LUISA IHLE

Full-Stack Web Developer

PORTFOLIO

Frontend – Achievement 1

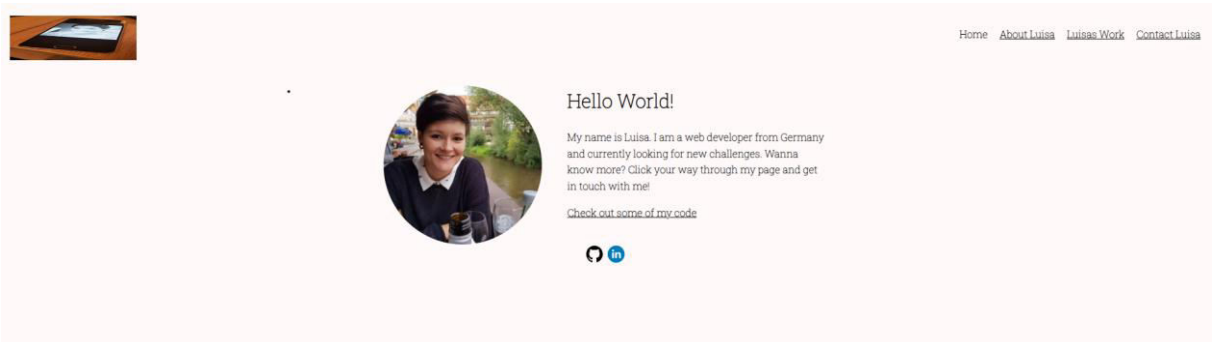
1) Description of the project.

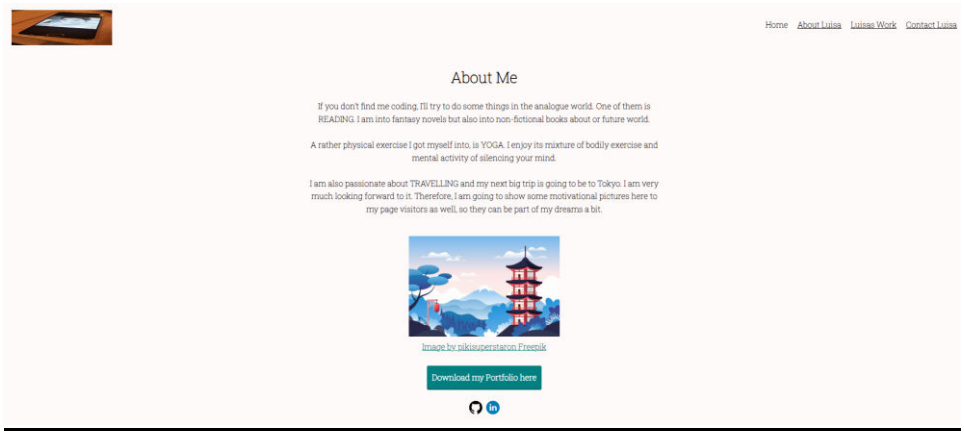
Welcome to my personal portfolio page! Here you'll find information about myself, my past projects, and how to get in touch with me.

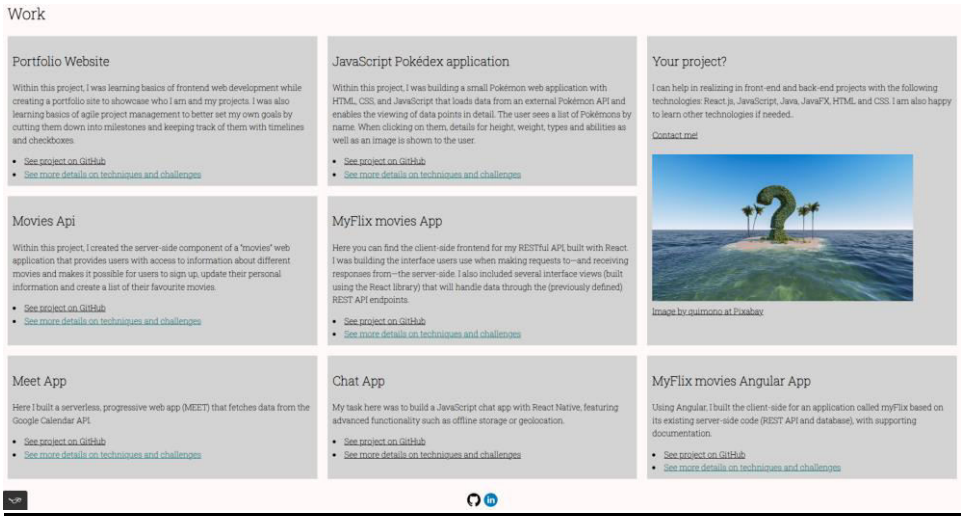
- What was my role for this project and what tasks did I face?
  - learning basics of frontend web development,
  - creating a portfolio site to showcase who I am and my projects,
  - learning basics of agile project management to better set my own goals by cutting them down into milestones and keeping track of them with timelines and checkboxes

- Lessons I learned / decisions I made during this project:
  - having basic knowledge of UX and design is important for a web developer,
  - basics of Agile project management and the Scrum method,
  - estimating time and effort for tasks with story points,
  - how the internet works,
  - how websites are structured,
  - commenting and code indentation,
  - creating HTML pages and linking them to create more complex websites,
  - use ARIA to create dynamic content,
  - the importance of web accessibility and why it's my duty as a web professional to ensure an accessible web experience for all users,
  - applying basic CSS styling to a web page as well as advanced CSS styling such as transitions and animations,
  - exploring CSS preprocessors,
  - version control with Git and GitHub,
  - utilising code linting to prevent errors and ensure consistent code quality,
  - conduct cross-browser testing and hosting.

