Micro campus project

Introduction

Premise

A mini campus where users can make reservations for rooms and resources.

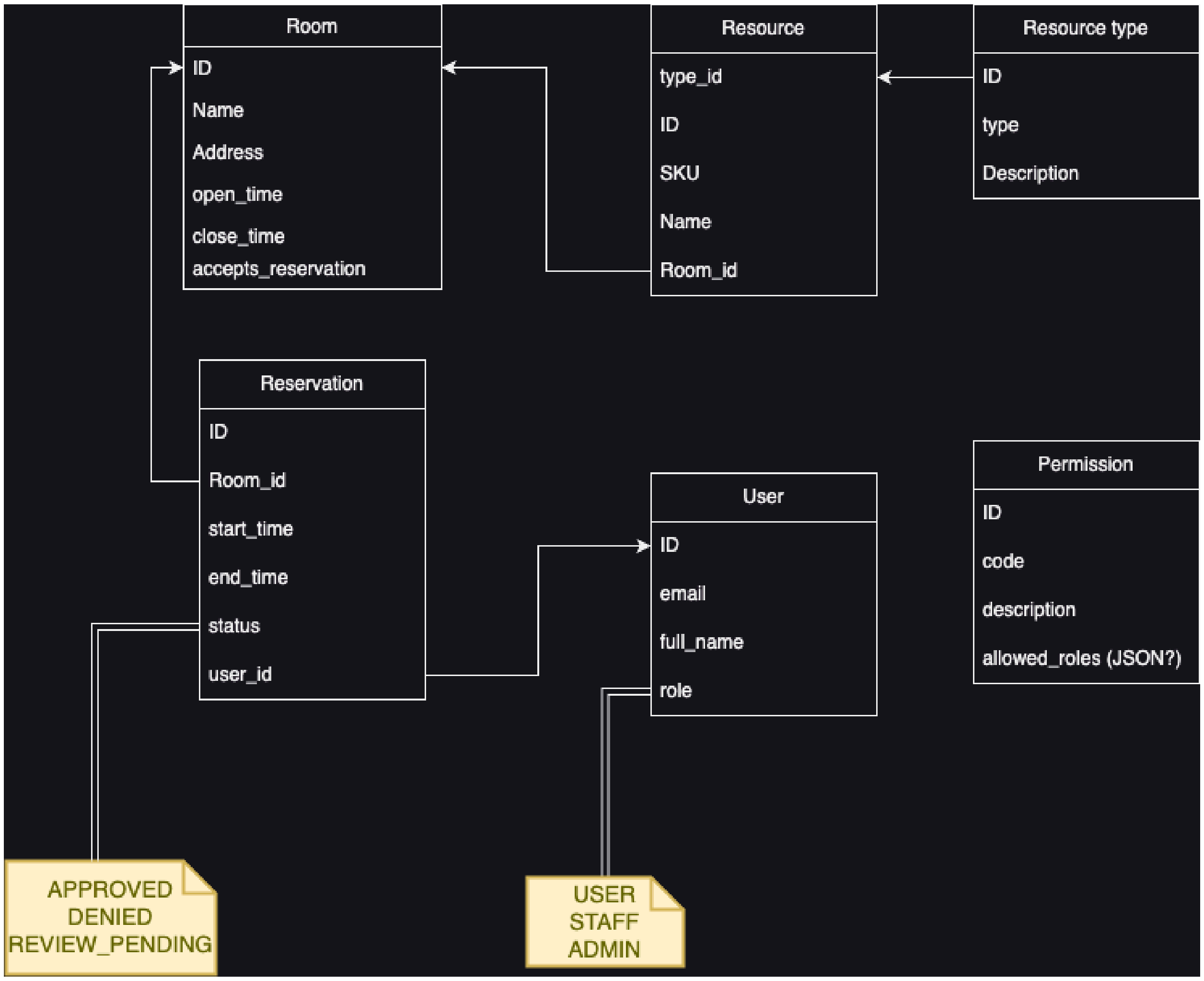
Design

How it should work

