend verknüpft, auch wenn die UI dafür existiert.
- Bewerbungen können lediglich angenommen oder abgelehnt werden, allerdings nicht weitergeleitet oder delegiert werden, auch wenn die UI dafür existiert.
- Falls das Token seine Gültigkeit verliert, kann man keine Daten mehr abfragen, wird aber nicht automatisch auf die Login-Seite umgeleitet.

1.2.6 Sicherheit

Eine wichtige Angelegenheit neben möglichst differenzierten und gut durchdachten Features war es für uns, die Anwendung möglichst gut und effizient gegen Angriffe von außen abzusichern, sodass sie den heutigen Sicherheitsstandards weitestgehend, sofern es uns möglich war, entspricht. Aus diesem Grund haben wir folgende Sicherheitsfeatures eingebaut.

Hashing der Passwörter

Die Passwörter der Nutzer dürfen auf keinen Fall als Klartext in der Datenbank stehen, um im Falle eines Daten-Lecks weiterhin die Sicherheit der Nutzerkonten zu gewährleisten. Des Weiteren ist das Speichern in kryptografischer Form in der DSGVO vorgeschrieben, ein Verstoß könnte juristische Folgen haben. Aus diesen Gründen werden die Passwörter in unserer Applikation mit einer Einwegfunktion gehasht und der resultierende Hashwert in der Datenbank gespeichert. Aus diesem Hashwert ist das eigentliche Passwort nicht mehr zu berechnen. Bei einem Login wird das eingegebene Passwort dann erneut mit derselben Funktion gehasht und mit dem Hashwert aus der Datenbank verglichen.

Wir nutzen für das Hashing die kryptografische Funktion PBKDF2 mit dem Keyed-Hash Message Authentication Code SHA512. Ein Hacker könnte nun trotzdem noch mit einem sogenannten Rainbowtable-Angriff versuchen, die Passwörter herauszufinden. Dafür hat er möglichst viele denkbare Passwörter mit der verwendeten Funktion gehasht und in einer Datenbank gespeichert. Um ein Passwort daraufhin zu knacken, vergleicht er das gehashte Passwort mit allen Werten aus der Datenbank und sucht nach einer Übereinstimmung. Um dies zu erschweren, hängen wir an jedes Passwort vor dem Hashen noch einen geheimen SALT-Wert an, der das Passwort um 32 Zeichen verlängert.

Verifizierung der E-Mail

Da die E-Mail-Adresse in unserem System eine zentrale Rolle spielt (Login und Passwort zurücksetzen), muss der Nutzer sie nach der Registrierung einmalig verifizieren. So kann sichergestellt werden, dass er Zugriff auf das angegebene E-Mail-Konto hat. Zudem werden auch Bot-Angriffe erschwert.

Dafür schicken wir bei erfolgreicher Registrierung eine E-Mail an die angegebene Adresse, die einen zufällig generierten Code enthält. Über den Aufruf der entsprechenden Route im Browser wird die E-Mail-Adresse im System verifiziert. Über den zufällig erstellten Code kann das zugehörige Konto eindeutig verifiziert werden. Anschließend kann der Nutzer sich normal einloggen.

Passwort zurücksetzen

Hat ein Nutzer sein Passwort vergessen, so kann er es bequem per E-Mail-Adresse zurücksetzen. Dafür muss er auf der Login-Seite lediglich auf „Passwort zurücksetzen“ klicken. Ihm wird anschließend eine E-Mail mit einem Code, so wie bei der Verifizierung der E-Mail-Adresse, zugesandt. Über die entsprechende Seite kann der Nutzer mit einem gültigen Code ein neues Passwort festlegen.

Der persönliche Pin ist dabei notwendig, um das Zurücksetzen des eigenen Passwortes durch Außenstehende zu verhindern. Theoretisch kann nämlich jeder über das Formular eine E-Mail anfordern, jedoch erhält nur der Besitzer des Postfaches den Pin. Dies hat er durch die Verifizierung zu Beginn sichergestellt. Erhält man folglich eine E-Mail, obwohl man das Passwort nicht zurücksetzen wollte, so ist dies ein Anzeichen für einen Versuch eines unautorisierten Zugriffs.