2) A screenshot to represent the project







3) A link to the project's GitHub repository

- https://github.com/Luisa-Inc/portfolio-website

4) A link to the live, hosted version of my app

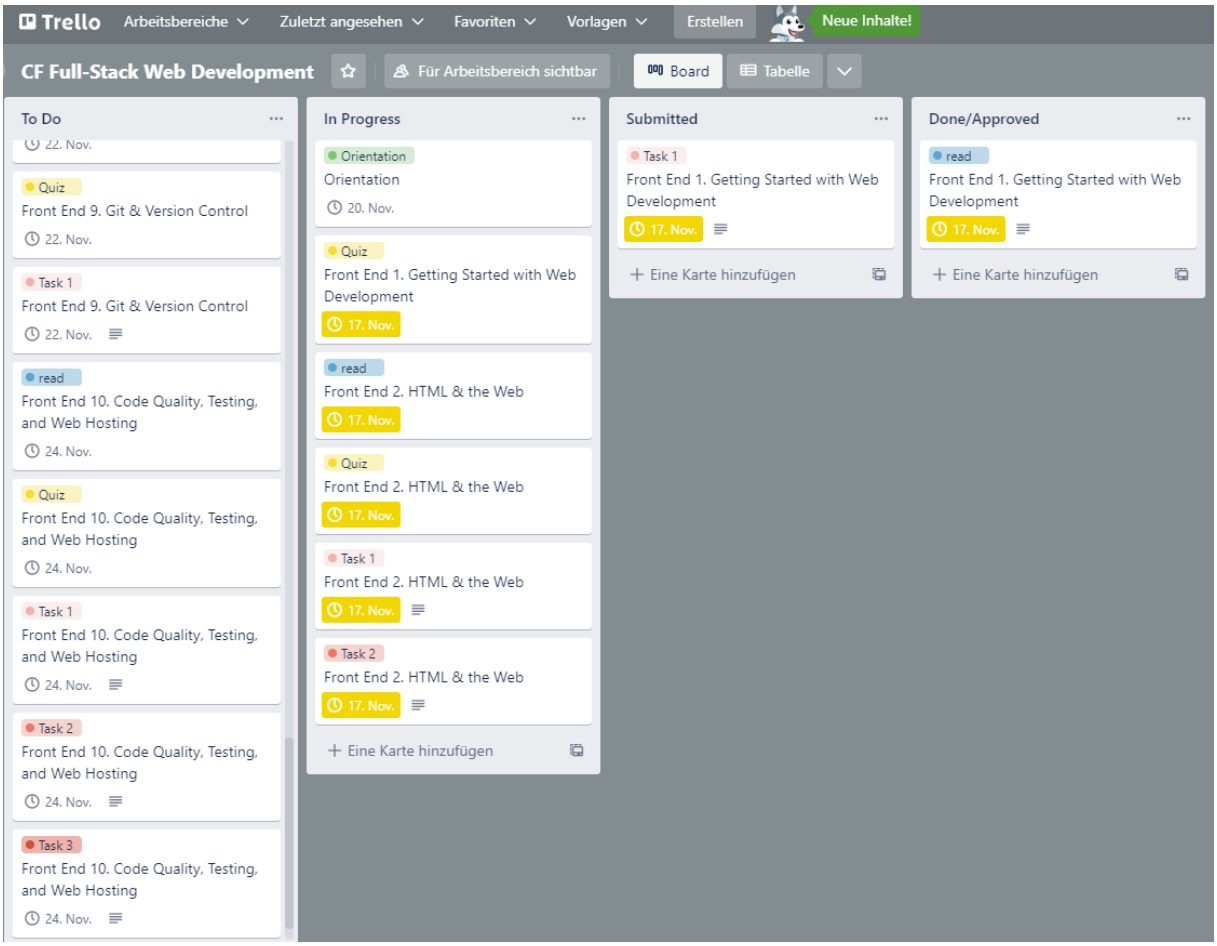
- https://luisa-inc.github.io/portfolio-website/

5) A list of the technologies used for each project

- HTML
- CSS
- Visual Studio Code
- CodePen
- GitHub
- GitDesktop

6) Any other relevant materials I created for the project

- A Kanban board (Trello)
  - visualising all project steps and tasks for this Achievement with a project management tool,
  - setting a to-do-list of which the single items can be moved towards a card “in progress”, “submitted” and “done/approved” to control their process,
  - setting due-dates and labels for me to keep on track and being able to follow a realistic project planning with time for unplanned delays,
  - project management in general was helping me to get better at estimating how long a project can take and to always be prepared for unplanned delays due to technical or human interference (this should be taken into consideration for future employers and how to communicate deadlines and realistic project development with them)



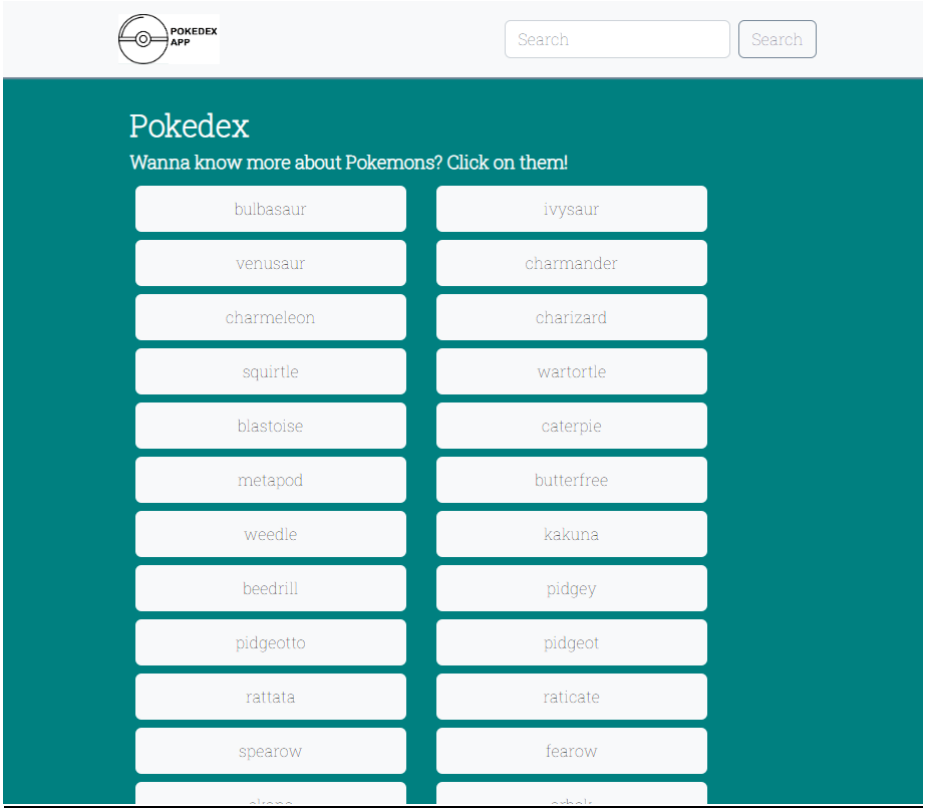
Backend – Achievement 1 – Introduction to JavaScript

1) Description of the project

- What was my role for this project and what tasks did I face?
  - building a small Pokémon web application with HTML, CSS, and JavaScript that loads data from an external Pokémon API and enables the viewing of data points in detail,
  - the user sees a list of Pokémons by name. When clicking on them, details for height, weight, types and abilities as well as an image is shown to the user

- Lessons I learned / decisions I made during this project:
  - basics of JavaScript and the DOM,
  - working with variables, mathematical expressions, primitive and complex data types, conditionals, loops and functions,
  - DOM interaction and manipulation,
  - event handling for web interactivity and accessibility,
  - APIs, Ajax & Asynchronous Behaviour,
  - data visualisation and animations,
  - creating UI Patterns with JavaScript,
  - using jQuery for app development,
  - building responsive application layouts using Bootstrap,
  - testing performance and practicing debugging JavaScript code.