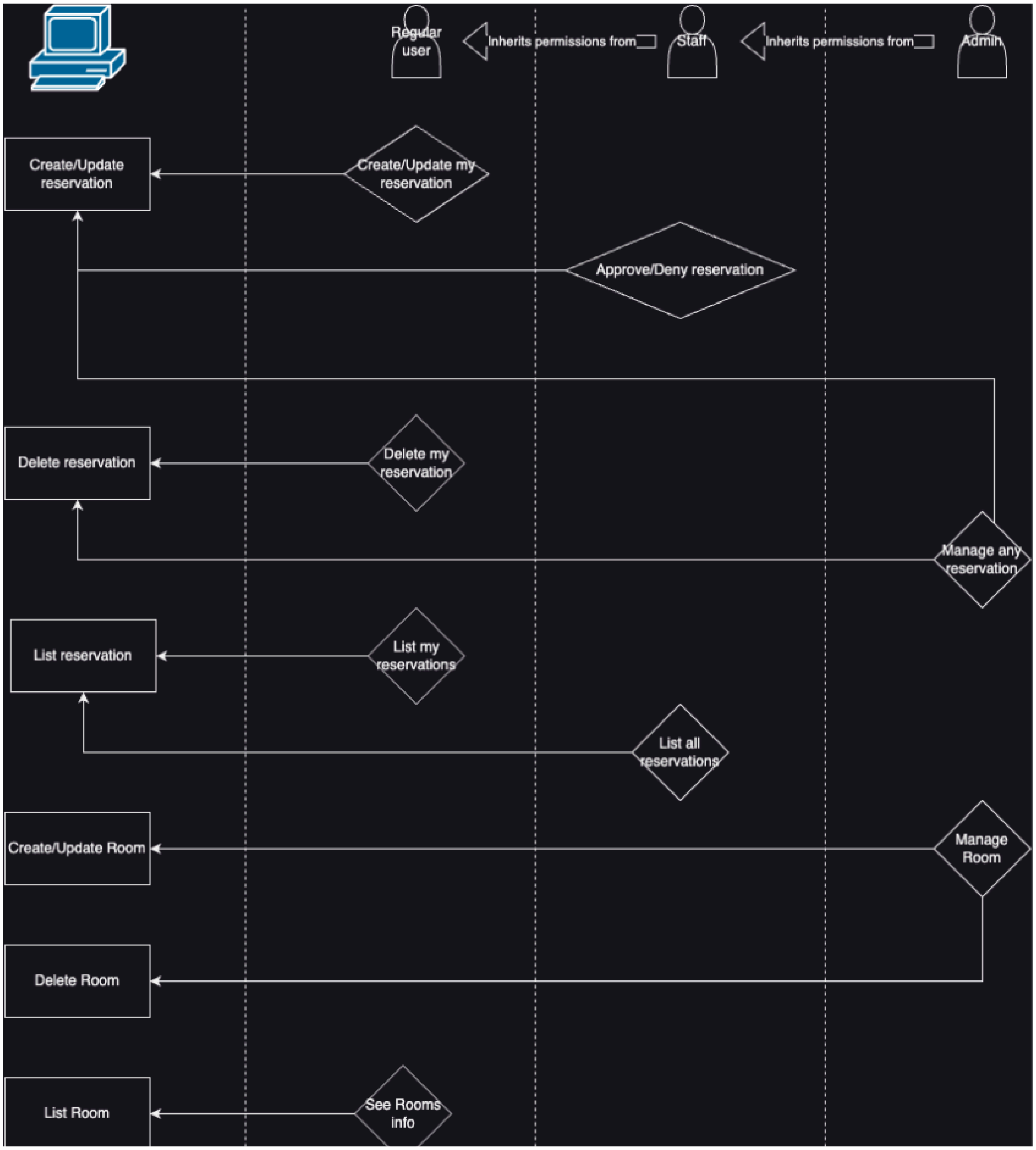
The idea is that the reservation service will only manage everything related to reservations, including rooms, resources, etc.