Authentifizierung & Sitzungsmanagement

Um sicherzustellen, dass nur authentifizierte Nutzer Zugriff auf das System haben, muss der Server überprüfen können, ob die Anfrage von einer berechtigten Person kommt. Für die Realisierung nutzen wir JSON Web Tokens, kurz JWT. Aus diesen kann außerdem die Email-Adresse ausgelesen werden, sodass der Nutzeraccount direkt aus dem Token geladen werden kann. Die JWTs sind immer wie folgt aufgebaut:

xxxxx.yyyyy.zzzzz

Die unterschiedlichen Teile des Tokens werden jeweils mit einem Punkt voneinander getrennt. Der erste Teil ist der Header, in dem der Algorithmus zur Erstellung des Tokens und der Typ gespeichert wird. In unserem Fall ist dies „HS256“ und als Typ „JWT“. Im Payload, dem folgenden Teil, werden nützliche Informationen als Base64Url kodiert. Dies umfasst den Aussteller des Tokens, das Ablaufdatum und die Email im Subject. Dabei handelt es sich um die „registered Claims“. Theoretisch könnten hier beliebig viele weitere private Claims angelegt werden. Damit die Tokens nun als gültig oder ungültig identifiziert werden können, wird noch der letzte Teil, die Signature benötigt, welche mithilfe eines SECRET_KEYS bei der Ausstellung des Tokens erstellt wurde.

Bei einem erfolgreichen Login sendet der Server als Antwort ein Token, welches zehn Minuten lang gültig ist und im Anschluss noch fünf Minuten verlängert werden kann, an den Client. Dieses wird je nach Verfügbarkeit im Session- bzw. Localstorage gespeichert. Bei jeder folgenden Request an den Server, wird dieses dann im Authorization-Header der HTTP-Request mitgesandt. Um es nicht manuell jeder Request hinzufügen zu müssen, nutzen wir den Angular HTTP-Interceptor, der den Authorization-Header bei jeder Request selbständig setzt. Ist das Token gültig, so wird die angefragte Ressource übergeben. Ist das Token jedoch ungültig und kann nicht mehr erneuert werden, so wird direkt eine Response mit dem Status 401 „Unauthorized“ an den Client gesandt. So haben wir auch ein Sitzungsmanagement mit Session-Timeout realisiert. Vor erneuter Abfrage von Daten muss der Nutzer sich erneut einloggen.

Die im Payload kodierte E-Mail stellt des Weiteren sicher, dass Nutzer nur Zugriff zu ihren eigenen Ressourcen haben, weil das Backend die E-Mail aus dem Payload des Tokens dekodiert und sie nicht als Klartext im Body einer POST-Request steht. Um ein gültiges Token mit einer anderen E-Mail zu erhalten, müsste sich der Nutzer mit den zugehörigen Account-Daten im System anmelden.

2-Faktor-Authentifizierung

Heutzutage besteht bei den meisten Applikationen die Möglichkeit einer 2-Faktor-Authentifizierung. Dies erhöht die Sicherheit, da Unbefugten der Zugang nochmals erschwert wird. Selbst wenn ein Unbefugter die Zugangsdaten eines Nutzers erlangt hat, muss er beim Login-Prozess mithilfe eines Codes bestätigen, dass er Zugriff auf das zugehörige E-Mail-Postfach hat.

Wir aktivieren die 2-Faktor-Authentifizierung für Bewerber nicht standardmäßig, da es dem Nutzer selbst überlassen bleiben soll. Um Sicherheit des Unternehmens zu gewährleisten, ist sie bei Personaler immer an. Möchte ein Bewerber seine Daten besser schützen und nimmt dafür jedoch einen etwas längeren Anmelde-Vorgang in Kauf, so kann er die 2-Faktor-Authentifizierung in den Einstellungen aktivieren. Ab dem nächsten Login-Vorgang erhält er immer eine E-Mail mit einem Code, der zur Verifizierung des Vorgangs eingegeben werden muss, bevor er Zugriff auf alle weiteren Funktionen erhält.

1.3 Nachbetrachtung

Welche Ziele wurden erreicht?

Insgesamt lässt sich sagen, dass auf Grund der Vorabitur- und Abiturklausuren nur wenig Zeit für das Projekt übriggeblieben ist. Große Teile des Projektes konnten dadurch erst in der letzten Woche vor Abgabe finalisiert werden, vor allem seitens des Frontends. Daraus resultiert, dass nur einige zusätzliche Features implementiert werden konnten und die Analyse von Bugs nur sporadisch erfolgt ist.

