PROJEKT-DOKUMENTATION

WEB-APPLIKATION MIT SESSION-HANDLING

MODUL 133, GERD GESELL

Ngoc-Phuc Nguyen, Gian-Luca Kunfermann

INHALT

1. EINFÜHRUNG

1.1. PROJEKTAUFTRAG

Im Modul «Web-Applikation mit Session-Handling» galt es eine Applikation zu entwickeln, bei der ein Backend und ein Frontend in Verbindung stehen und zusammen eine schlüssige Applikation kreieren. Dabei galt es mehrere Punkte zu integrieren:

- Login / Register: Ein User, der noch keine Login Daten besitzt, soll über ein Registrierungsfenster einen Account erstellen können, mit dem man sich auf der Seite einloggen kann. Ist dieser Account nun erstellt oder wurde bereits zu einem früheren Zeitpunkt ein Account erstellt, sollen die in der Datenbank gespeicherten Informationen abgerufen werden.
- Ein eingeloggter User soll mit Session-Handling eingeloggt bleiben und darf so zwischen den verschiedenen Seiten navigieren.
- 3-Tier Architektur: Die Applikation soll in eine «Presentation Layer» (Visuell, Frontend, Design), «Business Layer» (Endpoints, Backend) und in die «Datalayer» (DB) unterteilt sein. Dabei erfüllt jeder Layer ihren eigenen Zweck.
- Das Thema, welches behandelt werden soll, ist uns freigestellt. Wir haben uns dafür entschieden, eine Webseite zu gestalten, bei der ein User aus Bildern, welche von einer API geholt werden, angezeigt bekommt und diese in einer eigenen Liste abspeichern kann. Bilder können aber auch wieder aus dieser Liste gelöscht werden.

Anhand dieser Kriterien und dieser Dokumentation, wird die Arbeit bewertet.

1.2. ENTSCHEIDUNG

Bei der Arbeit wird für das Frontend die Library React (basierend auf Typescript) und für das Backend das Framework Spring (basierend auf Java) verwendet. Für unsere Datenbank verwenden wir PostgreSQL, welche auf einem lokalen Docker Container läuft. Da wir frei sind in der Entscheidung, wie wir unser Projekt aufbauen möchten, haben wir uns dazu entschieden, mit dem was wir bereits kennen zu arbeiten. Zusätzlich sind wir dadurch auch sehr aktuell benutzen keine veralteten Sprachen. Um uns umständliche Arbeit abzunehmen, werden wir eine Komponenten-Bibliothek verwenden, damit wir im Frontend unnötigen Boiler Plate Code umgehen können.

Eine ebenso wichtige Entscheidung war die Aufteilung der Arbeit. Letztendlich haben wir beide sowohl am Frontend als auch am Backend gearbeitet. Man kann aber sagen, dass der Fokus von Phuc auf dem Frontend lag und er dort mehr gemacht hat und Gian-Luca sich mehr auf das Backend fokussiert hat. Bug Fixing wurde hauptsächlich von Phuc übernommen während Gian-Luca sich mehrheitlich an der Dokumentation beteiligt hat.

1.3. INSTALLATION

Wie bereits erwähnt, ist das Projekt in Frontend und Backend unterteilt. Die Installation des Backends ist wie folgt:

Es wird vorausgesetzt, dass eine IDE vorinstalliert ist (vorzugsweise IntelliJ) zur Bearbeitung und starten des Backends, sowie Docker als Container für die Datenbank. Zu Beginn wird ein neuer Container mit dem Befehl:

```
docker run -d -p 5432:5432 -e POSTGRES_PASSWORD=secret123 --name m133db postgres
```

initialisiert, sodass die Daten in einer PostgreSQL Datenbank gespeichert werden. Im Anschluss soll der backend Teil in IntelliJ geöffnet und via Gradle «gebuildet» werden. Sind diese zwei Punkte abgeschlossen, sollte das Backend soweit vorbereitet sein.