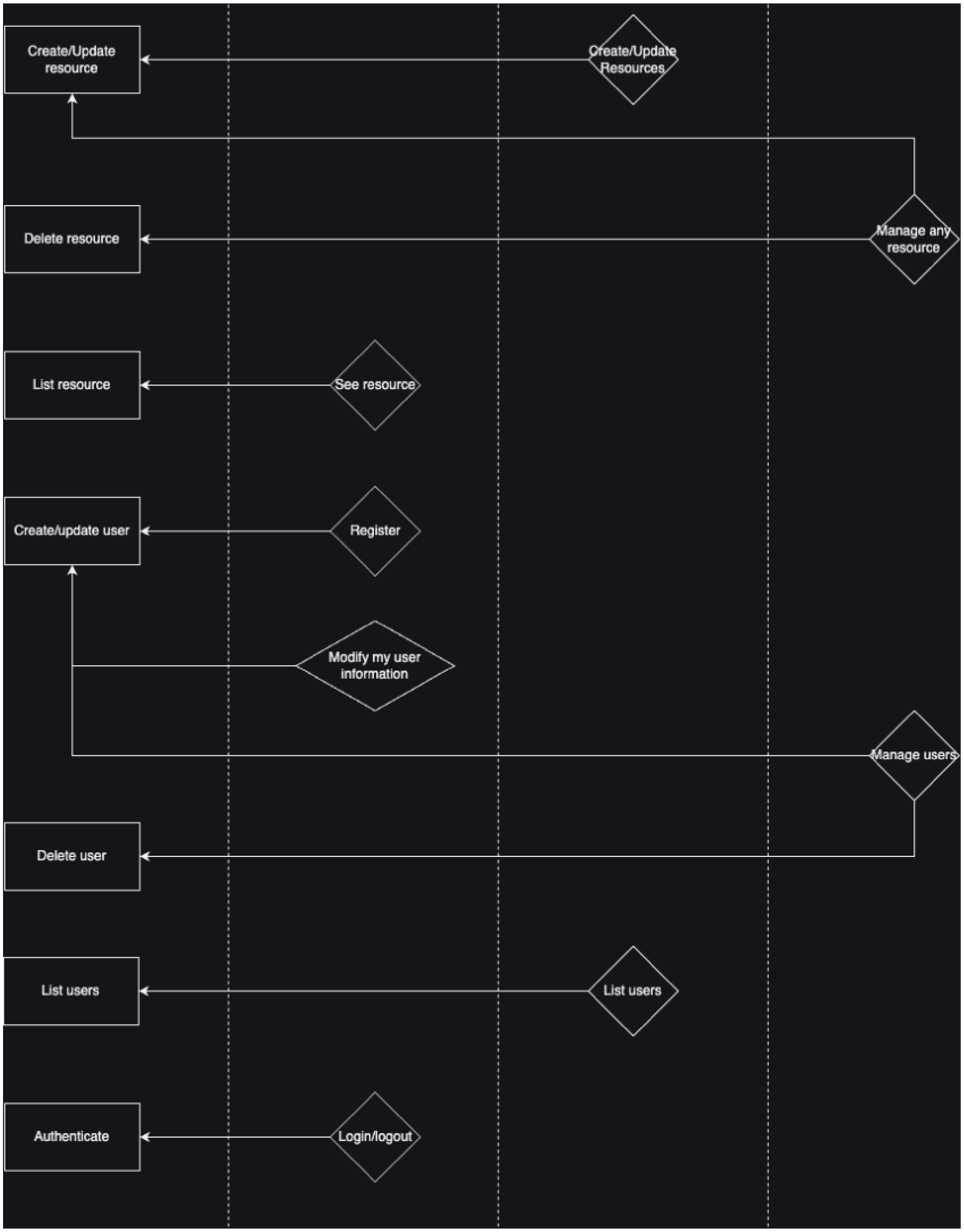
The source of truth for the users and the one that will grant access tokens and validate them will be the user service. For anything that requires authorization, reservation service will contact user service through an API request. Some of these requests will include the token of the service itself, credentials that are only for m2m communication. User service will only have one table which will be users.

Reservation service models


Room
ID
Name
Address
open_time
close_time
accepts_reservation
Resource
type_id
ID
SKU
Name
Room_id
Resource type
ID
type
Description
Reservation
ID
Room_id
start_time
end_time
status
user_id
User
ID
email
full_name
role
Permission
ID
code
description
allowed_roles (JSON?)
APPROVED
DENIED
REVIEW_PENDING
USER
STAFF
ADMIN

Reservation service flow


Regular user
Inherits permissions from
Staff
Inherits permissions from
Admin
Create/Update reservation
Create/Update my reservation
Approve/Deny reservation
Delete reservation
Delete my reservation
Manage any reservation
List reservation
List my reservations
List all reservations
Create/Update Room
Manage Room
Delete Room
List Room
See Rooms info