Sehr stolz sind wir beim Frontend auf unsere Login-Page sowie auf die Landing-Page mit der Job-Suche. Diese beiden Seiten verbinden Funktionalität mit einem schönen Design. Generell ist das Frontend zudem sehr effizient durch Feature-Modules aufgebaut, was die User-Experience durch kürze Ladezeiten erhöht.

Aber vor allem die Funktionalität des Backends ist sehr umfangreich. Besonders hervorzuheben ist dabei unsere Art und Weise, wie wir Dateien speichern. Letztes Jahr haben wir diese noch als BLOB in der JavaDB gespeichert, was allerdings sehr ineffizient ist. Nun haben wir es geschafft, dass die Dateien im internen Filesystem von Glassfish speichert werden. Dadurch wird die Geschwindigkeit erhöht und die Datenbank entlastet. Ebenfalls können wir mit der Hilfe einer externen API ein Umkreissuche realisieren, welche vor allem bei großen Unternehmen mit mehreren Standorten eine Erleichterung für die Bewerber darstellt. Das Einbinden hat uns mehrere Wochen gekostet, da Java API-Anfragen standardmäßig blockiert und die nötigen Zertifikate manuell installiert werden mussten. Eine weitere Verbesserung unseres Backends zum Vorjahr ist es, dass Routen nun Backendseitig auf die Berechtigungen des Clients überprüft werden. Dies ist besser, als wenn dies nur Frontendseitig getan wird.

Wo besteht noch Verbesserungsbedarf? Welche Features hätten wir gerne noch eingebaut?

Insgesamt besteht auf Grund des hohen Zeitdrucks noch viel Optimierungsbedarf. Im Folgenden wollen wir einige der wichtigsten Punkte nennen:

- Ein wichtiger Punkt ist die Anpassung der Webseite an alle möglichen Bildschirmgrößen. Wir haben unser Design nach dem Desktop-First- Prinzip ausgelegt, da man sich mit Bewerbungen vor allem an einem Desktop-PC auseinandersetz. Auch haben wir die Oberfläche an kleinere Bildschirme angepasst, aber einige Aspekte, wie zum Beispiel Fehler-Meldungen, weggelassen.
- Der Punkt Fehler-Meldungen und generell gesprochen der Punkt User-Feedback ist auch ein wichtiger Punkt, für den wir nur wenig Zeit haben. Zum Punkt User-Feedback gehören neben Fehler-Meldungen zum Beispiel auch Ladeanimationen, welche wir bisher gar nicht implementiert haben. Besonders wichtig ist aber auch eine Rückgabe, ob die Eingabe in ein Textfeld richtig ist. Dies ist bisher nur in der Login-Page eingebaut, sollte aber auf jeden Fall noch auf alle anderen Ansichten ausgeweitet wird. Die richtige Fehlermeldung kommt bereits vom Backend zurück, wird allerdings noch nicht richtig angezeigt.
- Ein weiterer bekannter Fehler ist zurzeit, dass die Umkreissuche nicht auf dem Betriebssystem Windows funktioniert. Dies liegt daran, dass im Gegensatz zu Unix-basierten Systemen, die nötigen Zertifikate nicht installiert werden können.

Zudem können wir uns auch noch einige zusätzliche Funktionen vorstellen, welche zum Teil bereits im Backend vorhanden sind:

- Auf der Startseite werden bereits angepinnte Jobs angezeigt. Dies könnte man noch auf ganze Fachbereiche ausweiten, wenn man hier dringend Mitarbeiter sucht.
- Momentan sind die verfügbaren Abteilungen und Anstellungsverträge vornherein festgelegt. Dies sollte man umbauen, sodass der Chef der Firma zum Beispiel neue Abteilungen selber angelegten kann.

- Generell haben wir uns bereits eine komplexe Hierarchie überlegt, um ein großes Unternehmen bestmöglich darzustellen. Dieses System enthält bereits die Funktion von eigenen Teams mit eigenen Chefs. Diese Teams haben je nach Rang unterschiedliche Zugriffsrechte, welche jedoch im Frontend noch nicht umgesetzt wurden. Dieses System sollte man auf jeden Fall noch ausbauen, da es die Arbeit beim Verwalten von Bewerbungen deutlich erleichtern kann.
- Auch die Bereitstellung unserer Applikation als Docker-Image für eine einfachen Inbetriebnahme haben wir dieses Mal leider nicht geschafft.

