UNIVERSITÀ
DEGLI STUDI
DE L'AQUILA

Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica

Università degli Studi dell'Aquila

Frontend Engineering

Final Exam Project

2023 / 2024

Course: Front End Engineering

Professor: *Prof. Serafino Cicerone*

Project GitHub Repository: https://github.com/sant1dom/Frontend-Homework.git

Project developed by:

Domenico Santone	Matr. ***288664***	domenico.santone@student.univaq.it
Matteo Leoncini	Matr. ***288518***	matteo.leoncini@student.univaq.it
Mario D'Andrea	Matr. ***285363***	mario.dandrea@student.univaq.it
Giorgio Morico	Matr. ***280856***	giorgio.morico@student.univaq.it

UNIVERSITÀ
DEGLI STUDI
DE L'AQUILA

Summary

Deployed frontend & backend

You can access the deployed version of the website at the following Link

The deployed API is here.

Be sure to wait a bit since the application needs to wake up if it isn't used for a while.

Preloaded Users

admin@react.org
password: admin

pippo@react.org
password: pippo

Installation on another server

Enter inside the 'backend' folder:

```
cd backend
```

To start the backend first create a new virtual environment:

```
python -m venv <env_name>
```

Activate the virtual environment

Then install the requirements:

```
pip install -r requirements.txt
```

Finally launch the server with:

```
python movies_api.py
```

If you want to enable the real-time updates you need to launch the server with:

```
python movies_api.py --reload
```

The server will start on localhost on port 8000. Go to http://localhost:8000/docs to see the documentation.

Enter inside the 'frontend' folder:

```
cd frontend
```

To start the frontend first install the requirements:

```
npm install
```

Then launch the server with:

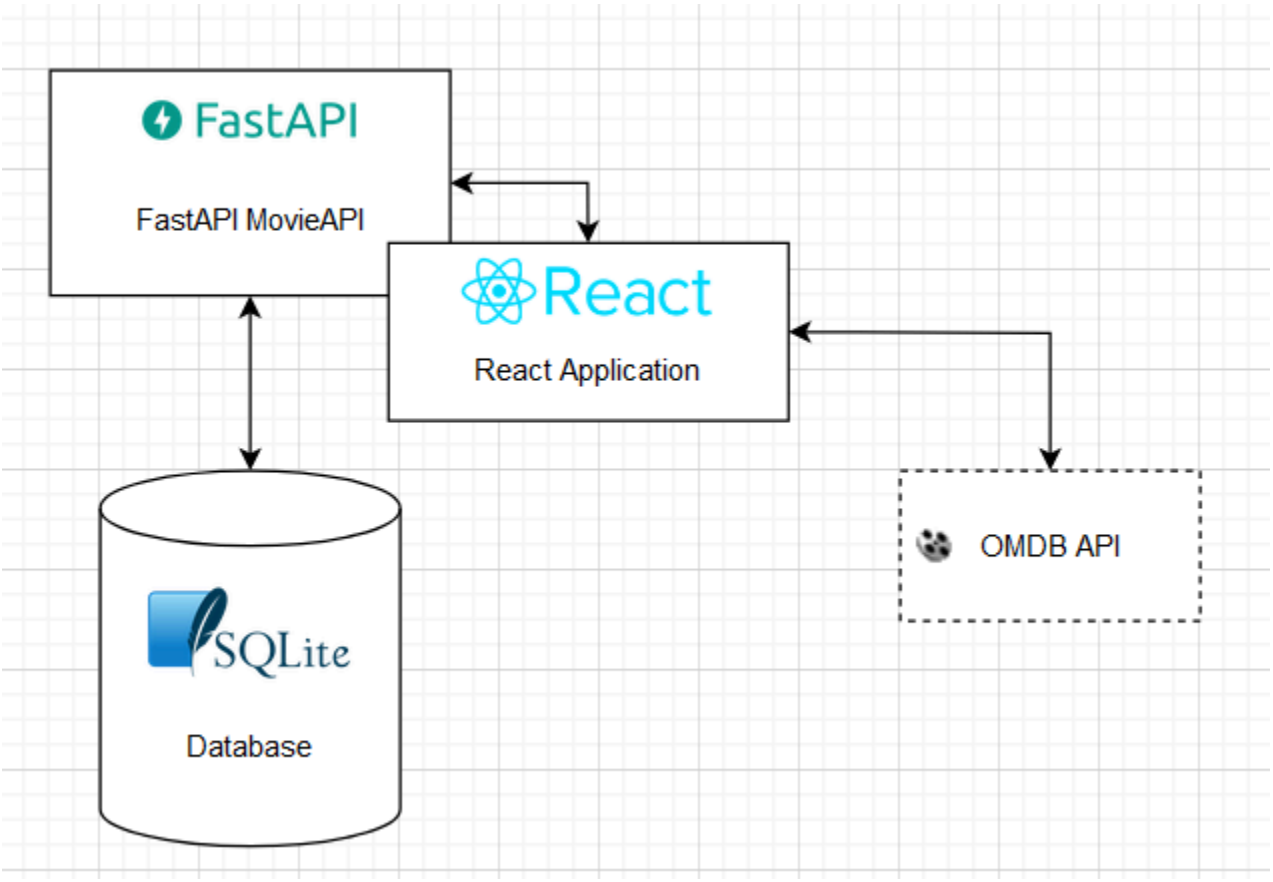
```
npm start
```

The server will start on localhost on port 3000. Go to http://localhost:3000 to see the frontend.

