BOOK STORE

This document contains the I+D technical details of the described solution that integrates Web API built in .NET Angular, MongoDB Cluster fully containerised Docker

Contents

Doc. Version	Date	Author	Description
1.0	01-01-2026	https://github.com/kraftcoding	First version of the release

Requirements

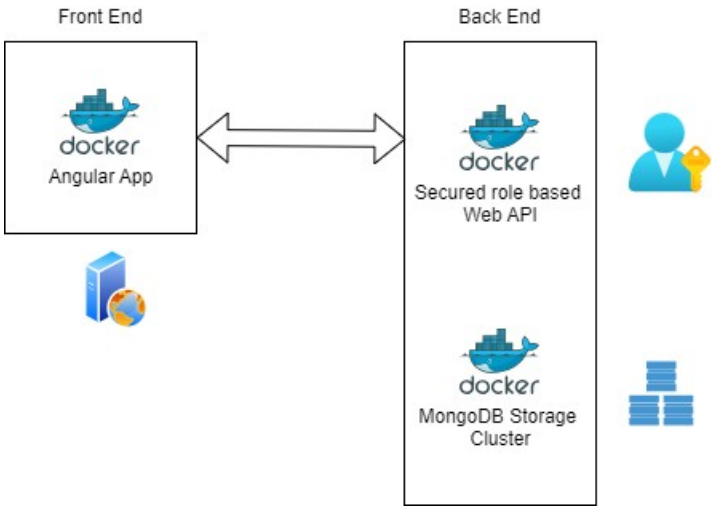
The next table shows the requirements for the solution described.

Requirement	Description
RQ-001 | Web API	Back-end web API and al resources to deploy to docker
RQ-002 | MongoDB Cluster	Mongo based NoSQL database cluster
RQ-003 | Angular App	Angular based frontend
RQ-004 | Docker	Containerisation system for back and front-end

Overview

Book Store v1.x solution built with Angular 18, MongoDB, Docker and .Net 8; enables to manage books records with the possibility to manage them by the Web UI and user roles.

Implements token-based authentication for registered users and policy-based authorisation in the endpoints.



The anonymous user can:

- Login
- Search books by title
- See book list with a few fields

The authenticated user in addition to the anonymous can:

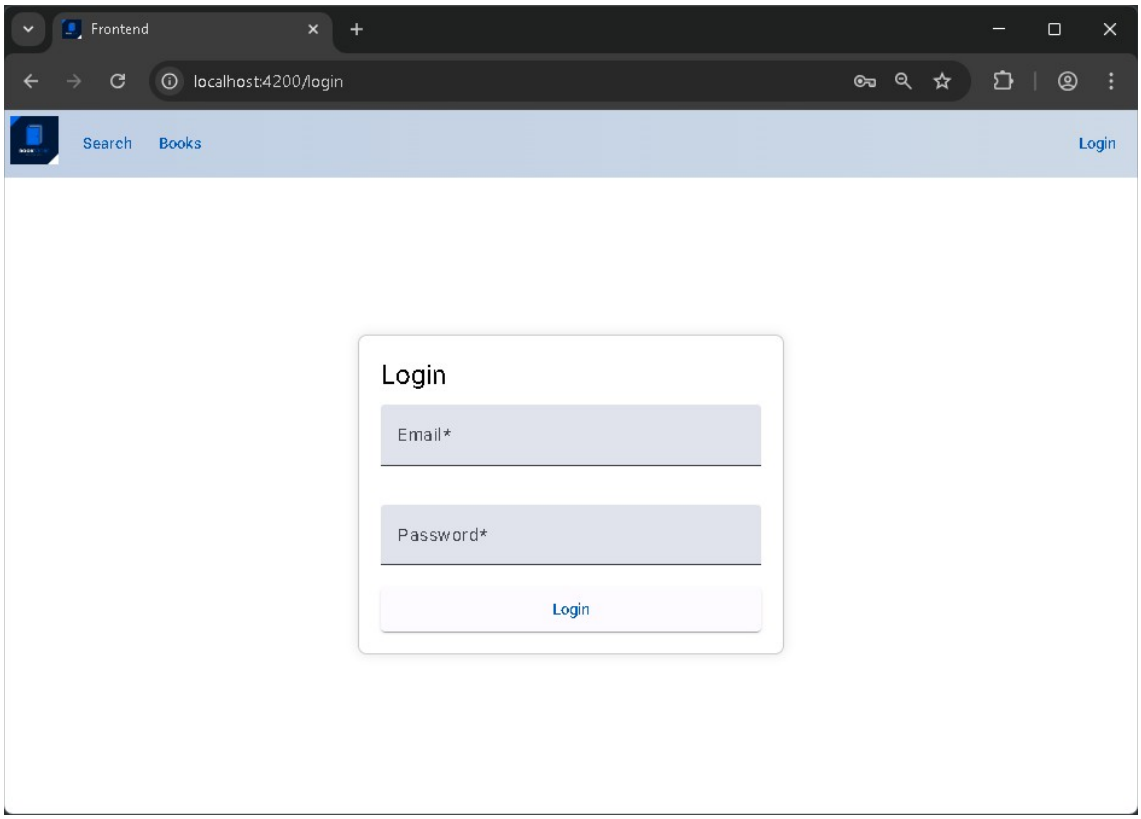
- Logout
- Edit and view book details
- Edit user profile