Create/Update resource
Create/Update Resources
Delete resource
Manage any resource
List resource
See resource
Create/update user
Register
Modify my user information
Manage users
Delete user
List users
List users
Authenticate
Login/logout

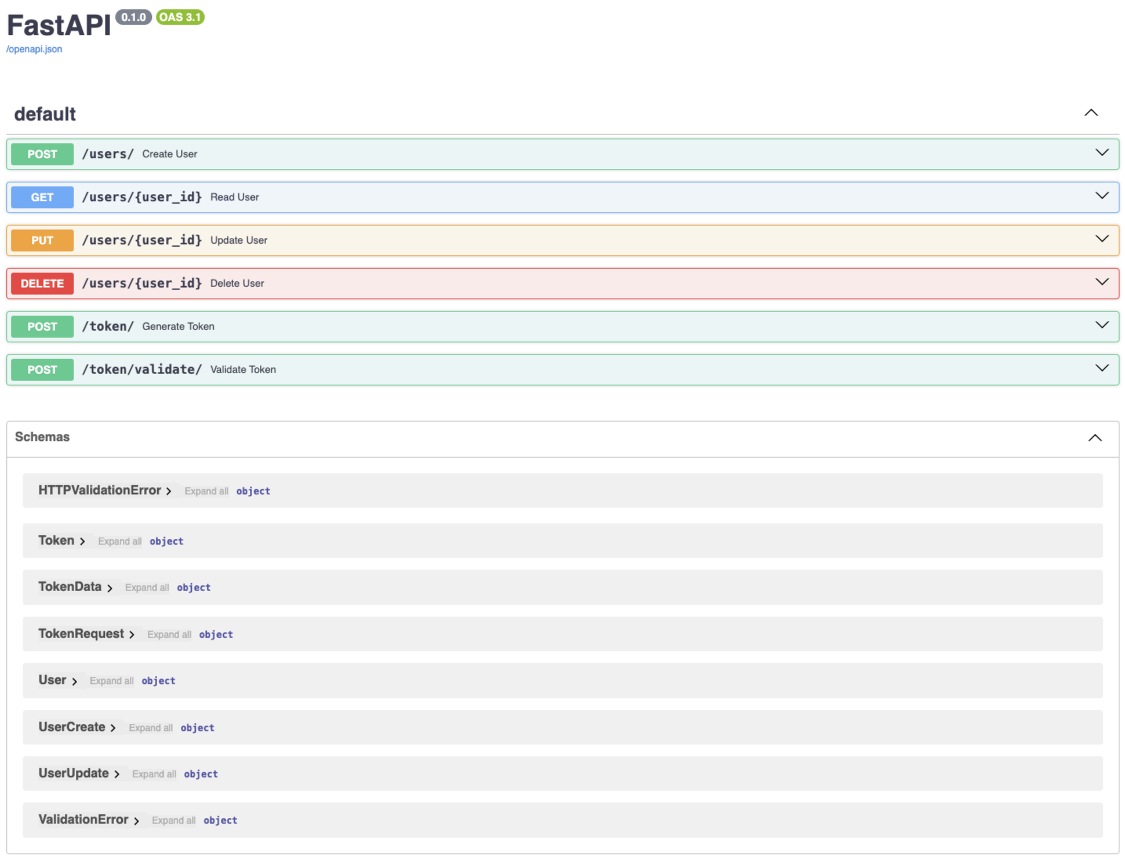
Set up

1. Clone the repository
2. Navigate to the repository root
3. Run `docker-compose build` then `docker-compose up`
4. The services should start running and all migrations will be applied

How to use

User service

The user service API documentation can be accessed at: http://localhost:8180/ or http://user-service:8000/



FastAPI 0.1.0 OAS 3.1
/openapi.json

default

POST /users/ Create User
GET /users/{user_id} Read User
PUT /users/{user_id} Update User
DELETE /users/{user_id} Delete User
POST /token/ Generate Token
POST /token/validate/ Validate Token

Schemas

HTTPValidationError > Expand all object
Token > Expand all object
TokenData > Expand all object
TokenRequest > Expand all object
User > Expand all object
UserCreate > Expand all object
UserUpdate > Expand all object
ValidationError > Expand all object