System Functional Requirements

- The system allows all users, even those who are not registered, to consult the homepage, carry out searches and open the movie detail page;
- The system allows registration and subsequent login to the application, thanks to which the user can access further functionalities;
- As soon as a user registers, the system creates two private lists for him called *Watchlist* and *Favourites*;
- At any point in the application where a film card is present, the user can use the '+' button to add that film to an already existing list or can create a new list that will contain that film;
- The system allows the user to search for films and lists via the search bar in the navbar;
- The system allows the user to perform advanced searches based on specific criteria on the "Advanced Search" page;
- The system allows the user to consult their lists, both public and private, via the "MyLists" page. From this page the user can also delete public lists or change their title, and create new lists using the '+' button;
- By opening a specific list, the system shows to the user various information:
    - List title;
    - Like button with likes counter;
    - Films on the list;
    - A set of filters to filter films by genre, language and release year;
    - Comments section: once a comment has been published, it can be modified or deleted using the appropriate buttons;
    - Editor to add new comments;
- As regards the private *Watchlist* and *Favourites* lists, the functions for likes and comments are not present because they are lists not visible to other users;
- On the movie cards present in a user's list there is, in addition to the '+' button, a button that allows to delete the movie from that list;
- The system allows the user to consult the best public lists via the "BestLists" page.
  In this page the list cards show, in addition to the name, also the author and the counters for likes and comments;
- The system allows the user to change profile picture and password via the “Profile” page;
- Admin users can insert new films, modify/delete those already present, delete lists and comments. Except for the page to insert new films, in all the others there is a search bar which facilitates the search for the desired information;

System Architecture



Architecture Diagram

React App Architecture

Color scheme in the following diagrams:

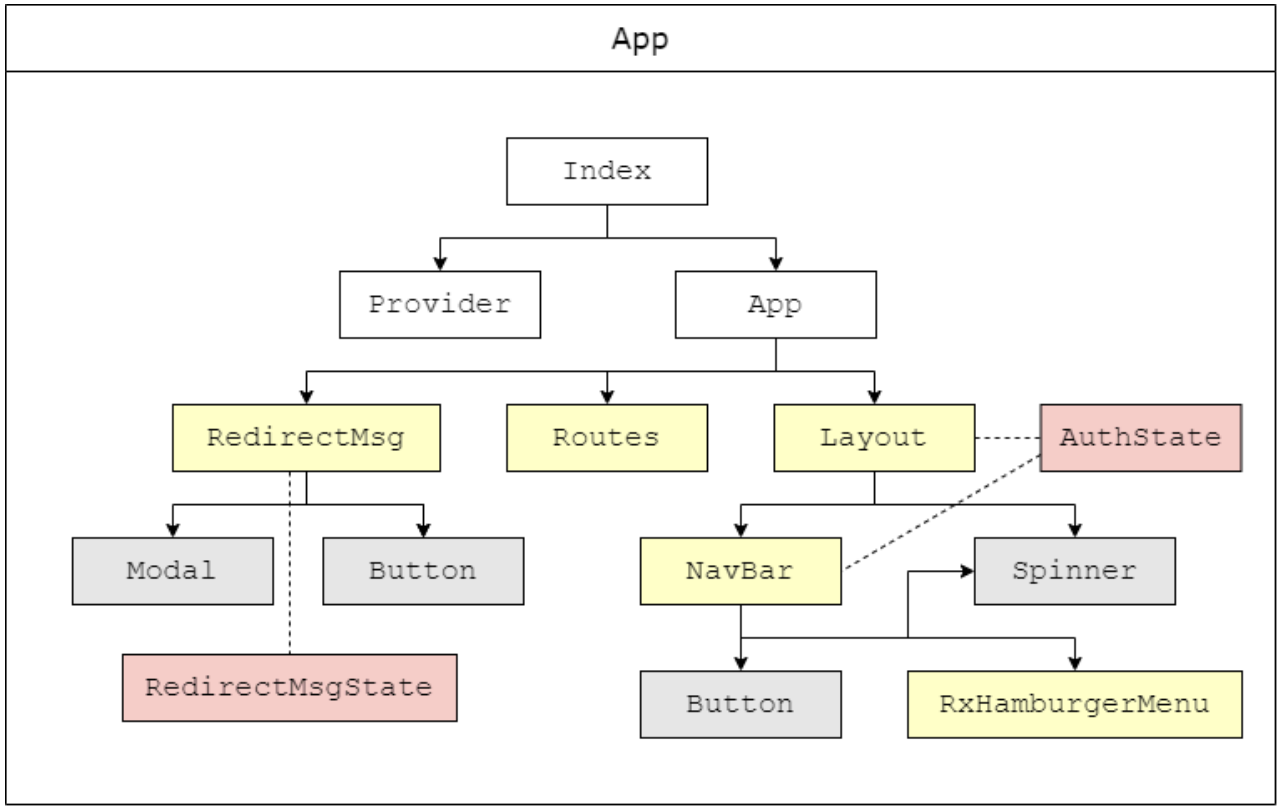
White background: base app

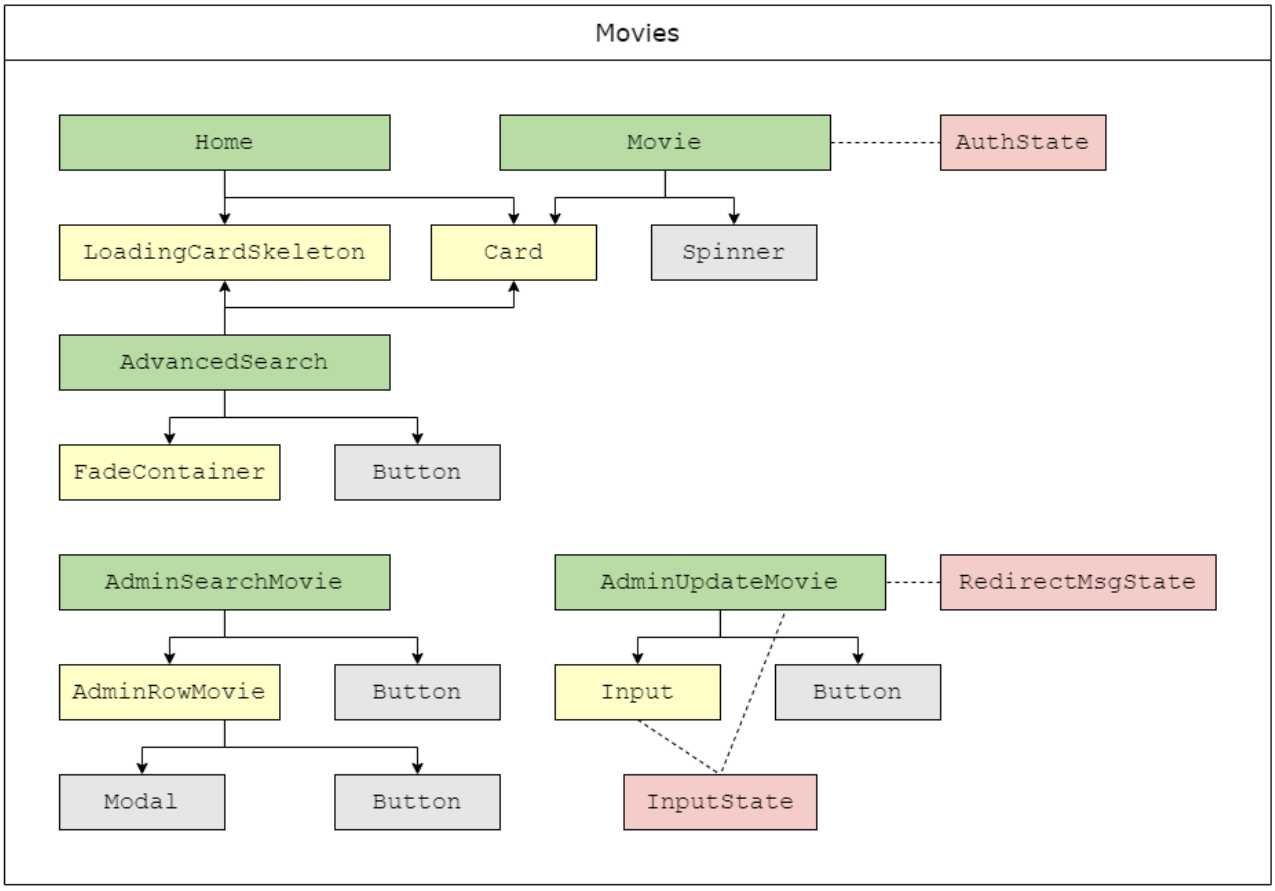
Green background: pages

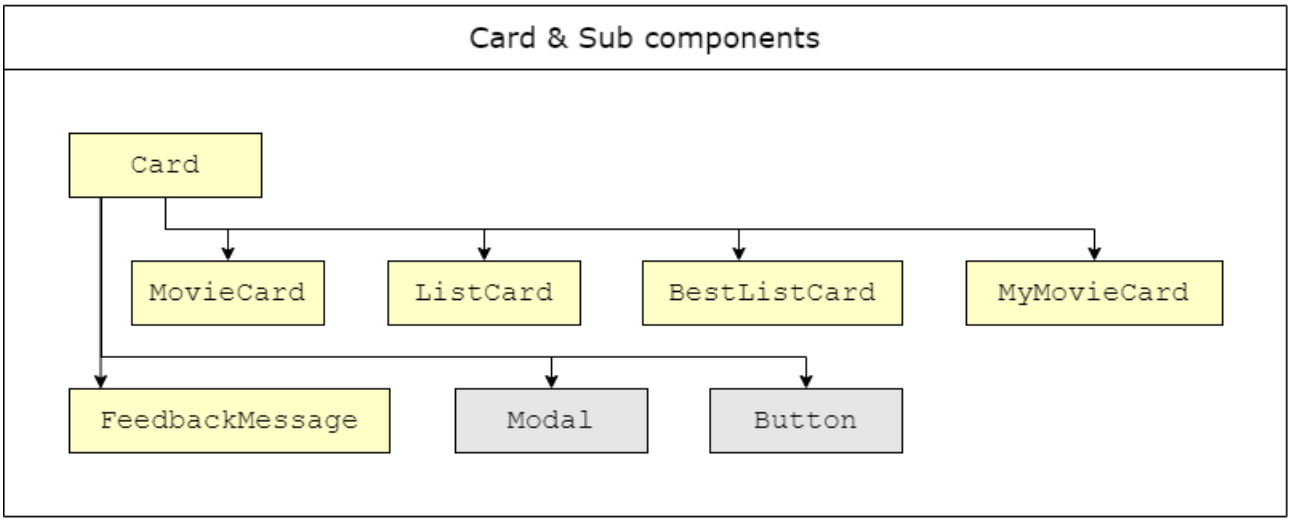