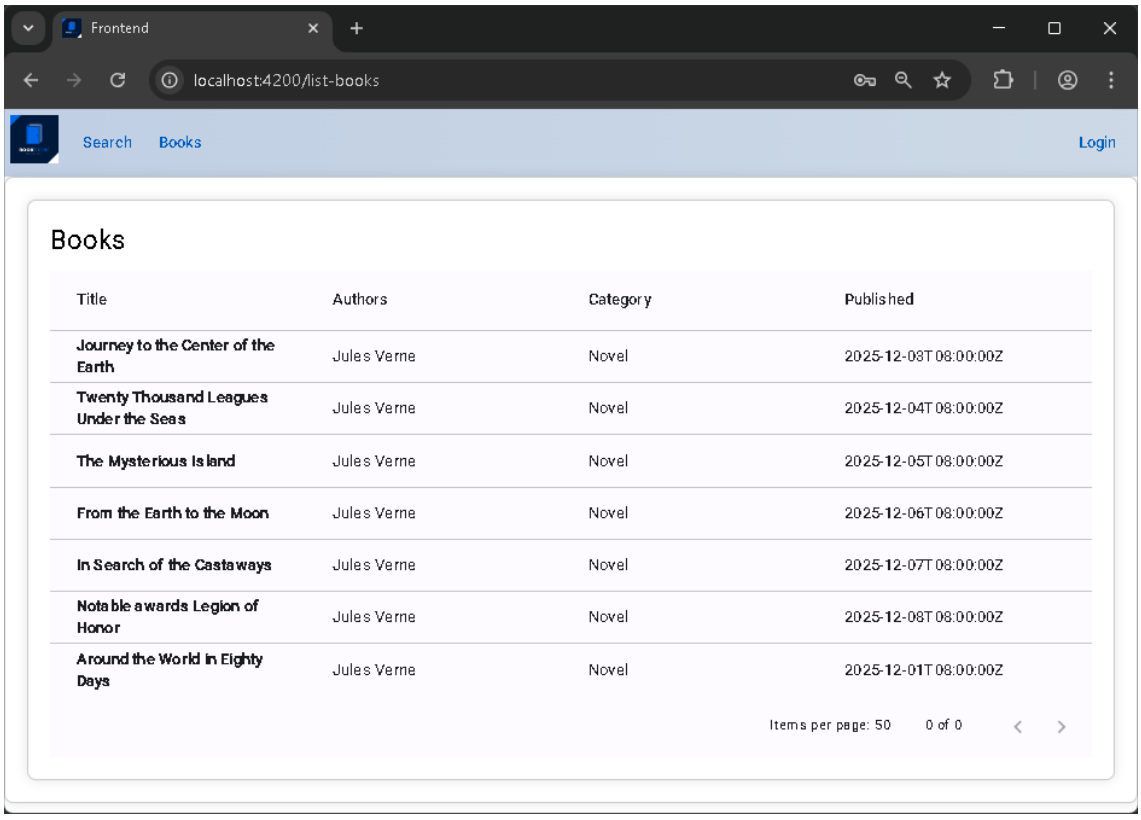
The administrator user in addition to the last pretermissions can:

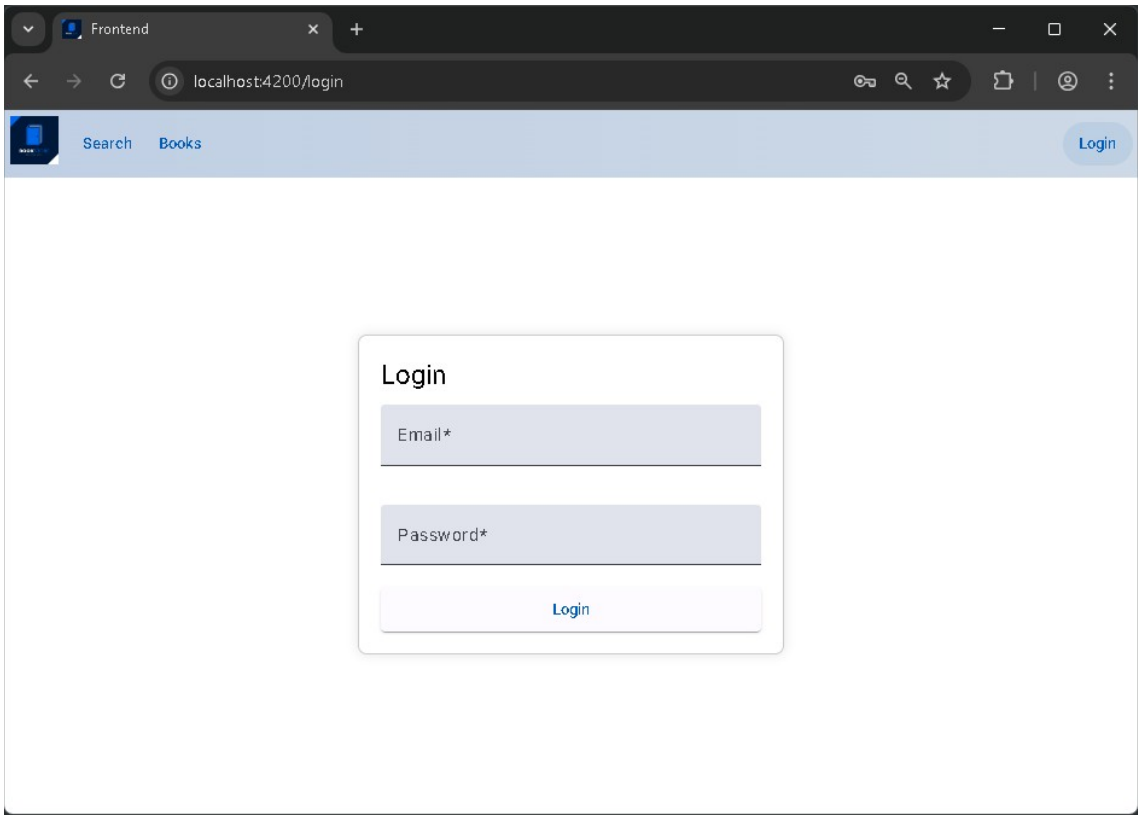
- Add and delete books
- Add and delete users and administrators
- Revoke user permission to login
- Revoke all permissions to login
- List users and view details

Anonymous user screens

Here some screenshots for the anonymous user.