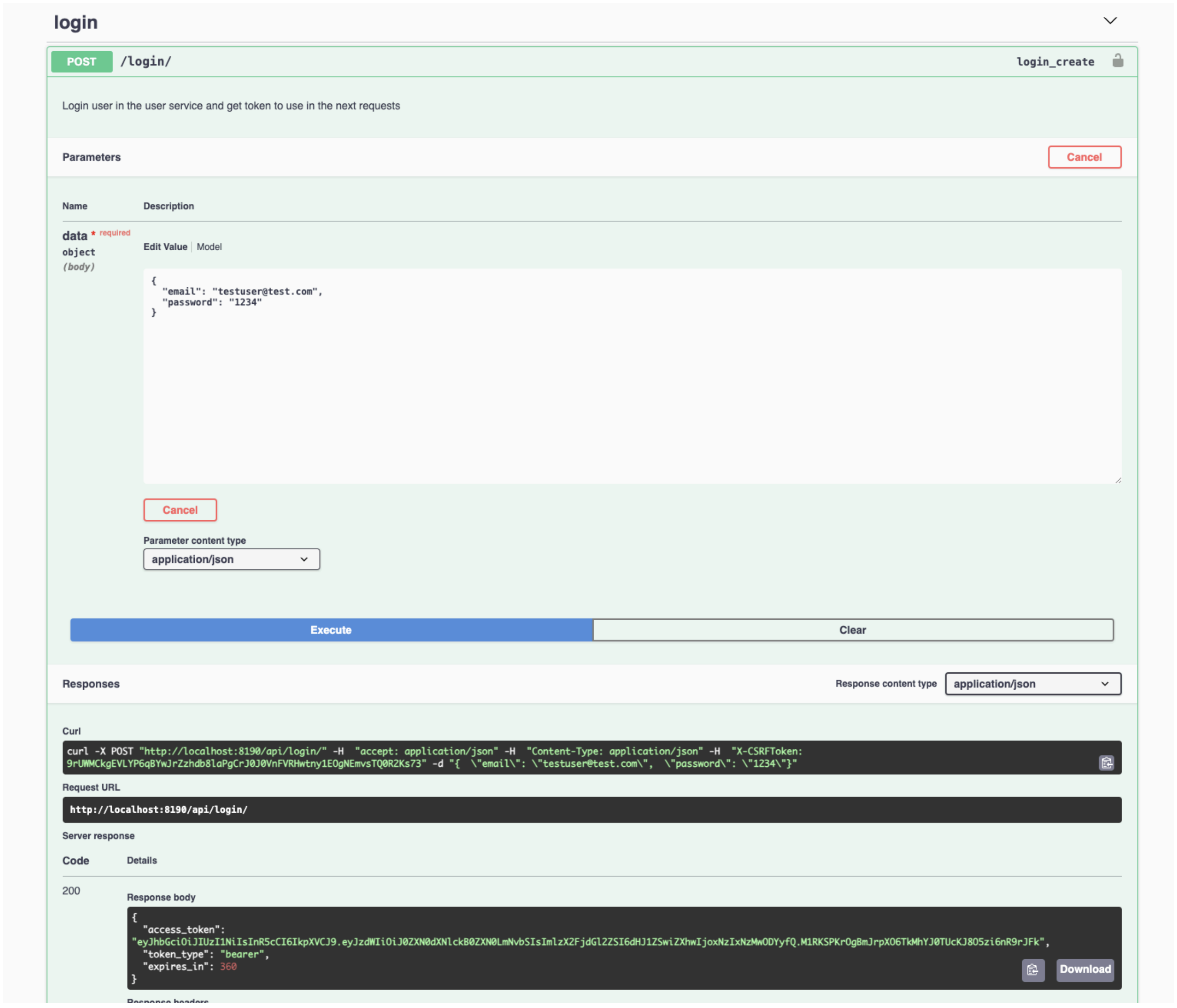
Reservation service

Swagger

Docs for the api can be accessed here http://localhost:8190/swagger/ or http://reservation-web:8000/swagger/

All the `GET` endpoints can be accessed without authorization.The other endpoints need an authorization token in the request.