Yellow background: main components

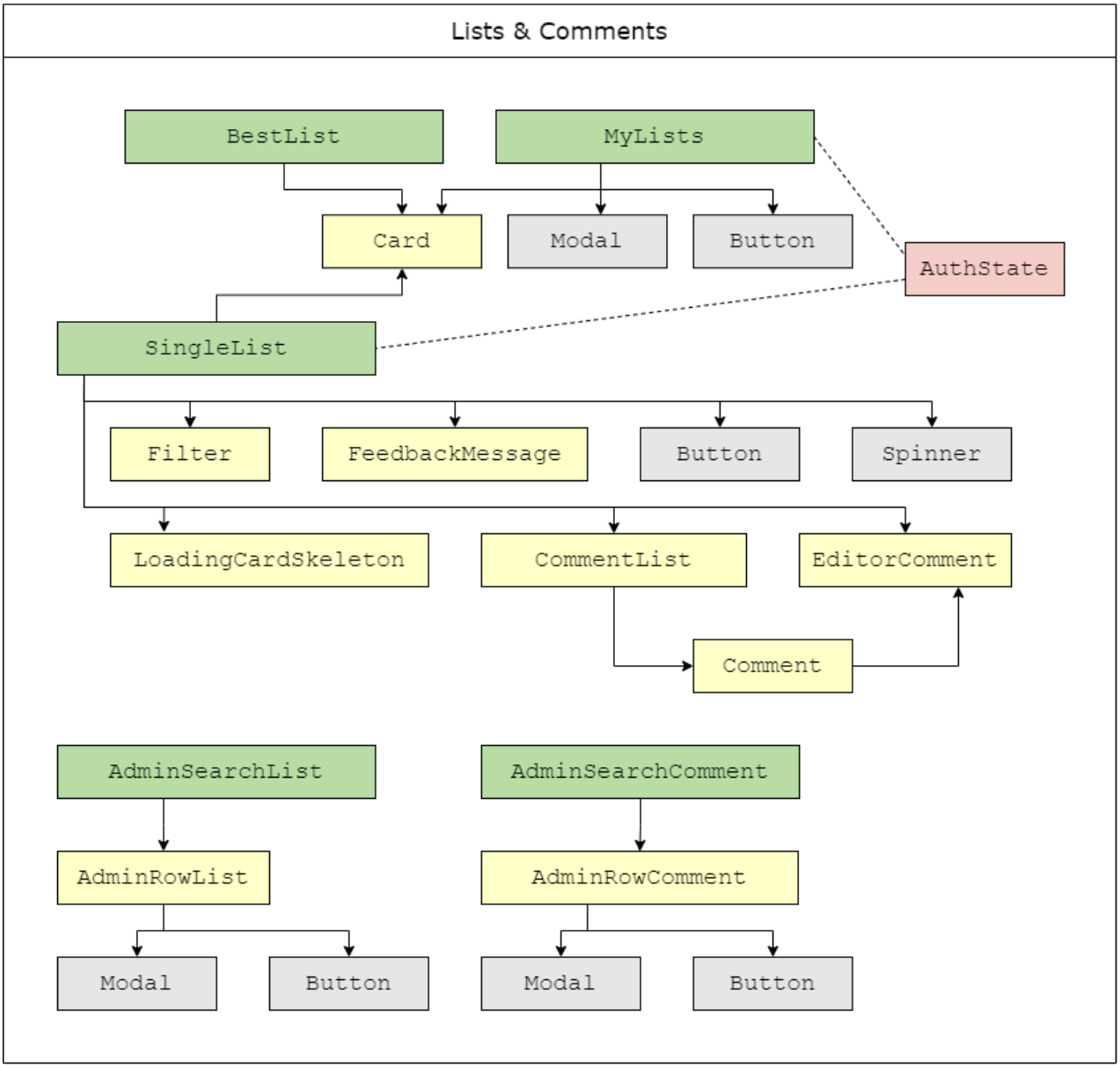
Gray background: three secondary components (Modal, Button, Spinner)