2) A screenshot to represent the project



3) A link to the project's GitHub repository

- https://github.com/Luisa-Inc/Pokedex-App

4) A link to the live, hosted version of my app

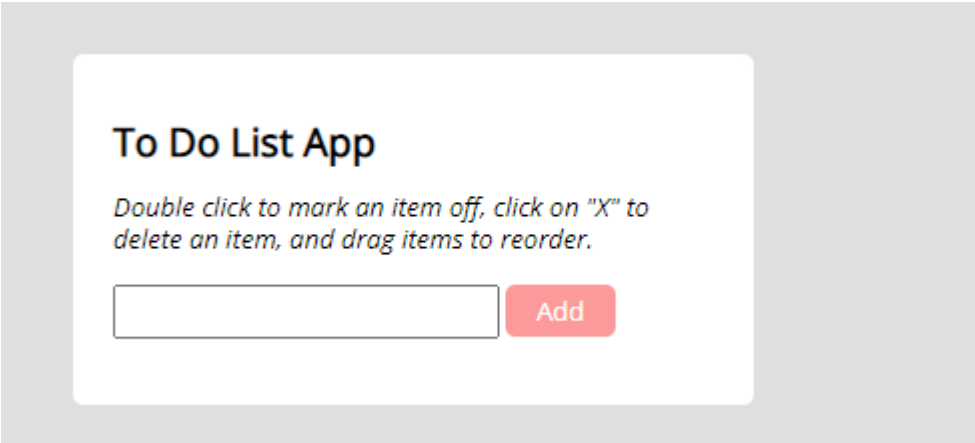
- https://luisa-inc.github.io/Pokedex-App/

5) A list of the technologies used for each project

- HTML
- CSS
- JavaScript
- External PokéAPI

6) Any other relevant materials I created for the project

- A basic To-Do-List app for practicing working with jQuery
  - features:
    - The user can add a new item to a list of items.
    - The user can cross out an item from the list of items.
    - The user can delete an item from the list of items.
    - The user can change the order of items in the list of items.
  - https://github.com/Luisa-Inc/to-do-list-app
  - https://luisa-inc.github.io/to-do-list-app/



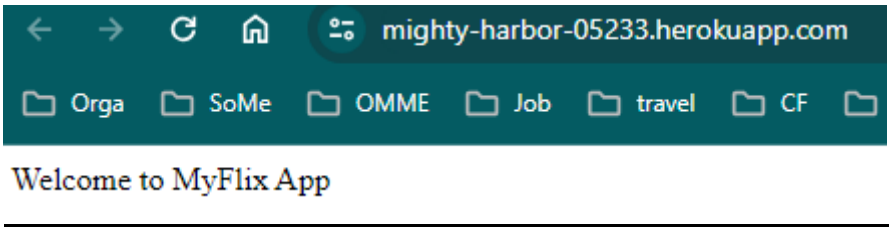
Backend – Achievement 2 – Server-Side Programming & Node.js

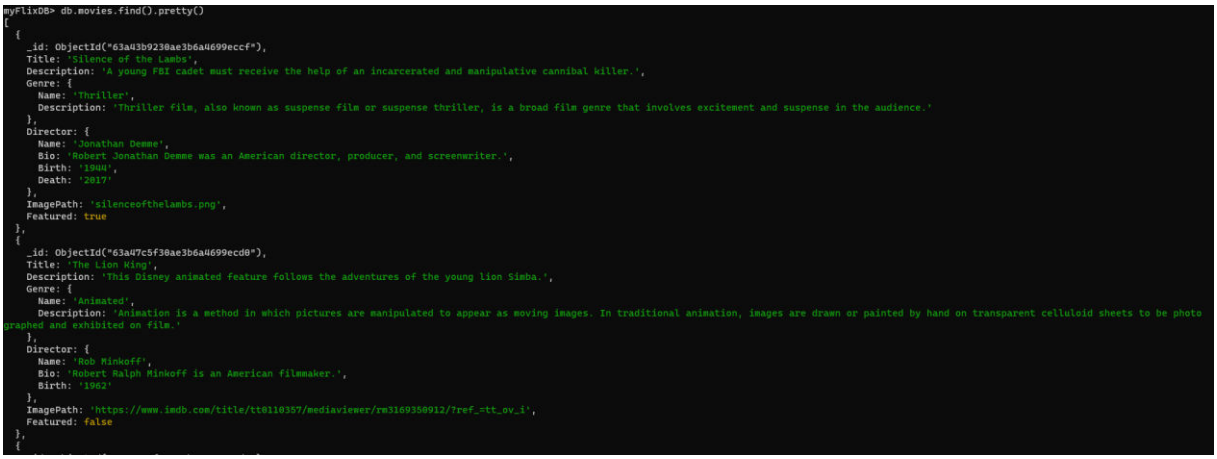
1) Description of the project

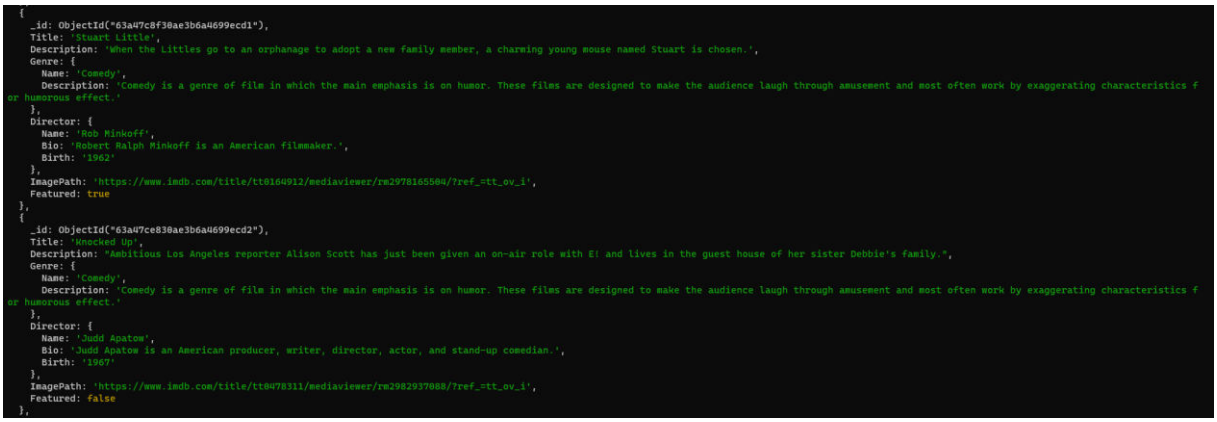
- What was my role for this project and what tasks did I face?
  - Building the server-side component of a “movies” web application that provides users with access to information about different movies and makes it possible for users to sign up, update their personal information and create a list of their favourite movies
  - Feature requirements:
    - return a list of ALL movies to the user,
    - return data (description, genre, director, image URL, whether it’s featured or not) about a single movie by title to the user,
    - return data about a genre (description) by name/title (e.g., “Thriller”),
    - return data about a director (bio, birth year, death year) by name,
    - allow new users to register,
    - allow users to update their user info (username, password, email, date of birth),
    - allow users to add a movie to their list of favourites,
    - allow users to remove a movie from their list of favourites,
    - allow existing users to deregister