Use the login to get a token that you can then use in the authorize option of the swagger. Make sure to include `Bearer ` before the token



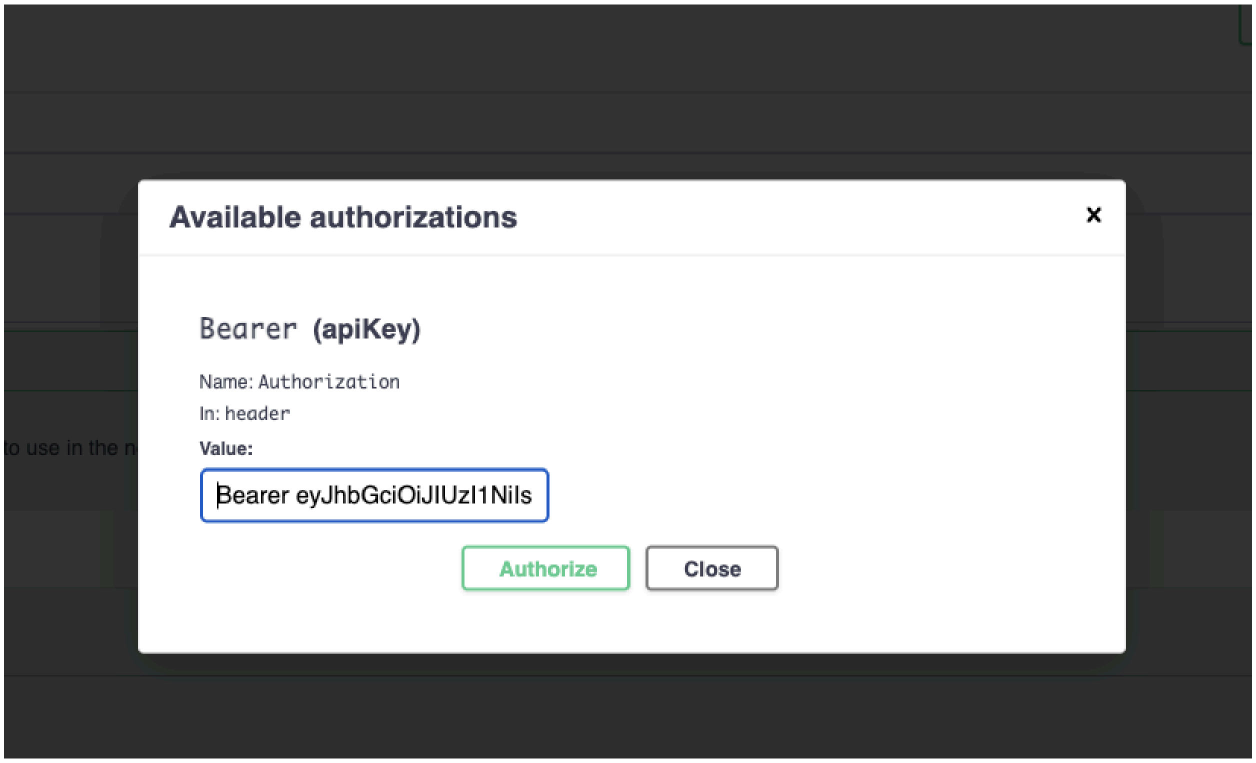
Login user in the user service and get token to use in the next requests

```
{
  "email": "testuser@test.com",
  "password": "1234"
}
```

```
curl -X POST "http://localhost:8190/api/login/" -H  "accept: application/json" -H  "Content-Type: application/json" -H  "X-CSRFToken: 9rUWMCkgEVLYP6qBYwJrZzhdb8laPgCrJ0J0VnFVRHwtny1EOgNEmvsTQ0R2Ks73" -d "{  \"email\": \"testuser@test.com\",  \"password\": \"1234\"}"
```

```
http://localhost:8190/api/login/
```

```
{
  "access_token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJ0ZXN0dXNlckB0ZXN0LmNvbSIsImlzX2FjdGl2ZSI6dHJ1ZSwiZXhwIjoxNzIxNzMwODYyfQ.M1RKSPKrOgBmJrpXO6TkMhYJ0TUcKJ8O5zi6nR9rJFk",
  "token_type": "bearer",
  "expires_in": 360
}
```


Available authorizations

Bearer (apiKey)

Name: Authorization

In: header

Value:

Bearer eyJhbGciOiJIUzI1NiIs

After clicking authorize, all the other requests you make in this page will include the token.