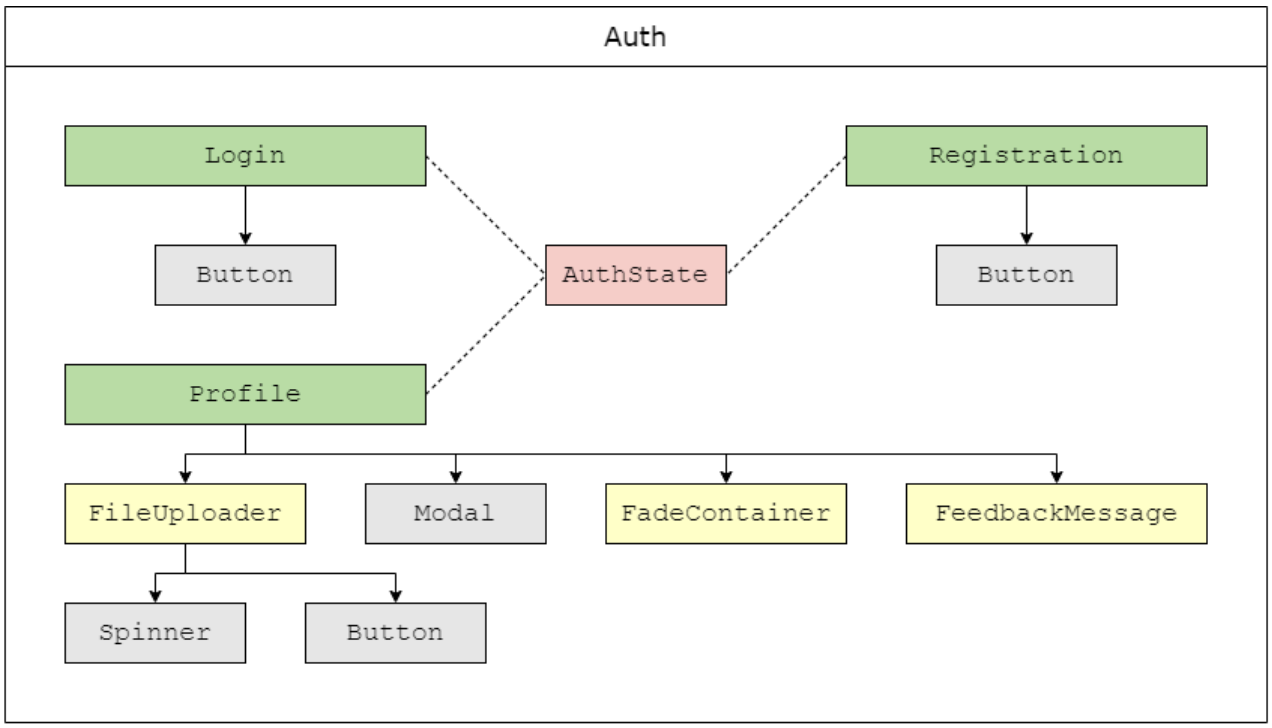
Red background: states

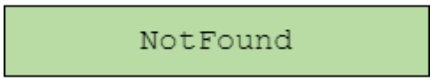



Movies
Home
Movie
AuthState
LoadingCardSkeleton
Card
Spinner
AdvancedSearch
FadeContainer
Button
AdminSearchMovie
AdminUpdateMovie
RedirectMsgState
AdminRowMovie
Button
Input
Button
Modal
Button
InputState


Card & Sub components
Card
MovieCard
ListCard
BestListCard
MyMovieCard
FeedbackMessage
Modal
Button


Lists & Comments
BestList
MyLists
Card
Modal
Button
AuthState
SingleList
Filter
FeedbackMessage
Button
Spinner
LoadingCardSkeleton
CommentList
EditorComment
Comment
AdminSearchList
AdminSearchComment
AdminRowList
AdminRowComment
Modal
Button
Modal
Button