- Lessons I learned / decisions I made during this project
  - setting up my project directory,
  - practicing writing Node.js syntax,
  - creating a “package.json” file,
  - importing all necessary packages into project directory,
  - defining my project dependencies,
  - routing HTTP requests for my project using Express,
  - defining the endpoints for my REST API,
  - creating a relational (SQL) database for storing movie data using PostgreSQL,
  - recreating my relational (SQL) database as a non-relational (NoSQL) database using MongoDB,
  - modeling my business logic using Mongoose,
  - implementing authentication and authorisation into my API using basic HTTP, authentication and JWT (token-based) authentication,
  - incorporating data validation logic into my API,
  - implementing data security and storage controls,
  - hosting my project on the web using Heroku,

2) A screenshot to represent the project







3) A link to the project's GitHub repository

- https://github.com/Luisa-Inc/movie_api

4) A link to the live, hosted version of my app

- https://mighty-harbor-05233.herokuapp.com/

5) A list of the technologies used for each project

- Node.js: JavaScript runtime for server-side scripting.
- Express: Back end web application framework for building RESTful APIs with Node.js.
- MongoDB with Mongoose: NoSQL database and Object Data Modeling library for Node.js.
- Postman: Allows you to design, develop, test and monitor APIs.
- body-parser: Express middleware for parsing request bodies.
- express-validator: Middleware for input validation in Express.
- jsonwebtoken: Library for JWT (JSON Web Token) generation and verification.
- lodash: Utility library for JavaScript.
- passport: Authentication middleware for Node.js.
- passport-jwt: Passport strategy for JWT authentication.
- passport-local: Passport strategy for username/password authentication.
- uuid: Library for generating unique identifiers.

Backend – Achievement 3 – Client-Side Programming & React

1) Description of the project

- What was my role for this project and what tasks did I face?
  - building a client-side frontend for my RESTful API, built with React
  - building the interface users use when making requests to—and receiving responses from—the server-side
  - including several interface views (built using the React library) that will handle data through the (previously defined) REST API endpoints

Requirements

Main view

- Returns ALL movies to the user (each movie item with an image, title, and description)
- Filtering the list of movies with a “search” feature
- Ability to select a movie for more details
- Ability to log out
- Ability to navigate to Profile view

Single Movie view

- Returns data (description, genre, director, image) about a single movie to the user
- Allows users to add a movie to their list of favorites
- Login view
- Allows users to log in with a username and password
- Signup view
- Allows new users to register (username, password, email, date of birth)
- Profile view
- Displays user registration details
- Allows users to update their info (username, password, email, date of birth)
- Displays favorite movies
- Allows users to remove a movie from their list of favorites
- Allows existing users to deregister

Optional Views & Features

Actors view

- Allows users to view information about different actors

Genre view

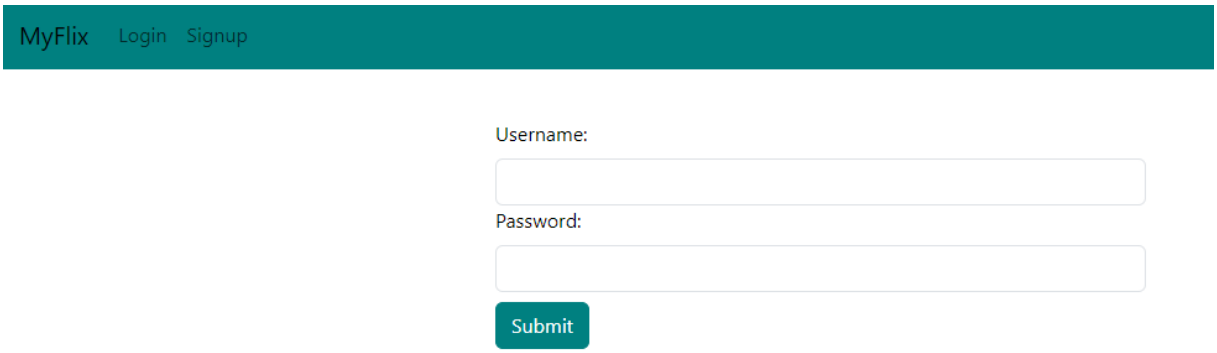
- Returns data about a genre, with a name and description
- Displays example movies

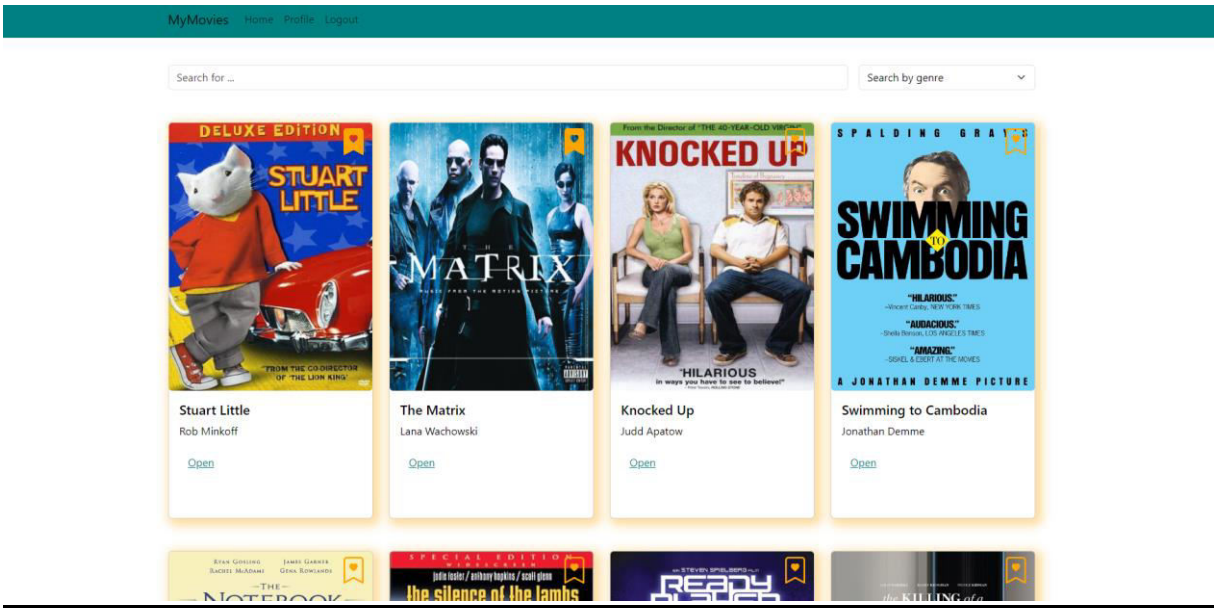
Director view

- Returns data about a director (name, bio, birth year, death year)
- Displays example movies from the director

- Lessons I learned / decisions I made during this project
  - introduction to Frameworks and Libraries and deciding when to use them for my project,
  - getting to know and using React as a framework for my client-side RESTful API,
  - React components such as state, props, the virtual DOM
  - developing components, using Parcel,
  - using Hooks in function components to interact with the React component lifecycle
  - handling flows for signup, login, user authentication, form validation
  - using bootstrap in React
  - client-side app routing, navigation between different views
  - design patterns like redux
  - deployment and hosting
  - readability and maintenance of my codebase and the design and usability of my app