Django admin

Django admin can be accessed here: http://localhost:8190/admin or http://reservation-web:8000/admin

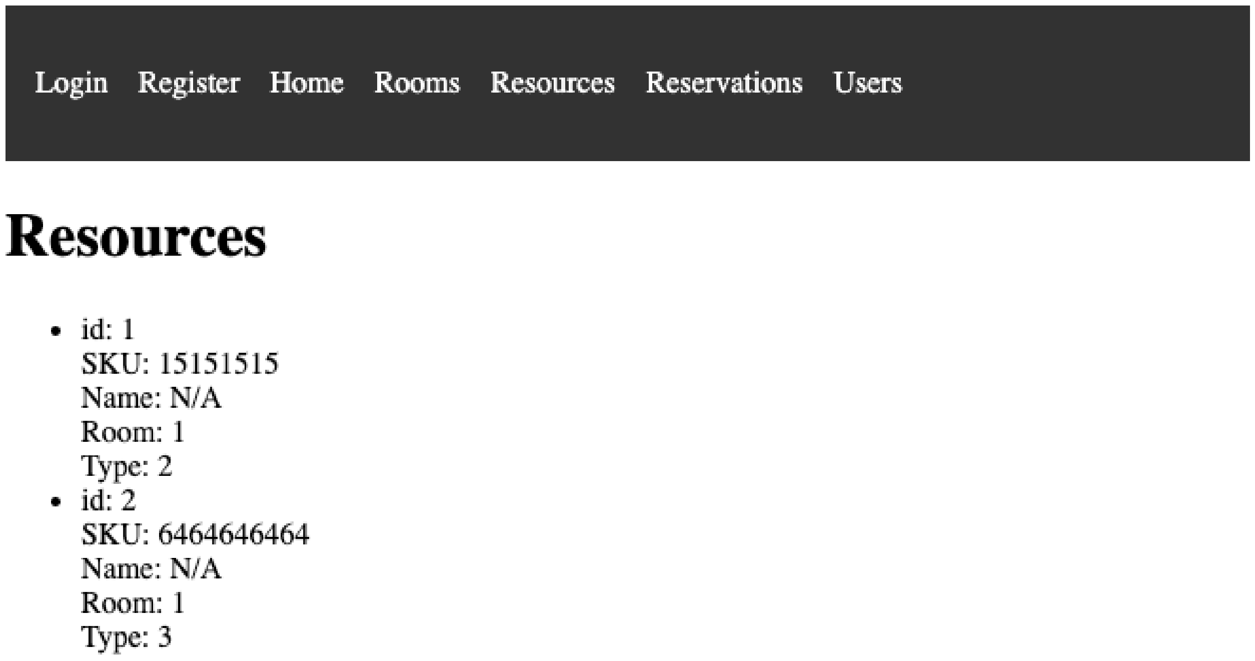
You can use the django admin to easily add data in the system. It is recommended however to not add new users through django admin but rather through the endpoint in swagger

A test superuser is added with the migrations that can be used for any operation in the platform. Make sure to use these credentials:
Email: `test-user-1@test.com`
Password: 1234

React app

A react application can be accessed here: http://localhost:8200/ or http://frontend:3000/

In this very simple web application you can access a list of certain models that are available in the reservations app. Authentication is not needed for these screens.



Resources

- id: 1
  SKU: 15151515
  Name: N/A
  Room: 1
  Type: 2
- id: 2
  SKU: 6464646464
  Name: N/A
  Room: 1
  Type: 3

Set backs

A series of issues were present during the development:

1. Time constraints
   a. The project development began a few days before the final date, mainly due to several personal issues that prevented me from starting earlier.
   b. Limited work time for development, meaning I had little time during the date to work on the project.
2. Undefined scope at the beginning of the project
   a. The scope of the project was too wide and complex in the beginning, had to simplify it a lot as I went on.
3. Usage of technologies that I have not used for a long time

    a. The usage of Django was a major setback. Django gave a lot of issues related to its internal functioning that took too much time to resolve.
4. No time to dedicate to bringing this to prod.
    a. I didn't even bother with fixing the initial terraform files that I autogenerated. I didn't have enough time to go over it.
    b. One of the project requirements asked for AWS, which I do not know how to use well. GCP is what I'm most comfortable with.

For future projects:

1. On a time constraint, stick to well known technologies you're familiarized with.
2. Less is more, define the scope better in the beginning.
3. Try to start the project from the beginning of the course.
4. Stick to what you know, deploy to the provider you know how to work with best.