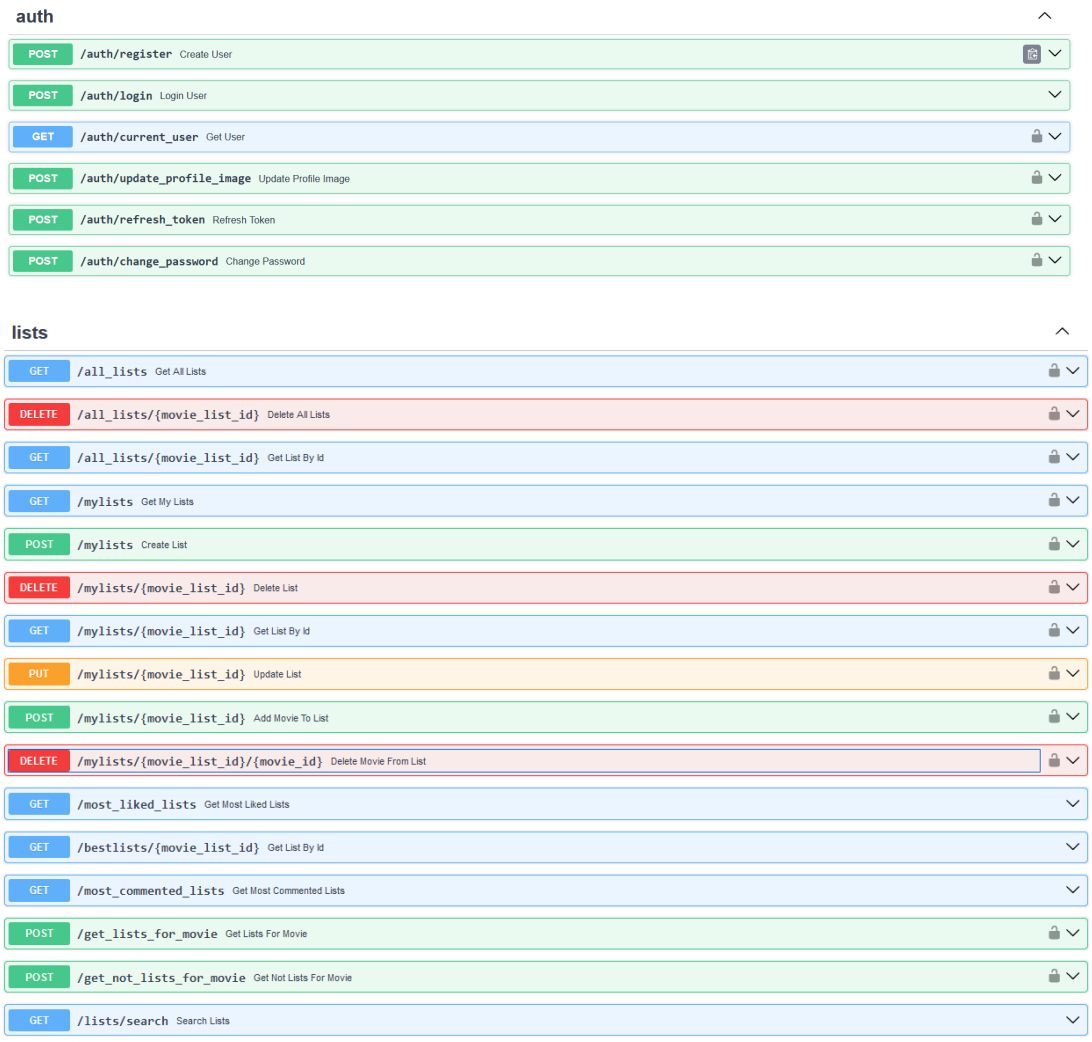

Auth
Login
Registration
Button
AuthState
Button
Profile
FileUploader
Modal
FadeContainer
FeedbackMessage
Spinner
Button


NotFound

Backend Logic

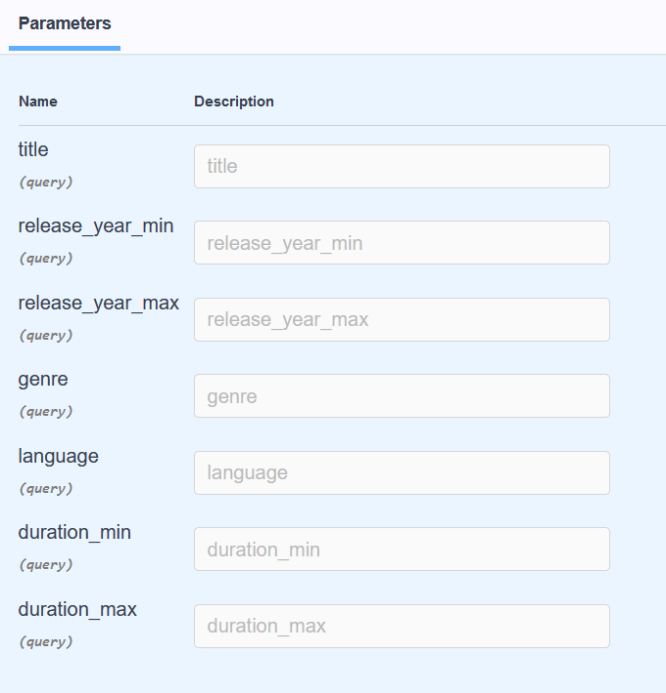
The backend logic was developed using **Python** with the **FastAPI** framework, all the data is saved in a **SQLite** Database. It provides all the interfaces for creating, reading, updating and deleting movies, comments, likes, lists and so on. It also provides several search functionalities that provides some filters for the various entities.
The API is responsible for the User Authentication using OAuth2 and JWT tokens.