2) A screenshot to represent the project





3) A link to the project's GitHub repository

- https://github.com/Luisa-Inc/myFlix_client

4) A link to the live, hosted version of my app

- https://myluisaflix.netlify.app/

5) A list of the technologies used for each project

- React
- ReactDOM
- React-Router-Dom
- Bootstrap
- React-Bootstrap
- React-Bootstrap-Icons
- Prop-Types
- Moment
- Parcel/Transformer-Sass (v.2.10.2)
- Parcel (v.2.10.2)
- Process

Backend – Achievement 4 – Testing in the Development Process

1) Description of the project

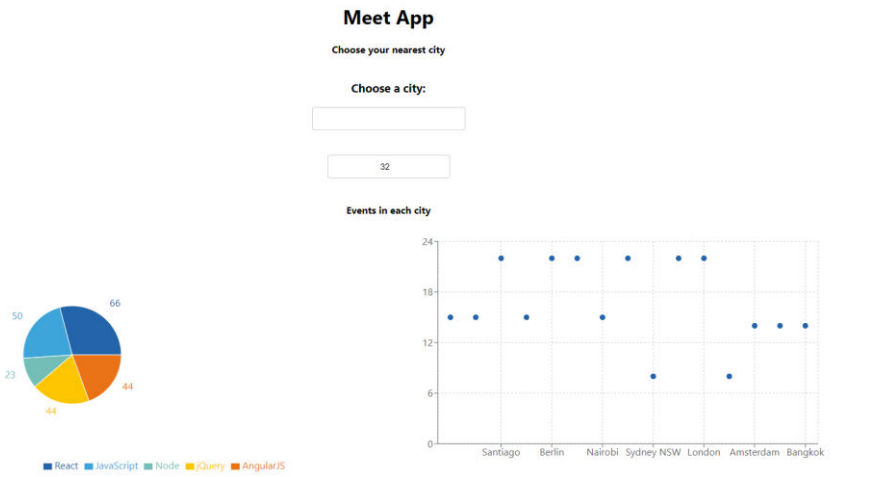
- What was my role for this project and what tasks did I face?
  - built serverless, progressive web app (MEET) that fetches data from the Google Calendar API
    - Serverless: No backend maintenance, easy to scale, always available, no cost for idle time
    - PWAs: Instant loading, offline support, push notifications, “add to home screen” prompt, responsive design, and cross-platform compatibility
- Key Features:
  - Filter events by city
  - Show/hide event details
  - Specify number of events
  - Use the app when offline
  - Add an app shortcut to the home screen
  - View a chart showing the number of upcoming events by city

- Technical Requirements:
  - The app is a React application
  - The app is built using the TDD technique
  - The app uses the Google Calendar API and OAuth2 authentication flow
  - The app uses serverless functions (AWS lambda is preferred) for the authorisation server instead of using a traditional server
  - The app’s code is hosted in a Git repository on GitHub
  - The app works on the latest versions of Chrome, Firefox, Safari, Edge, and Opera, as well as on IE11
  - The app displays well on all screen sizes (including mobile and tablet) widths of 1920px and 320px
  - The app passes Lighthouse’s PWA checklist
  - The app works offline or in slow network conditions with the help of a service worker
  - The users are able to install the app on desktop and add the app to their home screen on mobile
  - The API call used React axios and async/await
  - The app implements an alert system using an OOP approach to show information to the user
  - The app makes use of data visualization
  - The app is covered by tests with a coverage rate >= 90%
  - The app is monitored using an online monitoring tool

- Lessons I learned / decisions I made during this project
  - Wrote user stories based on the app’s key features
  - Translated user stories for each feature into multiple test scenarios
  - Used create-react-app to create a React application and push it to GitHub
  - Obtained a consumer secret from the Google Calendar API
  - Wrote a simple serverless function to refresh the access token
  - Called this function from a static page
  - Used test scenarios, wrote frontend unit tests using mock data for the app’s key features
  - Developed another feature for the app

- Wrote integration tests to test the interaction between the app's React components
- Wrote integration tests to test the data received from the mock API
- Wrote user acceptance tests for two key features, covering all defined test scenarios
- Set up app monitoring for the application to monitor its performance
- Used an OOP approach to create alerts for the application
- Learned advantages and disadvantages of regular web apps and native apps
- Used a service worker to ensure your app works offline
- Used a "manifest.json" file to make my app installable
- Showed a notification to the user to inform them the app is working offline (when the user is offline)
- Added charts, such as a ScatterChart, to my app's UI to visualize data using the recharts library
- Made visualizations responsive

2) A screenshot to represent the project



3) A link to the project's GitHub repository

- https://github.com/Luisa-Inc/meet
- User Stories and Scenarios

4) A link to the live, hosted version of my app

- https://luisa-inc.github.io/meet/

5) A list of the technologies used for each project

- React
- TDD
- Google Calendar API
- OAuth2 flow
- AWS Lambda
- Autorisation server
- Enzyme
- Jest
- Passing Lighthouse's PWA checklist
- React Axios and async/await
- alert system using an OOP approach

6) Any other relevant materials I created for the project

Serverless function

- Created a serverless deployment package with AWS Lambda
- Using the Serverless Toolkit and the Google Calendar API
- own authorization server for issuing OAuth2 access tokens
- Created an OAuth Consumer
- Setup an AWS Lambda Function
- Created an Authentication Server
- Created a Serverless Service
- Configured AWS Credentials
- Deployed the Authentication Process
- Wrote functions:
    - GET AUTH URL: https://59i7ltvzyg.execute-api.eu-central-1.amazonaws.com/dev/api/get-auth-url
    - return: USER AUTHORIZATION CODE: {"authUrl":"https://accounts.google.com/o/oauth2/v2/auth?access_type=offline&scope=https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fcalendar.readonly&response_type=code&client_id=250900529821-rit2jpo282rokk6p0vdrbuo6rcnq0f9o.apps.googleusercontent.com&redirect_uri=https%3A%2F%2FLuisa-Inc.github.io%2Fmeet%2F"}