Damit das Ganze auch noch dargestellt werden kann, muss ein Frontend vorhanden sein. Dieses wird vorzugsweise über VSC installiert. Dabei soll der Frontend Ordner in VSC geöffnet werden und in der Konsole soll «yarn install» geschrieben werden. Nach Beendigung der Installation, kann mit «yarn start» die Applikation gestartet werden und sollte über den Localhost in einem Browser geöffnet werden.

1.4. EXTERNE RESSOURCEN

MUI: Material-UI ist eine React Komponenten Bibliothek, die es uns ermöglicht vorgefertigte Komponenten in unsere Applikation zu integrieren, ohne sie von Grund auf selbst zu programmieren. Dabei sind insbesondere Komponenten wie Textfelder oder Knöpfe gefragt, aber auch Hilfen zur Darstellung wie «Grid».

API (picsum.photos): Anstelle eines Suchfensters oder einer Ablage, bei der eigene Bilder hochgeladen werden können, erhält ein Benutzer eine grosse Auswahl an Bildern, die von einer API «gefetched» werden. Diese werden dem Benutzer angezeigt und können anschliessend in der eigenen Liste abgespeichert werden.

2. SOFTWARE-ARCHITEKTUR

2.1. 3-TIER-APPLIKATION

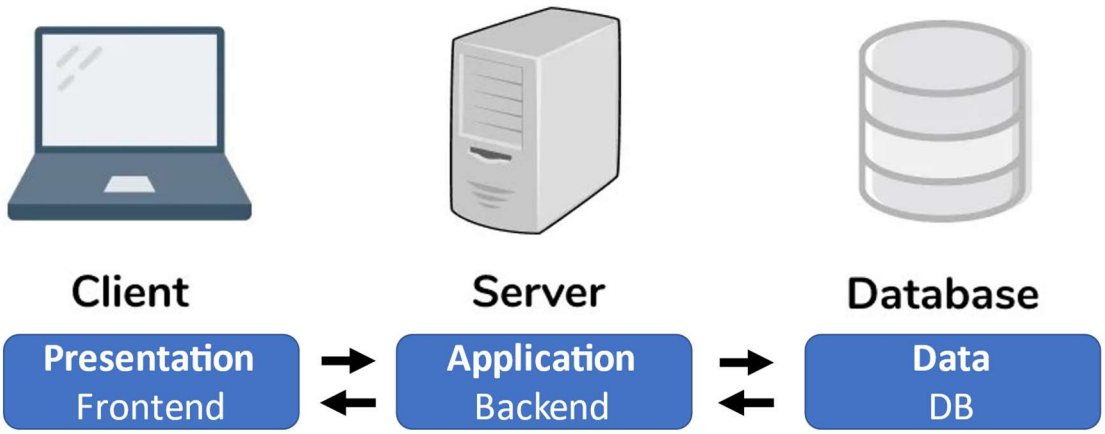


Abbildung 1 Aufbau einer 3-Tier-Applikation

Die gesamte Applikation ist in eine 3 Tier Struktur unterteilt. Wichtig ist, dass jeder Layer separat geführt wird und ihre eigene Daseinsberechtigung hat. Auch wenn alle stark miteinander agieren und man so eine Fullstack Applikation erhält, muss man darauf achten, die unterschiedlichen Layer nicht zu vermischen, so dass bspw. Frontend Komponenten im Backend verwendet werden. Es sollen lediglich Daten von einem Layer in die nächste geschickt werden und dann entsprechend weiterbearbeitet werden.

In unserem Falle läuft das gesamte Frontend über React. Wenn nun ein «http request» getätigt wird, bspw. ein POST bei einem Login-Versuch, werden die Daten des Logins (username und password) an den entsprechenden Endpoint des Backends geschickt. Dort wird ein Abgleich mit den Daten in der Datenbank getätigt und bei einem erfolgreichen Abgleich, wird dem User ein JSON Web Token geschickt, welches eine bestimmte Zeit gültig ist, damit sich der User nicht bei jedem weiteren request neu einloggen muss.