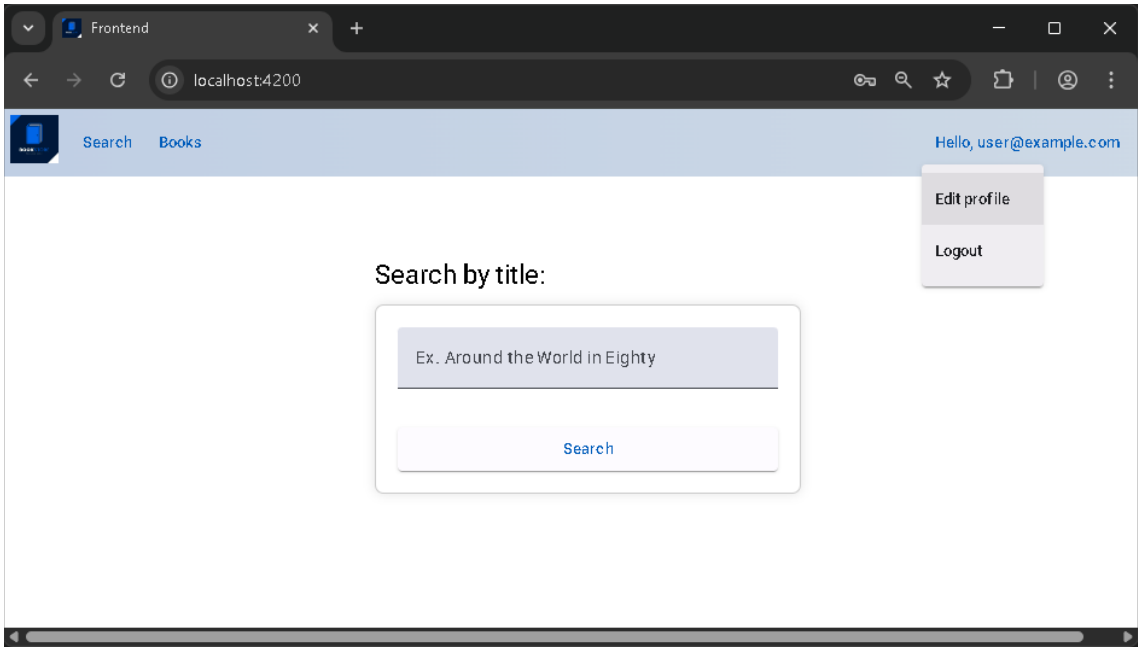

Frontend
localhost:4200/login
Search
Books
Login
Login
Email*
Password*
Login

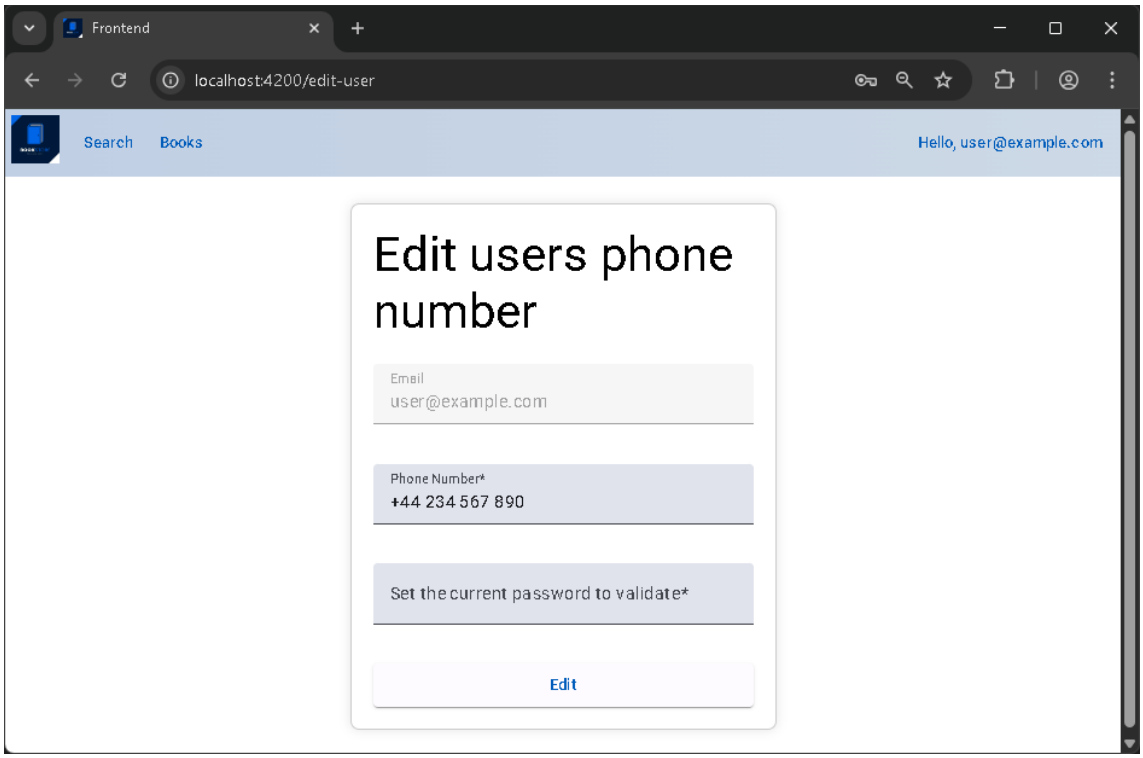

Frontend
localhost:4200/list-books
Search
Books
Login
Books
Title
Authors
Category
Published
Journey to the Center of the Earth
Jules Verne
Novel
2025-12-03T08:00:00Z
Twenty Thousand Leagues Under the Seas
Jules Verne
Novel
2025-12-04T08:00:00Z
The Mysterious Island
Jules Verne
Novel
2025-12-05T08:00:00Z
From the Earth to the Moon
Jules Verne
Novel
2025-12-06T08:00:00Z
In Search of the Castaways
Jules Verne
Novel
2025-12-07T08:00:00Z
Notable awards Legion of Honor
Jules Verne
Novel
2025-12-08T08:00:00Z
Around the World in Eighty Days
Jules Verne
Novel
2025-12-01T08:00:00Z
Items per page: 50
0 of 0