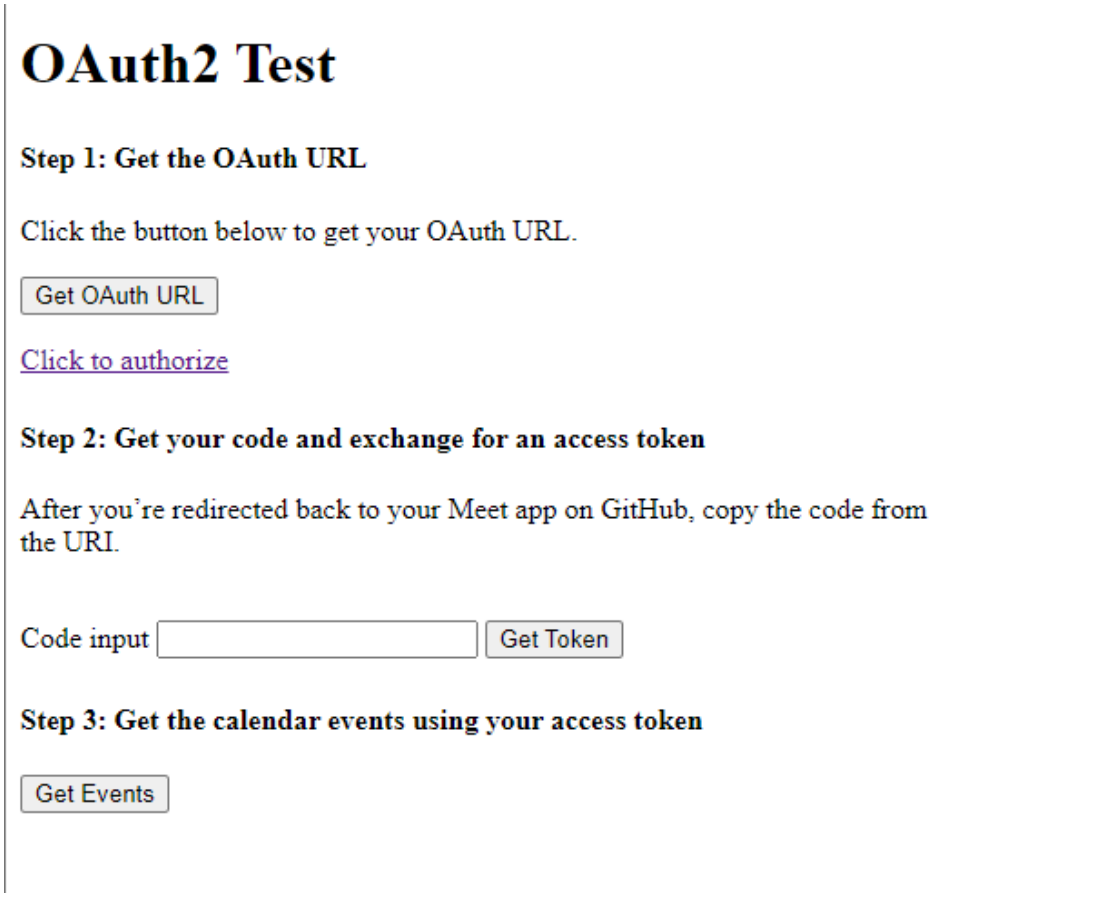
- GET TOKEN: https://59i7ltvzyg.execute-api.eu-central-1.amazonaws.com/dev/api/token/{code}

- return: ACCESS TOKEN: {"access_token":"ya29.a0AVvZVso71dic2NfpOw726rnwnLPYTLoQ5_yf8Pd4kHjjKAM-IlNERg7VONhC7OYRLPMPasmIySwszNTbRCPpL6u2oti2R2BKUFzGInfEafTLylY12EC3T9-K1N8-SKrgpevm5E7TglZsQK7_TrSwta4dTqZpOQaCgYKAcsSARISFQGbdwaI69Zl6BUbmGm3fPlseFKsGA0165","scope":"https://www.googleapis.com/auth/calendar.readonly","token_type":"Bearer","expiry_date":1675335063675}

- GET CALENDAR EVENTS: https://59i7ltvzyg.execute-api.eu-central-1.amazonaws.com/dev/api/get-events/{access_token}

- Tested a Serverless Function Using a Static Site
  - Set up a Local Node.js HTTP Server
  - Created HTML file

- can be found here: https://coach-courses-us.s3.amazonaws.com/exercises/1114/40489/99a9b7da98553b8dfae1cded99e29b2f/test-auth-server.html



Backend – Achievement 5 – Native App Development & React Native

1) Description of the project

- What was my role for this project and what tasks did I face?
  - Building a JavaScript chat app built with React Native, featuring advanced functionality such as offline storage or geolocation

- Key Features
  - A page where users can enter their name and choose a background color for the chat screen before joining the chat.
  - A page displaying the conversation, as well as an input field and submit button.
  - The chat must provide users with two additional communication features:
    - sending images
    - and location data.
  - Data gets stored online and offline
  - Users can enter their name and choose a background color for the chat screen before joining the chat
  - Send and receive messages
  - Send and receive images (from media library or device's camera)
  - Send and receive locations
  - Record, send and receive audio
  - Users can view previous messages when offline

- Technical Requirements

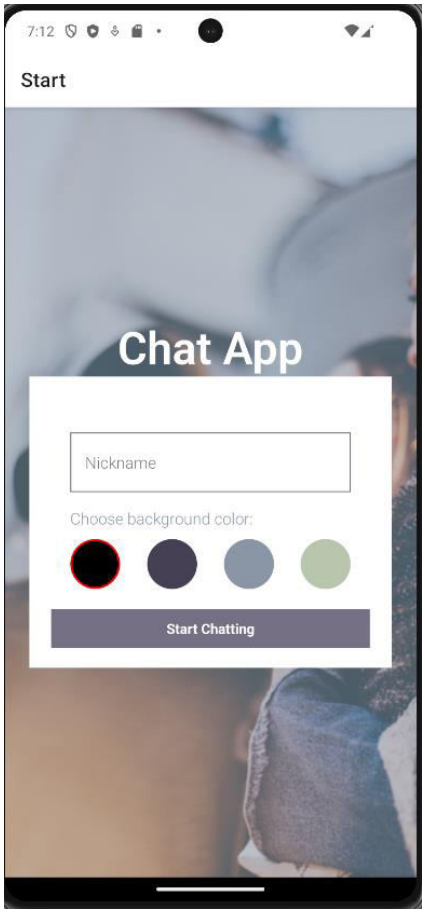
  - The app must be written in React Native.
  - The app must be developed using Expo.
  - The app must be styled according to the given screen design.
  - Chat conversations must be stored in Google Firestore Database.
  - The app must authenticate users anonymously via Google Firebase authentication.
  - Chat conversations must be stored locally.
  - The app must let users pick and send images from the phone’s image library.
  - The app must let users take pictures with the device’s camera app and send them.
  - The app must store images in Firebase Cloud Storage.
  - The app must be able to read the user’s location data.
  - Location data must be sent via the chat in a map view.
  - The chat interface and functionality must be created using the Gifted Chat library.
  - The app’s codebase must contain comments.