2.2. 2.2 SINGLE-PAGE-APPLIKATION

Webseiten können auf zwei verschiedene Arten erstellt und unterhalten werden. Die etwas in die Jahre Gekommene wäre eine Multi-Page Applikation. Dabei wird für jede Seite der Applikation ein neues «index» File erstellt und bei einem Wechsel wird eine ganz neue Seite geladen. Dadurch muss bei jeder Anfrage an den Server die gesamte Seite wieder komplett geliefert werden, was schnell zu einer Überlastung des Servers führen kann. Deswegen haben wir uns für die moderne Variante, die Single-Page Applikation entschieden. Dabei wird die Seite nur beim ersten request komplett geladen und bei weiteren Abfragen wird nur der nötige Bereich neu geladen. Ebenso werden verschiedene Seiten via Routing angesteuert.

3. FUNKTIONSWEISE

3.1. BESCHREIBUNG

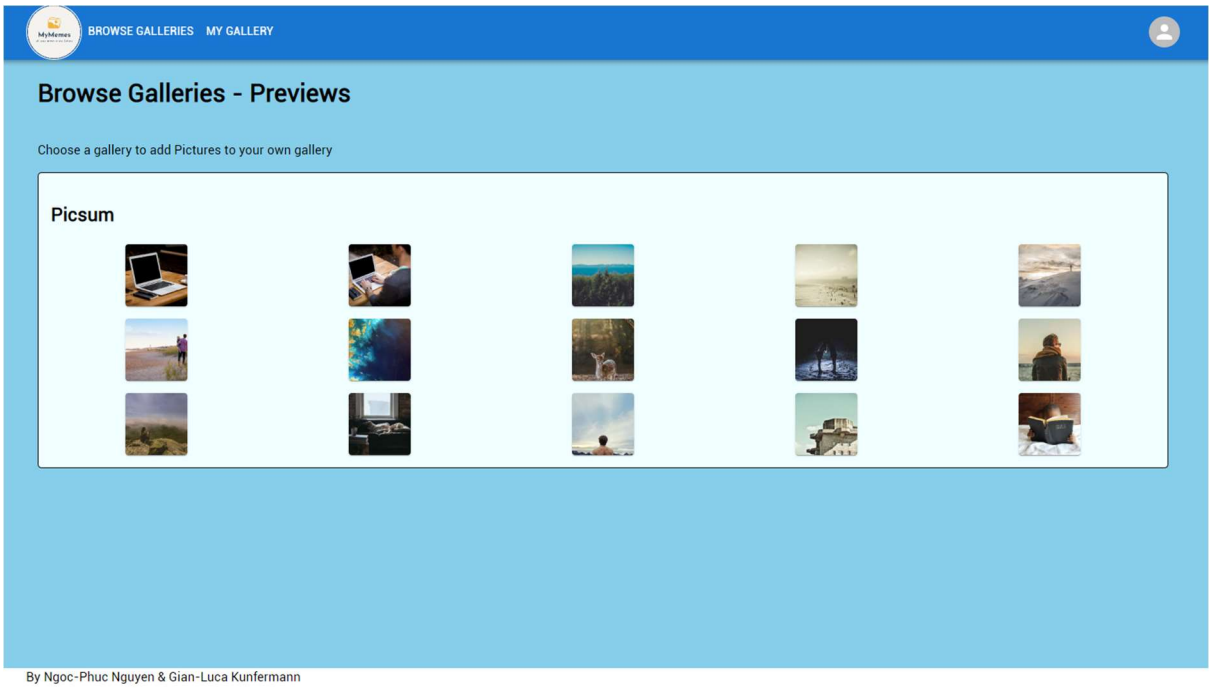
Unsere Webapplikation: «MyMeme Gallery», ist kleines Verwaltungstool, mit der sich Bilder in einer Galerie darstellen lassen. Dabei ist zu beachten, dass wir kein File-Hosting betreiben, sondern lediglich die URLs der Bilder bei unseren Usern abspeichern. Die Bilder können von unserer API von picsum ausgesucht und zur eigenen Galerie hinzugefügt werden. Momentan ist kann ein User nur Bilder der API picsum auswählen.

3.2. AUFBAU

Unsere Applikation hat folgende Seiten, die öffentlich – also ohne Registrierung – zugänglich sind:

- Browse-Galleries-Seite: Hauptseite

Auf dieser Seite gelangt der Besucher, wenn er unsere Applikation über die Hauptdomäne erreicht.