Fazit

Trotz hohen Aufwandes hat uns die Aufgabenstellung sehr viel Spaß gemacht und wir haben eine Menge gelernt. Wir hätte gerne noch einen letzten Feinschliff implementiert, sind aber trotzdem mit unserem Resultat zufrieden und denken, dass wir die Aufgabenstellung sehr gut erfüllt haben. Alle wichtigen Funktionen sind enthalten und auch die Benutzeroberfläche ist aufgeräumt und strukturiert. Die Daten werden effizient verwaltet und unser Code ist durch das Aufteilen in mehrere Dateien ordentlich und verständlich.

2. Benutzerhandbuch

2.1 Installation

Zur Installation des Projektes stellen wir Ihnen zwei Varianten zur Verfügung. Sie können es zum einen über die Netbeans IDE aufsetzen oder zum anderen über Docker. Wir empfehlen Ihnen, sofern Sie mit der Nutzung von Docker vertraut sind, die Installation mit dem Docker-Compose File, da dies weniger Aufwand für Sie bedeutet. Folgen Sie bitte je nach Entscheidung der folgenden Anleitung.

2.1.1 Installation mit Netbeans

Backend

Um das Backend auf diesem Weg aufzusetzen, benötigen Sie wie erwähnt die Netbeans IDE. Wir haben die LTS Version 12.0 mit der JDK 8.281 genutzt. Es ist wichtig, dass Sie keine neuere JDK als Version 8 nutzen, da der von uns genutzte GlassFish Application-Server nur die Versionen bis JDK-8 unterstützt. Netbeans können Sie unter folgendem Link herunterladen: **NetBeans installieren**

Sobald Sie Netbeans installiert haben müssen Sie das Projekt in Netbeans importieren. Dafür klicken Sie oben links auf „File“ und anschließend auf den Reiter „Open Projects“.

Sollten Sie nebenstehende Fehlermeldung erhalten, so klicken Sie bitte auf „Install nb-javac“ und starten Sie die IDE bei Aufforderung neu.

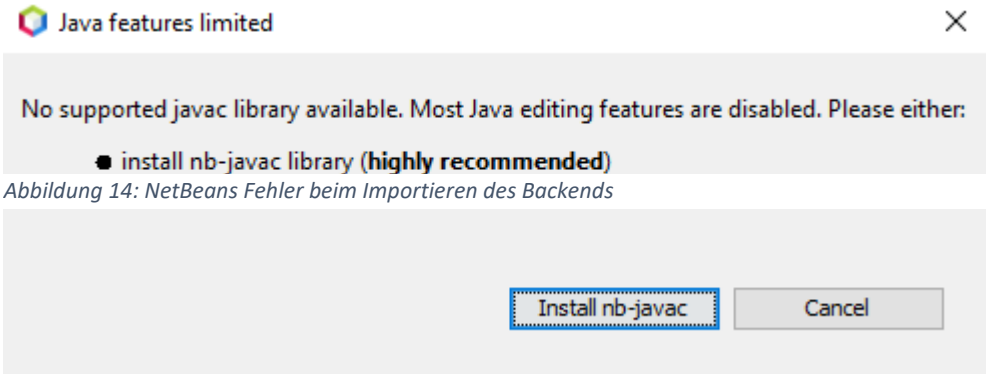


Abbildung 14: NetBeans Fehler beim Importieren des Backends

Haben Sie das Projekt erfolgreich importiert, so benötigen Sie noch den GlassFish Application-Server, um das Backend später für die Nutzung bereitzustellen. Dafür navigieren Sie bitte zum Reiter „Tools“ in der Navigationsleiste und dann zum Reiter „Servers“. In dem sich öffnenden Fenster klicken Sie links unten auf „Add Server“ und wählen „GlassFish Server“ aus. Drücken Sie danach auf „Next“ und wählen Sie als Version 5.1.0 aus. Akzeptieren Sie nun noch die Lizenzbedingungen und klicken Sie auf „Download now“. Der GlassFish Server wird nun automatisch in das richtige Verzeichnis installiert. Mit einem Klick auf „Next“ und „Finish“ schließen Sie das Fenster wieder. Da wir in unserem Projekt auch Emails versenden, es aber in der aktuellen GlassFish Version einen Bug gibt, muss in dem Installationsverzeichnis noch ein Ordner gelöscht werden:

In dem Ordner *C:\Users\<User>\GlassFish_Server\glassfish\modules\endorsed* muss aus dem .jar-File *grizzly-npn-bootstrap* der Ordner „sun“ vollständig entfernt werden. Nutzen Sie dafür einen Encoder ihrer Wahl, beispielsweise WinRAR.