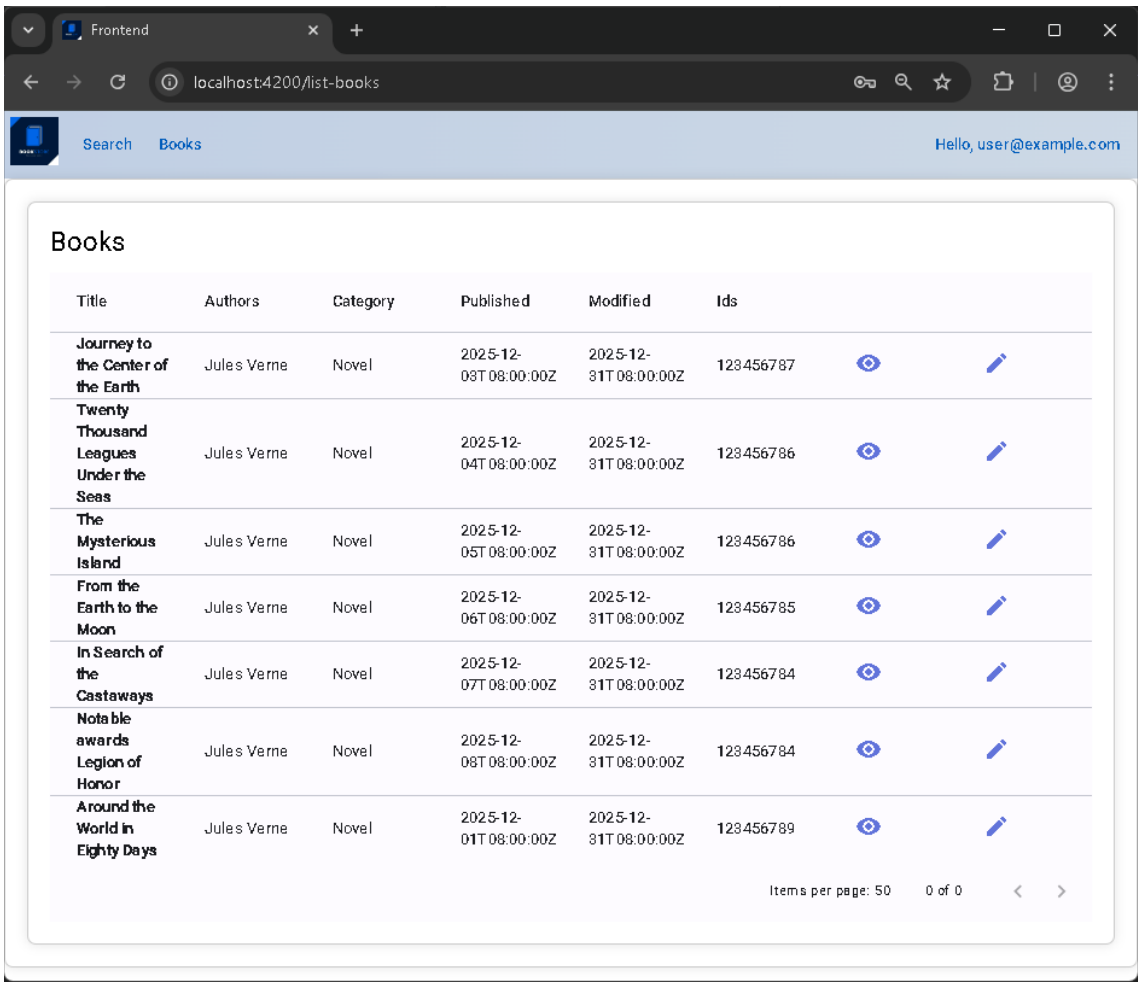


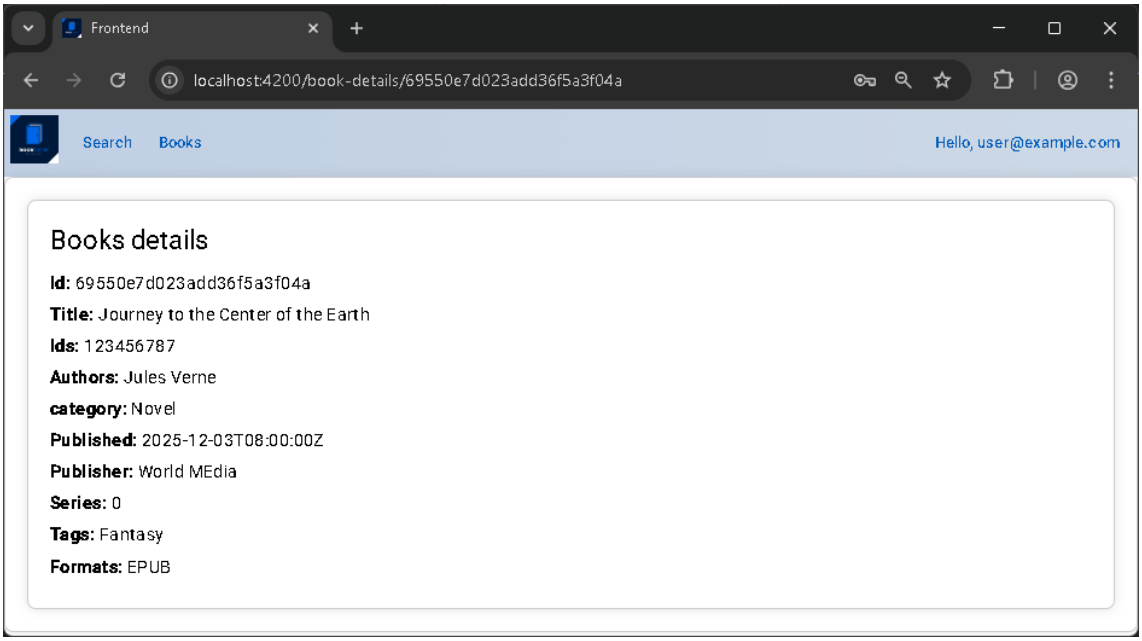
Authenticated user screens