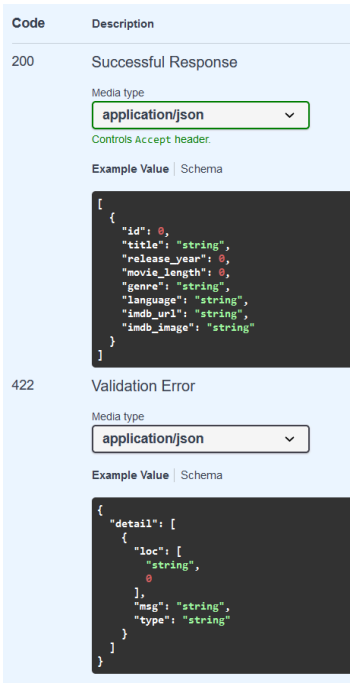
The API is documented using OpenAPI documentation style and you can find it at the url **localhost:8000/docs**.




comments
GET /all_comments Get All Comments
DELETE /all_comments/{comment_id} Delete All Comments
POST /comment/{movie_list_id} Comment Movie List
PUT /comment/{comment_id} Update Comment
DELETE /comment/{comment_id} Delete Comment
GET /comments/{movie_list_id} Get Comments
movies
GET /movies Get Movies
POST /movies Create Movie
PUT /movies/{movie_id} Update Movie
DELETE /movies/{movie_id} Delete Movie
GET /movies/{movie_id} Get Movie By Id
GET /movies/search Search Movies
genres
GET /genres Get Genres
languages
GET /languages Get Languages
likes
POST /like/{movie_list_id} Like Movie List
DELETE /like/{movie_list_id} Unlike Movie List
users
GET /users/{user_id} Get User Details By Id
default
GET / Root

Example (this one is for the movie search):





How we deployed

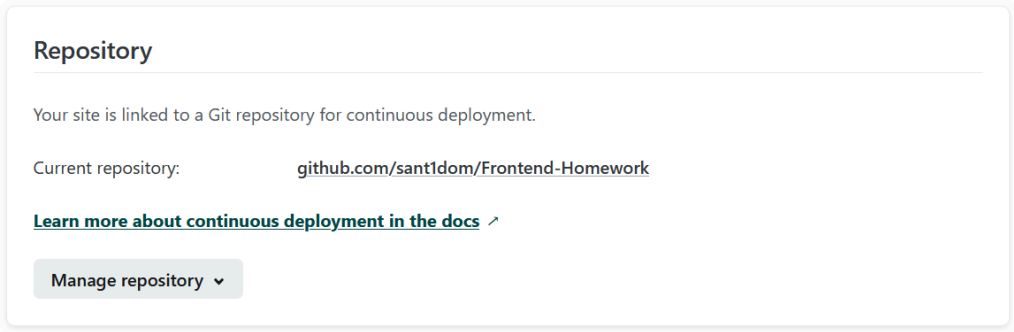
The deployment of the platform was in 2 different free hosting sites for frontend and backend. The services we have chosen are:

- Netlify
- Render.com

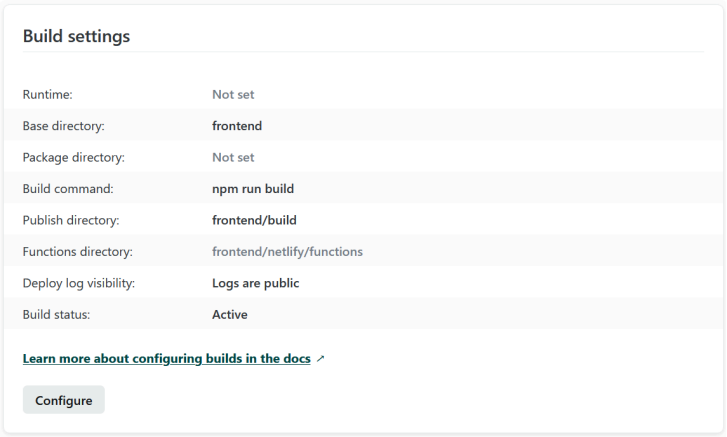
Netlify

The deploy the frontend on the Netlify platform is particularly easy to setup, following the tutorial here. The followings are the settings we used to deploy our app:

1. The linked repository allows to deploy on every commit



2. Using the command `npm run build` is possible to build the application and the following settings are used to publish the build directory