Hier kann der Besucher eine Vorschau an Bildern der verschiedenen APIs anschauen. Bis jetzt ist nur die API picsum verfügbar. Per Klick auf die entsprechende API gelangt der User zu einer detaillierteren Ansicht der API, bei der er sich dann Bilder für seine eigene Galerie aussuchen kann. Der Gast wird hingegen zur Login-Seite geführt.

- Login-Seite

Auf dieser Seite kann sich ein registrierter User authentifizieren, sofern er über die nötigen Anmeldedaten verfügt. Beim Erfolg gelangt der User auf die Hauptseite. Bei Misserfolg wird der User darüber in Kenntnis gesetzt. Falls ein Gast sich einen Account erstellen möchte, gelangt er über einen kleinen Link unterhalb des Login-Knopfes auf die Registrierungs-Seite.

- Registrierungs-Seite

Diese Seite ähnelt vom Aussehen her der Login-Seite. Nach der Eingabe der gewünschten Angaben, wird der User zur Login-Seite zurückgeführt und kann sich anmelden.

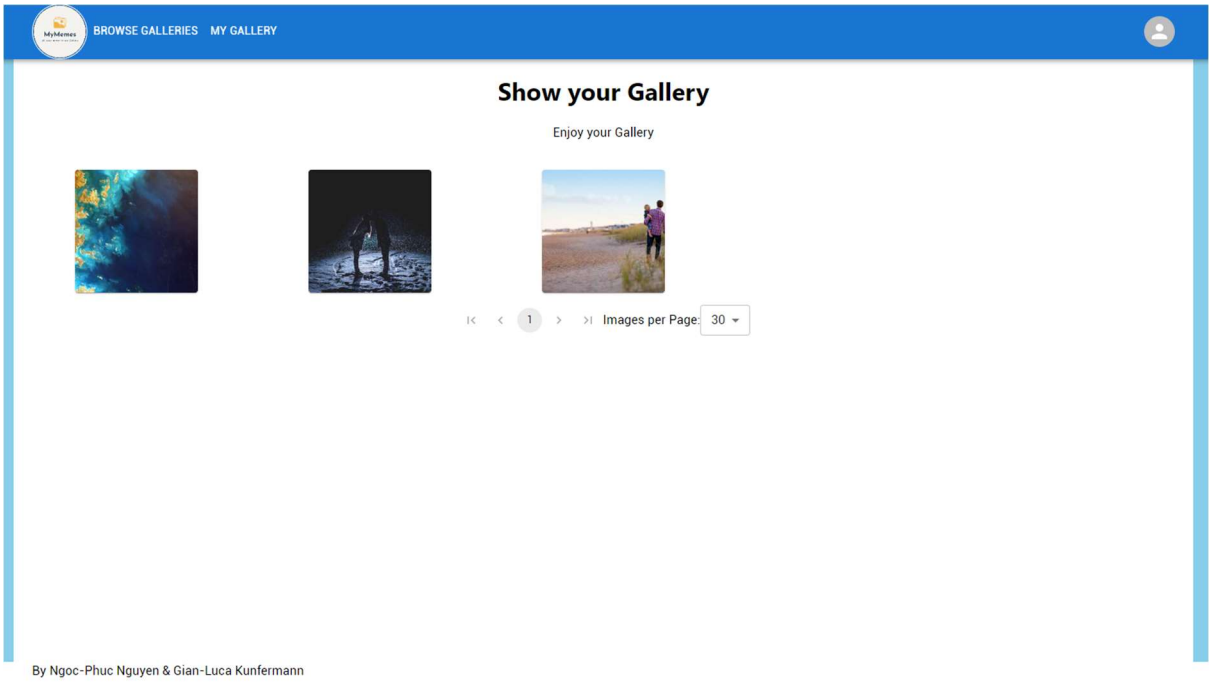
Folgende Seite benötigen eine Authentifizierung:

- Galerie-Ansicht-Seite

Hier ist die Galerie der gewählten API grösser dargestellt. Ebenso stehen dem User hier bis zu 1'000 Bilder zur Auswahl – sofern die API so viele Bilder anbietet - im Vergleich zu den 15 der Vorschau. Falls der User auf dieser Seite mit der Maus über ein Bild fährt, erscheint ein Knopf, mit der sich das Bild zur eigenen Galerie hinzufügen lässt. Ansonsten lässt sich per Klick auf das Bild ein neues Tab öffnen, bei der das Bild in sehr gross angezeigt wird. Ebenso werden pro Seite immer nur eine bestimmte Anzahl Bilder dargestellt und man kann zwischen verschiedenen Tabs mit neuen Bildern wechseln.

- Eigene-Galerie-Seite

Zu guter Letzt sollten die Bilder, welche man seiner Datenbank hinzugefügt hat, auch irgendwo angezeigt werden. Dies geschieht auf dieser Seite. Dabei werden die Bilder in einer zufälligen Reihenfolge angezeigt. Wir haben uns aktiv dafür entschieden, damit es eine grössere Variation gibt. Bilder die vorhin in der Galerie Ansicht hinzugefügt worden sind, können in dieser Ansicht wieder gelöscht werden. Dabei muss der Benutzer über das gewünschte Bild hovern, wobei ein Minus Button erscheint. Mit Klick auf diesen Button wird das Bild aus der eigenen Liste entfernt und wird auch nicht mehr angezeigt.



4. FRONTEND

4.1. REACT TYPESCRIPT

React ist nicht wie viele glauben ein Framework, sondern eine Library, mit der man basierend auf Javascript, oder wie in unserem Falle Typescript, Applikationen erstellen kann. Dabei können viele interaktive UIs auf Basis vieler kleinerer Komponenten erstellt werden. Ein wichtiger Bestandteil von React sind Props, also Daten, welche in Komponenten verwendet und bearbeitet werden. Dies geschieht über «hooks», welche React zur Verfügung stellt. Insbesondere die zwei «hooks» useState und useEffect, welche gebraucht werden, um einerseits einen Status zu tracken und mit einer Funktion anzupassen und andererseits, um die Seite zu rendern. Wir möchten hier aber nicht allzu sehr ins Detail gehen, was React alles kann. Dabei möchten wir lieber vorzeigen, was wir daraus gemacht haben.

4.2. ATOMIC DESIGN

Grundsätzlich werden in React grössere oder kleinere Bausteine in Komponenten unterteilt. Dabei kann man aber noch einen Schritt weiter gehen. Brad Frost hat sich vor einigen Jahren ein Design überlegt, welches stark an die Realität erinnert. Die kleinsten Komponenten, welche nicht weiter zerkleinert werden können, werden Atome genannt. In diese Sparte fallen bspw. Knöpfe. Geht man einen Schritt weiter, entstehen aus mehreren Atomen Moleküle. Diese werden dann weiter zu Organismen kombiniert und zu guter Letzt zu Seiten zusammengespannt. Diese weitere Unterteilung erlaubt es uns, einen besseren Überblick über die verwendeten Komponenten zu wahren und diese perfekt für die Wiederverwendung bereit zu halten.

4.3. CODE BEISPIEL

Um zu zeigen, wie wir das Frontend implementiert haben, werden wir Code-Snippets mit kurzen Texten beschreiben:

```typescript
import axios, { AxiosInstance } from "axios";
const api: AxiosInstance = axios.create({
    baseURL: "http://localhost:8080"
});

api.interceptors.request.use(
    (request) => {
        const token = localStorage.getItem("accessToken");
        if (token) {
            if (request.headers) {
                request.headers.Authorization = token;
            } else {
                return false;
            }
        }
        return request;
    },
    (error) => {
        return Promise.reject(error);
    }
);

export default api;
```