In nächsten Schritt müssen Sie nun noch eine Datenbank einrichten. Hierzu haben wir die in Netbeans integrierte JavaDB genutzt.

Die Datenbank muss manuell mit dem Skript „db.sql“ eingerichtet werden (in Z.213 muss die Emailadresse des Ober-Chefs eingetragen werden, auf die auch Zugriff besteht, da sonst die automatisch aktivierte 2FA nicht durchgeführt werden kann). Dafür klicken Sie bitte in der Sidebar am linken Bildschirmrand auf den „Services“ Tab und klappen dann den Reiter „Databases“ aus und machen einen Rechtsklick auf „Java DB“. Sollte der Services-Tab bei Ihnen nicht angezeigt werden, so können Sie ihn mit der Tastenkombination Strg + 5 einblenden. Wählen Sie nun „Create Database“. Im neuen Fenster erstellen Sie dann eine Datenbank mit

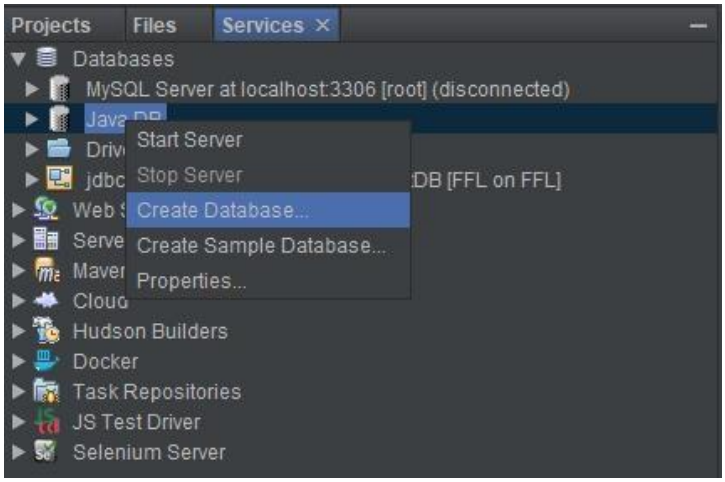
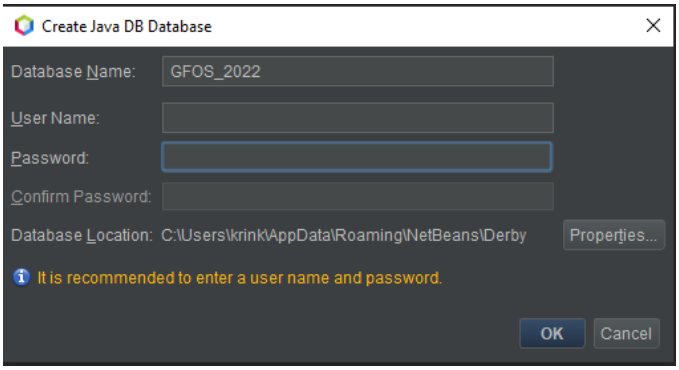


Abbildung 15: Hinzufügen einer neuen Datenbank



dem Namen „GFOS_2022“. Dabei müssen die Felder ”User Name” und ”Password” leer bleiben. Bestätigen Sie ihre Eingaben mit OK.

Führen Sie nun einen Doppelklick auf *jdbc:derby://localhost:1527/GFOS_2022* aus, wodurch Sie sich mit der Datenbank verbinden. Der nach Benutzername und Passwort fragende Dialog kann einfach mit einem Klick auf "OK” bestätigt werden. Um die Tabellen zu erstellen, öffnen Sie bitte links den "Files”-Tab, klappen den Ordner GFOS_2022 aus und öffnen die Datei ”JavaDB_init.sql”. Sollte der Files-Tab bei Ihnen nicht angezeigt werden, so können Sie ihn mit der Tastenkombination Strg + 2 einblenden. In der gerade geöffneten Datei wählen Sie bitte in dem Dropdown-Menü oben in der Mitte die gerade erstellte Datenbank-Verbindung aus (jdbc:derby://localhost:1527/GFOS_2022) und führen diese mit einem Klick auf ”Run SQL” aus .