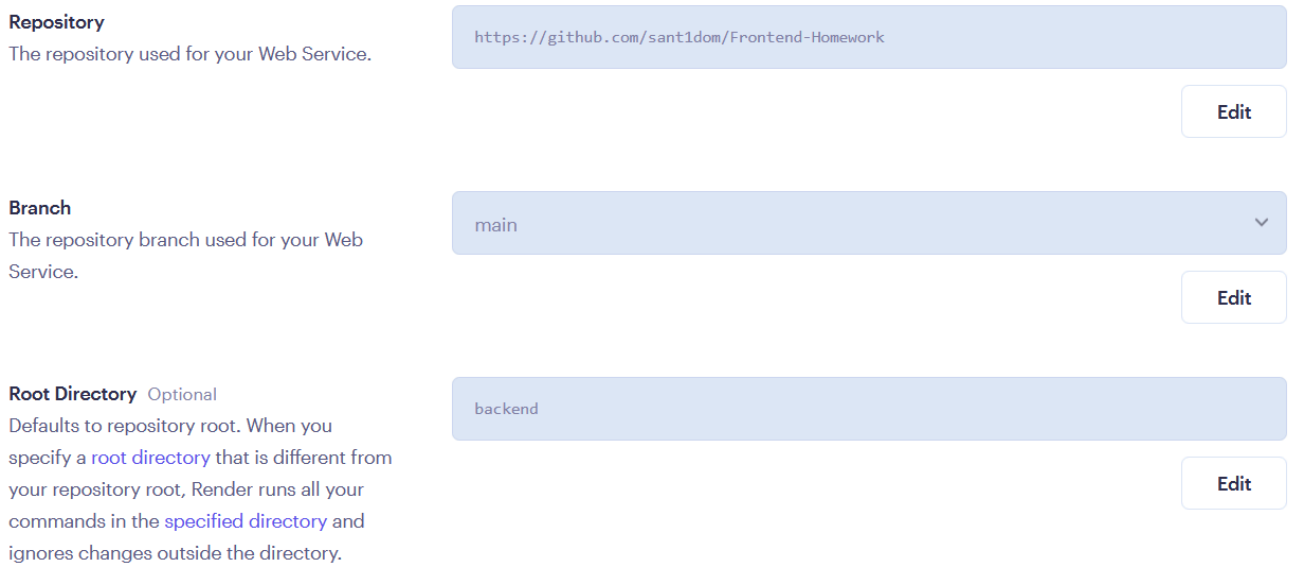


3. Is possible to customize also the domain of the deployed website

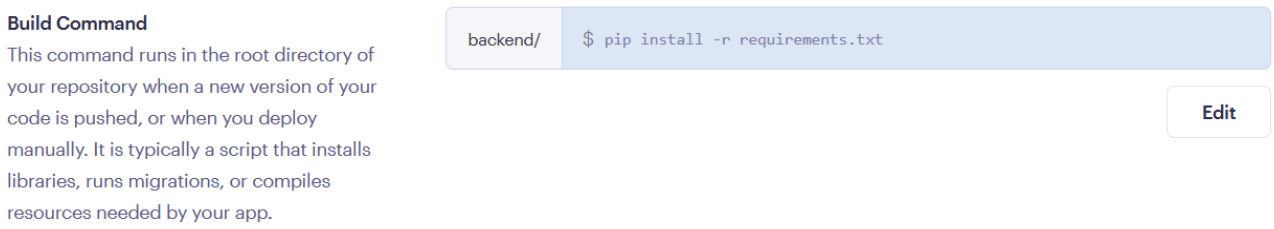
Render.com

For **render.com** the situation is very different and each programming language and framework may have a different solution. Our backend is built with Python and FastAPI so we opted to use the standard python build that Render offers.

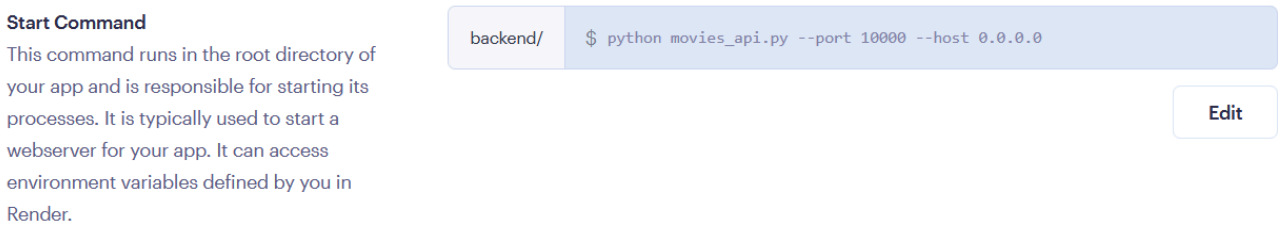
1. Is possible to setup the repository and the branch in order to have different branches deployed in different instances, the root directory is the one where all the commands are executed.



2. In order to build the Python app we wrote all the requirements in a file called requirements.txt (this is the standard practice using pip, with more advanced dependency management tools like poetry the build command may be different) and then we execute the following command:



3. Last we launched our main file being sure that it runs on host 0.0.0.0 and port 10000 (the parameters are probably customizable but we haven't found a way to handle this).



In order to make this as parametric as possible the main function of our api looks like this:

```
if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--reload", action="store_true", help="Reload the server on code changes")
    parser.add_argument("--host", type=str, default="localhost", help="Host to run the server on")
    parser.add_argument("--port", type=int, default=8000, help="Port to run the server on")
    parser.add_argument("--workers", type=int, default=1, help="Number of workers to run")
    parser.add_argument("--log-level", type=str, default="info", help="Log level")
    arguments = parser.parse_args()

    uvicorn.run(
        "movies_api:app",
        host=arguments.host,
        port=arguments.port,
        reload=arguments.reload,
        workers=arguments.workers,
        log_level=arguments.log_level,
    )
```

In this way we can specify whatever we want for the application and this facilitates a lot also the local development of the application.

Technologies Used