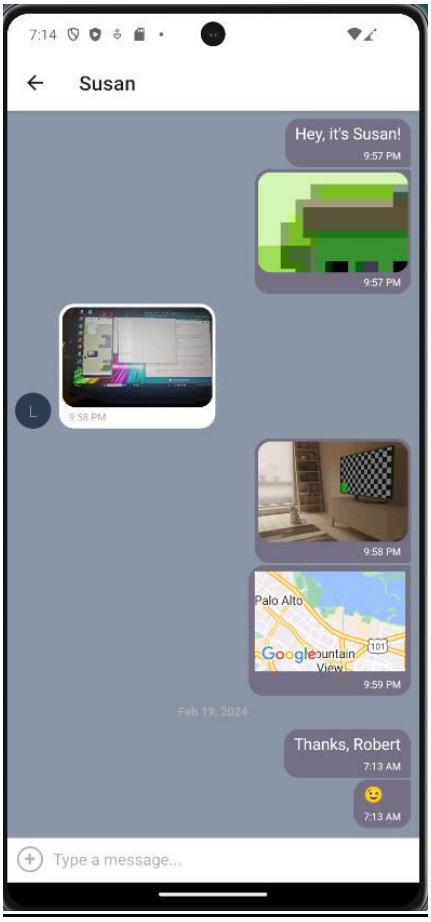
- Lessons I learned / decisions I made during this project

  - I articulated the distinction between native apps and other (web) app models
  - I established a framework, layout, and UI elements for a native app
  - I set up a virtual device to emulate your app
  - I implemented an accessible chat UI for a native app
  - I used and customized pre-build packages in a native app (Gifted Chat)
  - I learned customizing pre-build packages

- I addressed edge cases for different platforms and devices (iOS and Android)
- I compared different possibilities for implementing real-time apps
- I used Firestore to set up a database
- I managed to query a Firestore database in real time via a React Native app
- I rendered data from Firestore in a React Native app
- I implemented client-side data storage in React Native using AsyncStorage
- I described solutions for storing data on the client-side
- I learned how to partially enable some of the app's features when the device is offline using NetInfo
- I implemented communication features into a native app
- I implemented modern JavaScript APIs into an app
- I used and customize a third-party library
- Valuable lesson: Whether I'm developing a website, web app, or native mobile app, I must always think about how I'm handling users' data and respect their privacy—especially when working with a device's camera and microphone. Many users distrust products that use communication features, and it's part of a designer or developer's job to create apps that make the user feel secure. When users give my app access to their camera (or audio), I need to ensure that I don't breach their trust. For example, I must not save the image or video (or audio) files if it's not necessary for my app to do so.

2) A screenshot to represent the project





3) A link to the project's GitHub repository

- https://github.com/Luisa-Inc/chat-demo

4) A link to a video version of my app

- Chat App Demo Video

5) A list of the technologies used for each project

  - React Native: Framework for building mobile applications using JavaScript and React
  - Expo: Development platform for building React Native applications.
  - GiftedChat: A library for creating chat interfaces in React Native applications.
  - Google Firebase: Cloud-based platform that provides various services, including Firestore for real-time database and authentication.
  - AsyncStorage: Local storage system in React Native for caching and persisting data.
  - Expo ImagePicker: Expo API for accessing the device's image picker to choose images from the gallery.
  - Expo MediaLibrary: Expo API for accessing and managing media assets on the device.
  - Expo Location: Expo API for obtaining location information from a device.
  - react-native-maps: React Native Map components for iOS + Android.
  - MapView: Specific component from the react-native-maps library used to display maps in React Native applications.

6) Any other relevant materials I created for the project

- User Stories
  - As a new user, I want to be able to easily enter a chat room so I can quickly start talking to my friends and family.
  - As a user, I want to be able to send messages to my friends and family members to exchange the latest news.
  - As a user, I want to send images to my friends to show them what I'm currently doing.
  - As a user, I want to share my location with my friends to show them where I am.
  - As a user, I want to be able to read my messages offline so I can reread conversations at any time.
  - As a user with a visual impairment, I want to use a chat app that is compatible with a screen reader so that I can engage with a chat interface.

- Design Assets
  - Can be found here.

Backend – Achievement 6 – Collaboration & Documentation

1) Description of the project

- What was my role for this project and what tasks did I face?
  - Using Angular, build the client-side for an application called myFlix based on its existing server-side code (REST API and database), with supporting documentation.

- Key Features
  - The app should display a welcome view where users will be able to either log in or register an account.
  - Once authenticated, the user should now view all movies.
  - Upon clicking on a particular movie, users will be taken to a single movie view, where additional movie details will be displayed. The single movie view will contain the following additional features:
    - A button that when clicked takes a user to the director view, where details about the director of that particular movie will be displayed.
    - A button that when clicked takes a user to the genre view, where details about that particular genre of the movie will be displayed.

- Technical Requirements

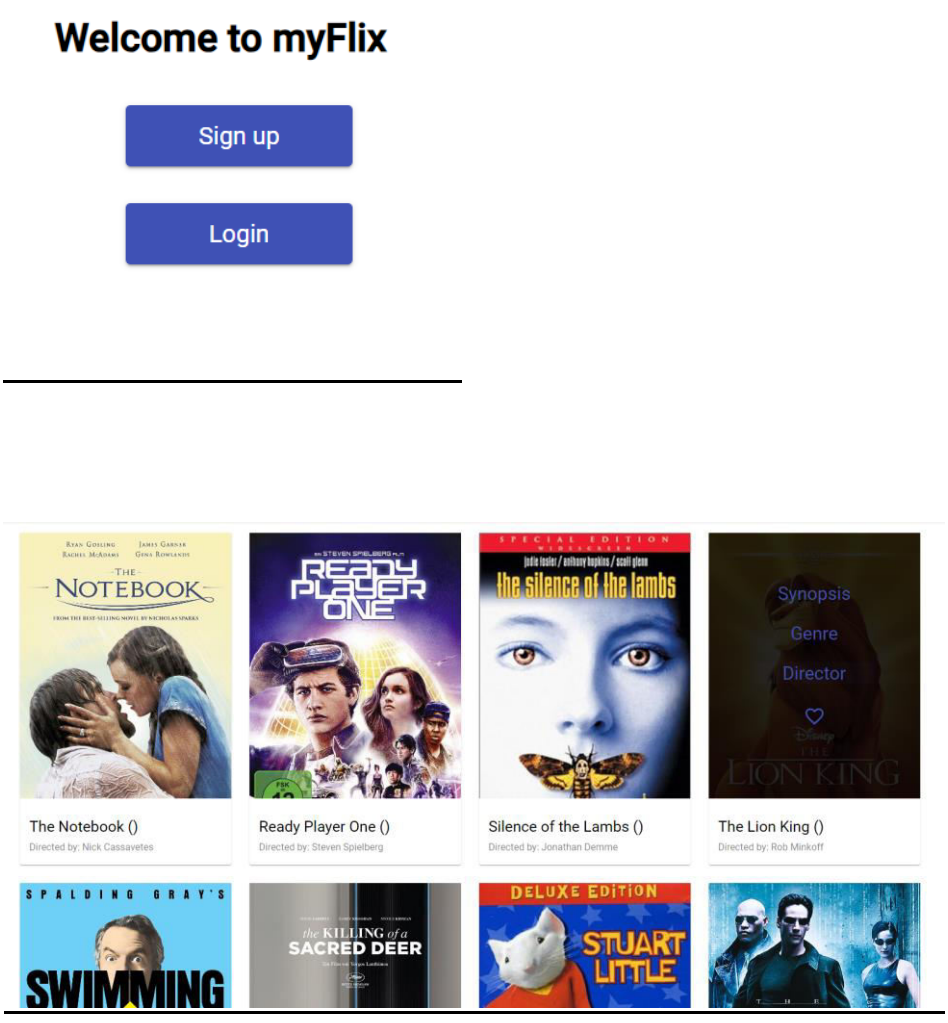
  - The application must be written in Angular (version 9 or later)
  - The application requires the latest version of Node.js and npm package
  - The application must contain user registration and login forms
  - The application must be designed using Angular Material
  - The application's codebase must contain comments using Typedoc
  - The project must contain technical documentation using JSDoc
  - The project must be hosted on GitHub Pages