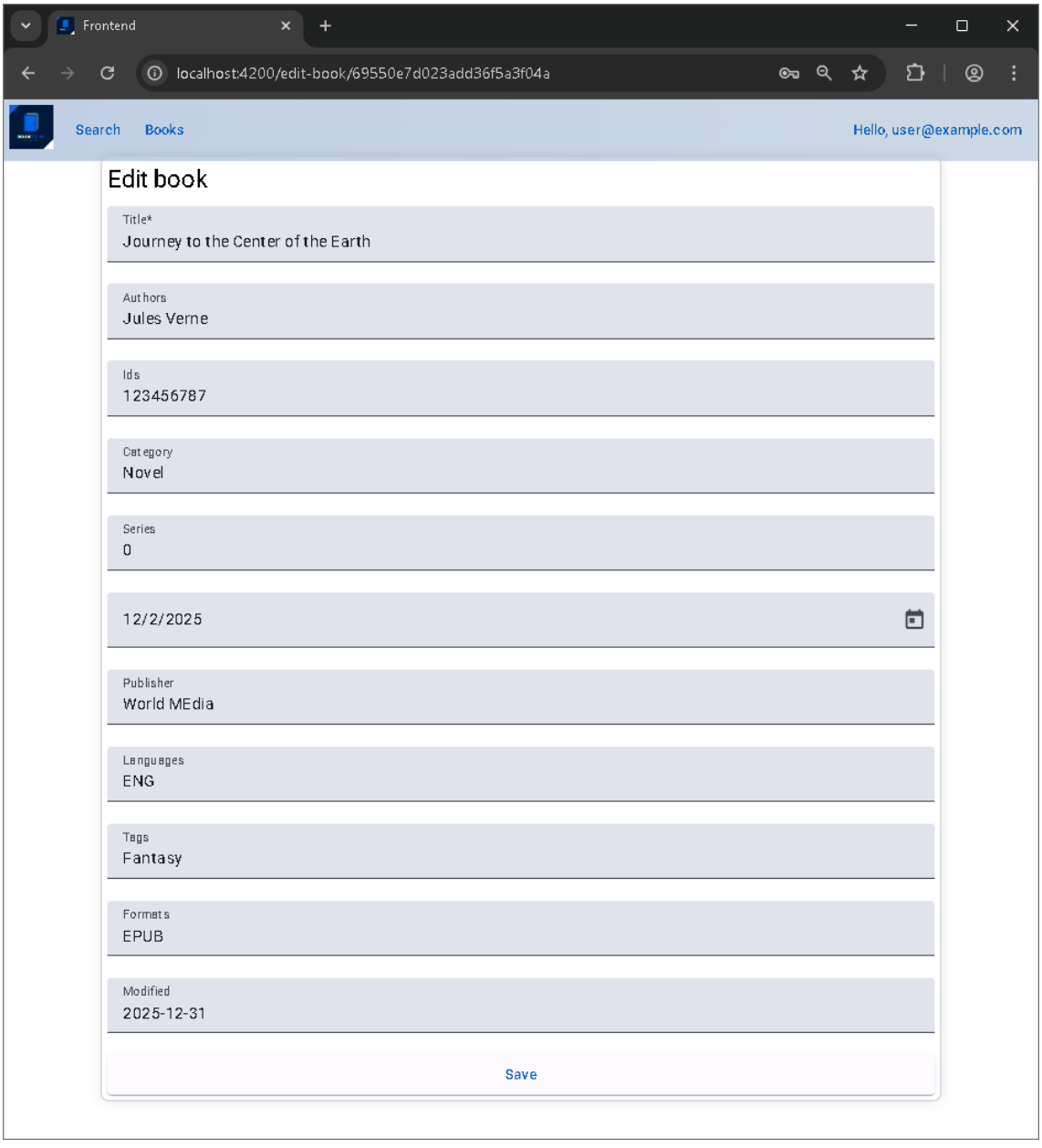
Here some screenshots for the authenticated user.