Abbildung 16: Konfigurieren der Datenbank

Als letzten Schritt müssen nur noch die Ordner zur Dateispeicherung angelegt werden. FInden Sie dafür den GlassFish-Ordner, in dem ihr GlassFish-Server installiert ist. Unter dem Pfad “.../GlassFish_Server/glassfish/domains/domain1/config” erstellen Sie bitte den Ordner “projectFiles” und öffnen Sie diesen. Darin müssen nun vier weitere Ordner erstellt werden: “bewerbungen”, “lebenslaeufe”, “lebenslaufstationen” und “profileimages”. Dabei ist es wichtig, dass die Ordner auch wirklich genauso heißen, da sonst einige Server-Funktionen nicht zur Verfügung stehen.

Das Backend ist nun betriebsbereit. Sie starten den GlassFish-Server mit einem Klick auf das Icon in der Toolbar. Er läuft nun unter Port 8080 auf *localhost*.

Um die EInrichtung des Backends abzuschließen, muss im Browser einmal manuell die Route ”http://localhost:8080/api/anmeldung/setup/test@gfos.de” auf, wobei hier die E-Mailadresse des Chef-Accounts übergeben werden muss. Damit über diesen ein Account ein Login möglich ist, ist es notwendig,

dass die E-Mails dieser E-Mailadresse abgerufen werden können, da die Zweifaktor-Authentifizierung für Personaler immer eingeschaltet ist.

Falls das Backend doch auf einem anderen Betriebssystem als Windows laufen soll, finden Sie hier eine Anleitung, wie die MapBox-Zertifikate Hinzugefügt werden können.
Öffnen sie ein Terminal in dem Ordner "GlassFish_Server/glassfish/domains/domain1/config" und führen Sie die folgenden zwei Befehle aus.
1. true | openssl s_client -connect api.mapbox.com:443 2>/dev/null | openssl x509 > api.mapbox.com
2. keytool -importcert -file api.mapbox.com -alias MapBox -keystore cacerts.jks -storepass changeit
Beantworten Sie beim zweiten Befehl die Frage "Diesem zertifikat vertrauen?" Mit "Ja".
Jetzt ist die Entfernungssuche aktiviert und Java kann API-Anfragen an die MapBox-Server senden.

Frontend

Für das Frontend benötigen Sie die JavaScript-Laufzeitumgebung NodeJS. Sollten Sie diese nicht installiert haben, so laden Sie sie bitte von folgender Seite herunter und folgen Sie den Installationsanweisungen: **NodeJS installieren**. Im Anschluss installieren sie mit dem Befehl „npm install“ im Hauptverzeichnis des Projektes alle benötigten Dependencies. Der Befehl „ng serve -o“ startet die App und öffnet sie im Browser.

2.2 Quick-Start

Im Grunde gibt es zwei verschiedene Anwendertypen. Je nach Berechtigung werden nach einem Login automatisch die Ihnen zur Verfügung stehenden Seiten freigeschaltet.

Bewerber

Als Bewerber haben Sie die Möglichkeit, sich ein personalisiertes Profil zu erstellen, in dem alle relevanten Daten für eine Bewerbung gespeichert werden. So können Sie sich schnell und unkompliziert auf viele Jobs bewerben, die Sie in unserem Portal erkunden können. Unter dem Reiter „Abgeschickte Bewerbungen“ sehen Sie all Ihre abgeschickten Bewerbungen und ihren Status. Außerdem können Sie hier einen Chat mit dem jeweiligen Ansprechpartner beginnen. Ebenfalls können Sie ihre Bewerbung wieder löschen.

Ein Bewerber-Account, auf dem Beispieldaten gespeichert sind, ist tom@mustermann.de mit dem Passwort „gfos2022“.

Personaler

Als Personaler wird ihr Konto von ihrem Chef erstellt. Dieser teilt Ihnen eine Email-Adresse und ein Passwort zu. Anschließend können Sie neue Jobangebote erstellen, eingehende Bewerbungen dazu einsehen und diese annehmen oder ablehnen.

Ein Beispiel für einen Personaler, den höchsten im Rang, ist der Chef-Account mit der von Ihnen gewählten Email-Adresse und ebenfalls dem Passwort „gfos2022“.