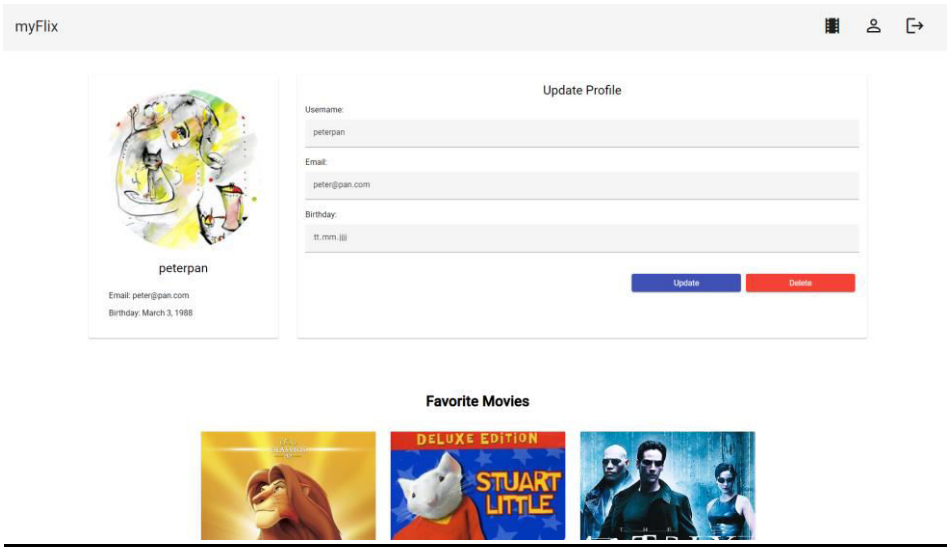
- Lessons I learned / decisions I made during this project

  - I discussed techniques for collaborating with designers,
  - discussed Agile project management techniques and common materials,
  - interpreted user stories to determine project requirements and story points,
  - identified the basic components of TypeScript,
  - created an app using Angular CLI,
  - explained the uses of Angular Material,
  - used Angular Material to implement the user registration and login forms of my app,
  - implemented a movie card component in my app,
  - defined the routing in my app,
  - explained the role of documentation in development projects and the handoff process,
  - implemented comments in my code,
  - prepared all necessary documentation for an application,
  - identified essential non-technical skills for developers,
  - demonstrated giving constructive feedback to others,
  - discussed best practices and ethical considerations for developers,
  - curated project deliverables for portfolio.

- Views
  - Welcome View
    - Allows users to either log in with a username and password or signup
  - Main View
    - Returns all movie from the API to the user
    - Ability to see director details, genre details and synopsis of each movie
    - in Navbar:
      - Ability to log out
      - Ability to navigate to Profile View
  - Profile View
    - Displays user registration details
    - Allows users to update their info (username, email, date of birth)
    - Allows existing users to deregister
    - Displays favorite movies
      - Allows users to remove a movie from their list of favorites

2) A screenshot to represent the project





3) A link to the project's GitHub repository

- https://github.com/Luisa-Inc/myFlix-Angular-client

4) A link to the live, hosted version of my app

- https://luisa-inc.github.io/myFlix-Angular-client/welcome

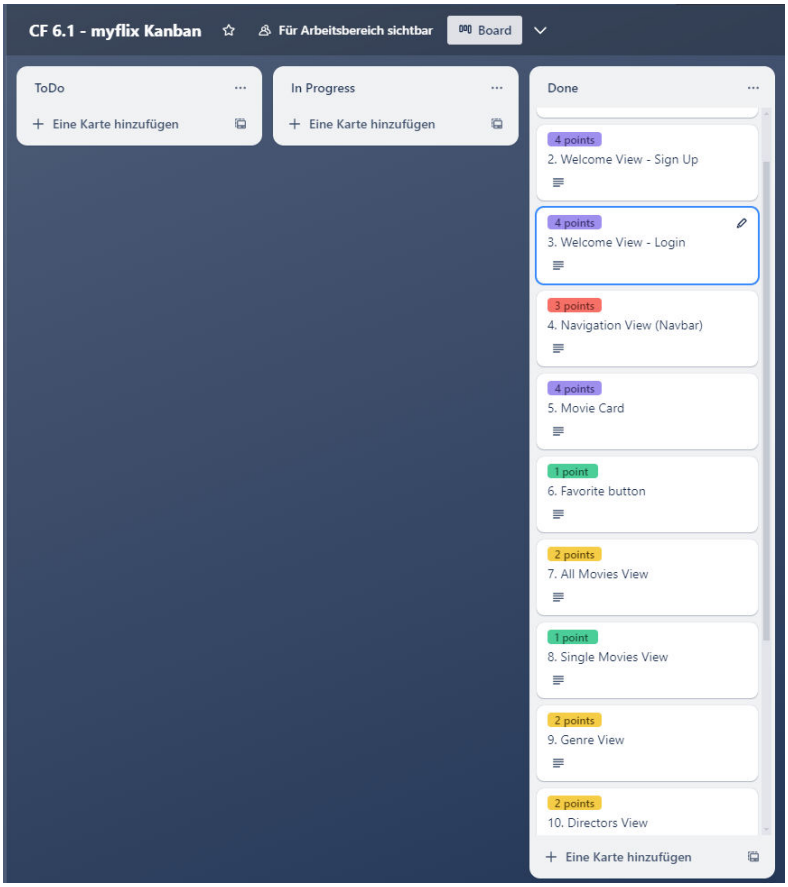
5) A list of the technologies used for each project

- Angular: Web application framework for building single-page client applications.
- Angular Material: UI component library for Angular, implementing Google's Material Design.
- TypeDoc: Generates HTML API documentation from TypeScript code.

6) Any other relevant materials I created for the project

- User Stories
  - As a user, I want to be able to receive information on movies, directors, and genres so that I can learn more about movies I've watched or am interested in.
  - As a user, I want to be able to create a profile so I can save data about my favorite movies

- Kanban-Board for showcasing my ability to work in an agile online environment
    - visualising all project steps and tasks for this Achievement with a project management tool,
    - setting a to-do-list of which the single items can be moved towards a card “in progress” and “done” to control their process,
    - trying to evaluate realistic amount of times each of those steps take and measuring them in story points which could relate to estimated hours of working time
    - project management in general was helping me to get better at estimating how long a project can take and to always be prepared for unplanned delays due to technical or human interference (this should be taken into consideration for future employers and how to communicate deadlines and realistic project development with them)



- Answers to technical job interview questions
    - 6.2
    - 6.3
    - 6.4