Die Verbindung zu unserem Backend wurde mit Axios gelöst. Dabei können wir letztendlich auf die Endpoints unseres Backends zugreifen und die Daten entweder an die Datenbank senden oder von der Datenbank holen.

```tsx
interface FormValues {
  username: string;
  password: string;
}

const validationSchema = yup.object().shape({
  username: yup.string().required("Please enter your Username"),
  password: yup.string().required("Please enter your Password"),
});

const LoginBox = () => {
  const navigate = useNavigate();
  const { showSnackBar } = useContext(SnackBarContext);

  return (
    <Box sx={LoginBoxStyle.box}>
      <Grid item sx={LoginBoxStyle.header}>
        <Typography>
          <h2>Login</h2>
        </Typography>
      </Grid>
      <Formik
        initialValues={{
          username: "",
          password: "",
```

Hier zu sehen ist ein Teil einer Komponente, die Box welche auf der Login Seite dargestellt wird. Hier gezeigt wird ein Ausschnitt, bei dem mittels yup eine Validierung direkt bei der Eingabe des Users stattfindet. Ebenso werden Hooks veranschaulicht, welche in React gebraucht werden. Bspw. wird useNavigate verwendet, um zwischen den einzelnen Seiten zu wechseln. Der return zeigt was effektiv im Frontend präsentiert wird, in diesem Falle auch mit Formik, damit die Daten über axios ans Backend übermittelt werden können.

```tsx
type Props = {};

const LoginPage = ({}: Props) => {
  return (
    <Box sx={{ ...PageStyle.defaultPageStyle, ...PageStyle.loginPageStyle }}>
      <LoginTitle></LoginTitle>
      <LoginBox></LoginBox>
    </Box>
  );
};

export default LoginPage;
```

Zu guter Letzt wird eine Seite gezeigt, in diesem Falle die Login Page. Sie besteht aus zwei Komponenten, die wiederum aus kleineren Komponenten besteht (anhand des Atomic Designs erstellt). Das Routing zwischen den Seiten ist dann in diesen kleineren Komponenten gelöst, wie bereits oben gezeigt.

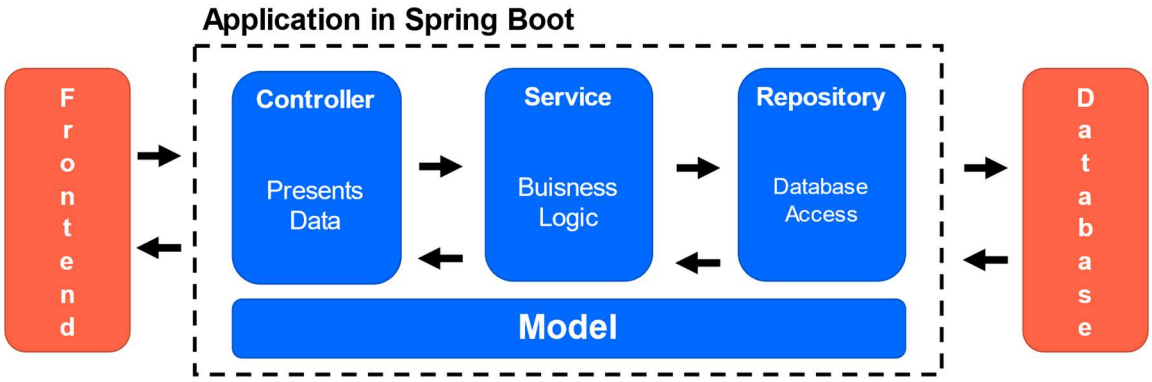
5. BACKEND

5.1. JAVA SPRING BOOT

Spring Boot, oder kurz gesagt einfach Spring, ist ein Framework, welches uns erspart, viel unnötigen Boilerplate Code zu schreiben, aber es gibt uns auch viele weitere Funktionen, die die Arbeit mit einer Datenbank vereinfachen. Spring zeichnet sich dadurch aus, dass Klassen und Methoden mit Annotations versehen werden können, welche uns einerseits vorgefertigte Funktionen bereitstellen, welche direkt angewendet werden und andererseits nimmt es uns die Arbeit durch «Dependency Injection» ab. Die Idee dahinter ist, dass wir die Konstruktion und das Management von Objekten direkt dem Framework überlassen (Beans) und wir gar nichts mehr machen müssen, was dieses Bean anbelangt. Man spricht dann von Inversion of Control through Dependency Injection.

5.2. AUFBAU

Den Aufbau des Spring Backends kann man auch wieder durch eine 3-Tier Struktur darstellen. In diesem Falle sind dies Controller, Service und Repository, wobei diese drei Strukturen immer genau mit dem danebenstehenden Interagieren. Grob gesagt, erhält der Controller über Endpoints, die darin definiert sind, vom Frontend die entsprechenden Daten, diese werden über den Service und das Repository zur Datenbank gebracht. Visuell dargestellt sieht das so aus:

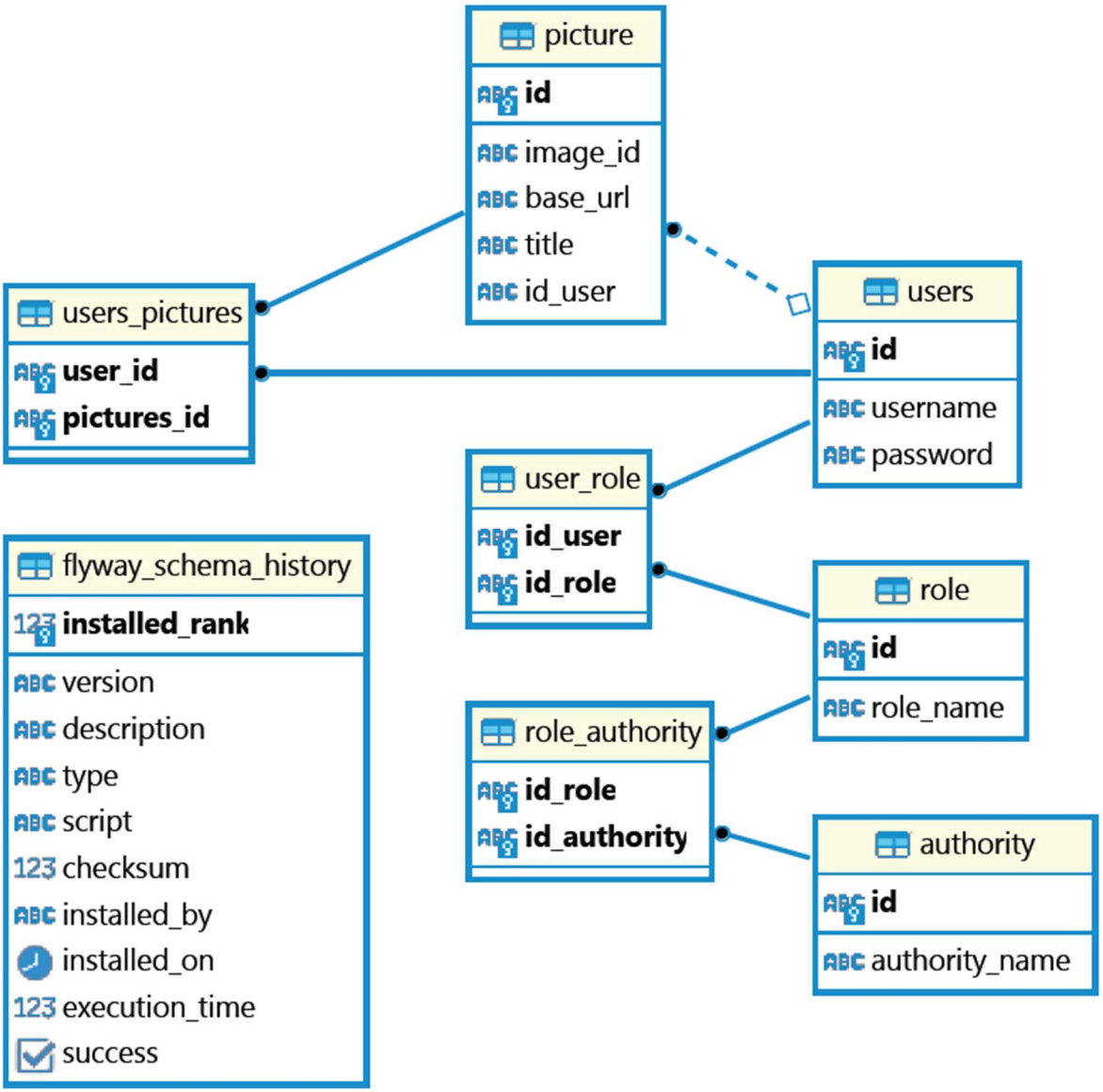


Zusätzlich zum generellen Aufbau können mit folgenden Beispielen genaueres erläutert werden:

5.3. DATABASE

Wie bereits erwähnt, werden unsere Daten in einer PostgreSQL Datenbank gespeichert. Postgres ist eine relationale Datenbank, die wie MariaDB open source ist. Sie entspricht dem SQL-Standard und wurde ursprünglich an einer Universität erstellt. Wir haben uns für diese Art der Datenbank entschieden, da wir bisher immer MariaDB verwendet haben und wir uns vorgenommen haben, für diesen Kurs etwas Neues auszuprobieren. Letztendlich hat sich aber eigentlich nichts Grosses geändert, da die Handhabung der Daten mit DBeaver ausgesprochen leicht und intuitiv war.

Anstatt die Datenbank genau zu beschreiben, werden Funktionalität etc. anhand des ERDs gezeigt:


picture
id
image_id
base_url
title
id_user
users_pictures
user_id
pictures_id
users
id
username
password
user_role
id_user
id_role
flyway_schema_history
installed_rank
version
description
type
script
checksum
installed_by
installed_on
execution_time
success
role
id
role_name
role_authority
id_role
id_authority
authority
id
authority_name

6. TESTING / VALIDIERUNG

Wir haben uns dazu entschieden, dass backend nicht mit umständlichen Unit Tests zu erweitern, sondern das Testing mit Postman durchzuführen und das Frontend mit Validierung zu erweitern.

Mit Postman kann man Abfragen ans backend machen, ohne das Frontend miteinzubeziehen. Dabei werden entsprechende requests direkt an die korrespondierenden Endpoints gestellt und man kann direkt erkennen, ob alles richtig funktioniert, anhand der retournierenden Daten. Insbesondere bei der Erstellung der Endpoints konnte so direkt getestet werden, ob alles richtig funktioniert. Auch falsche Abfragen konnten gemacht werden, um zu sehen, ob das Resultat auch in diesen Fällen den Wünschen entspricht.

Im Frontend hingegen wurden zwei verschieden Arten von Validierung verwendet. Einerseits enthält Formik, also die Komponente von MUI, welche es uns erlaubt, requests ans backend zu schicken, aber auch durch die Integration von Yup. Yup gibt dem Benutzer in Echtzeit Rückmeldung, ob das Eingetragene den Ansprüchen entspricht, oder ob sich bspw. bei der Erstellung eines Passworts Fehler eingeschlichen haben. Zusätzlich überprüft im Nachhinein, wenn diese erste Hürde überwunden wurde, Formik nach, ob auch wirklich alles stimmt. So kann der Benutzer perfekt geleitet werden, auch wenn sich dieser nicht mit der Materia auskennt.

7. SCHLUSSWORT

Auch in diesem Projekt waren wir wieder unsere grössten Gegner. Unser Zeitmanagement war wieder nicht ideal, was uns einiges an Arbeit in kürzerer Zeit abverlangte. Trotzdem können wir sagen, dass wir sehr zufrieden sind mit dem Resultat. Es galt viele Hürden zu überwinden, denn auch wenn wir mit der Grundlage bereits bekannt waren aufgrund unserer Tätigkeit im Büro, mussten wir doch viele neue Dinge erlernen. Insbesondere die Verwendung des JWT erforderte viel Grips und Zeit und auch die Implementation von mehreren Usern, die alle eine eigene Liste an Bildern besitzt, brauchte viel Zeit. Was auch ein Grund war, weswegen es gegen Ende doch ein wenig knapp wurde. Als Abschluss soll gesagt sein, dass wir froh sind, dass wir unsere Fähigkeiten noch weiter verfeinern konnten und wir stolz sind, dass wir dies auch im Alltag brauchen können.