Frontend
localhost:4200/edit-user
Search
Books
Hello, user@example.com
Edit users phone number
Email
user@example.com
Phone Number*
+44 234 567 890
Set the current password to validate*
Edit

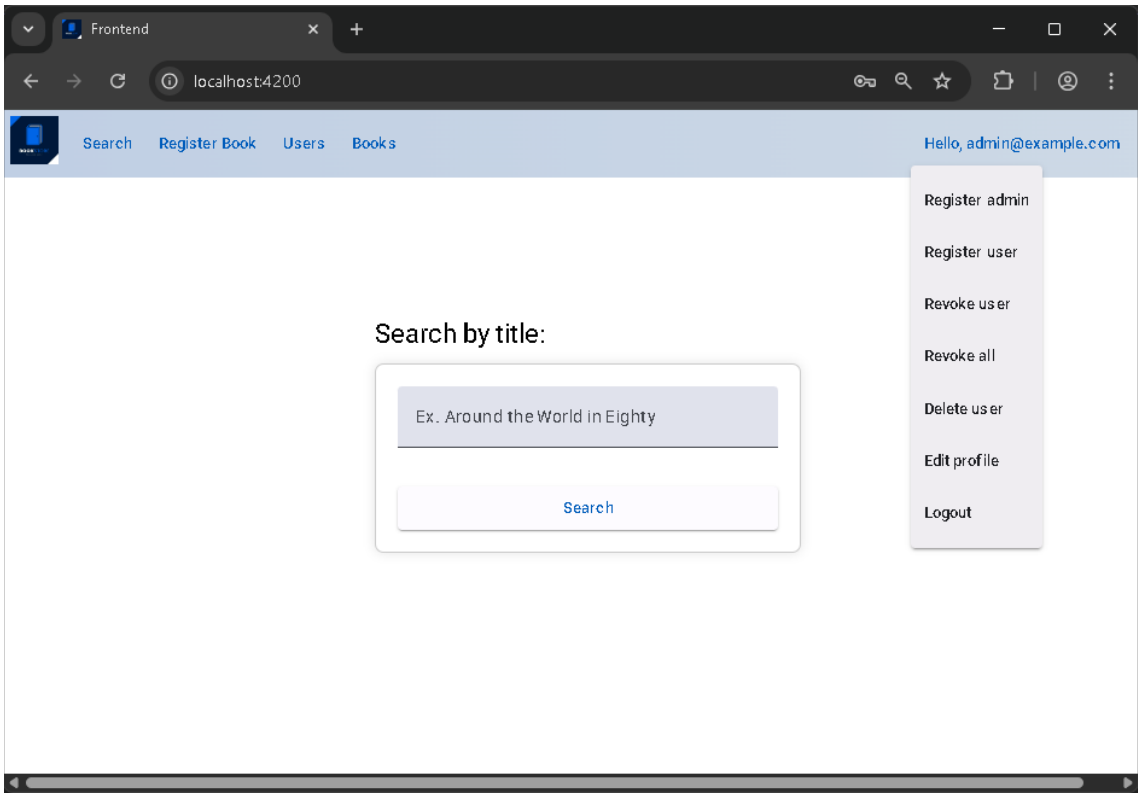

Frontend
localhost:4200/list-books
Search
Books
Hello, user@example.com
Books
Title
Authors
Category
Published
Modified
Ids
Journey to the Center of the Earth
Jules Verne
Novel
2025-12-03T08:00:00Z
2025-12-31T08:00:00Z
123456787
Twenty Thousand Leagues Under the Seas
Jules Verne
Novel
2025-12-04T08:00:00Z
2025-12-31T08:00:00Z
123456786
The Mysterious Island
Jules Verne
Novel
2025-12-05T08:00:00Z
2025-12-31T08:00:00Z
123456786
From the Earth to the Moon
Jules Verne
Novel
2025-12-06T08:00:00Z
2025-12-31T08:00:00Z
123456785
In Search of the Castaways
Jules Verne
Novel
2025-12-07T08:00:00Z
2025-12-31T08:00:00Z
123456784
Notable awards Legion of Honor
Jules Verne
Novel
2025-12-08T08:00:00Z
2025-12-31T08:00:00Z
123456784
Around the World in Eighty Days
Jules Verne
Novel
2025-12-01T08:00:00Z
2025-12-31T08:00:00Z
123456789
Items per page: 50
0 of 0

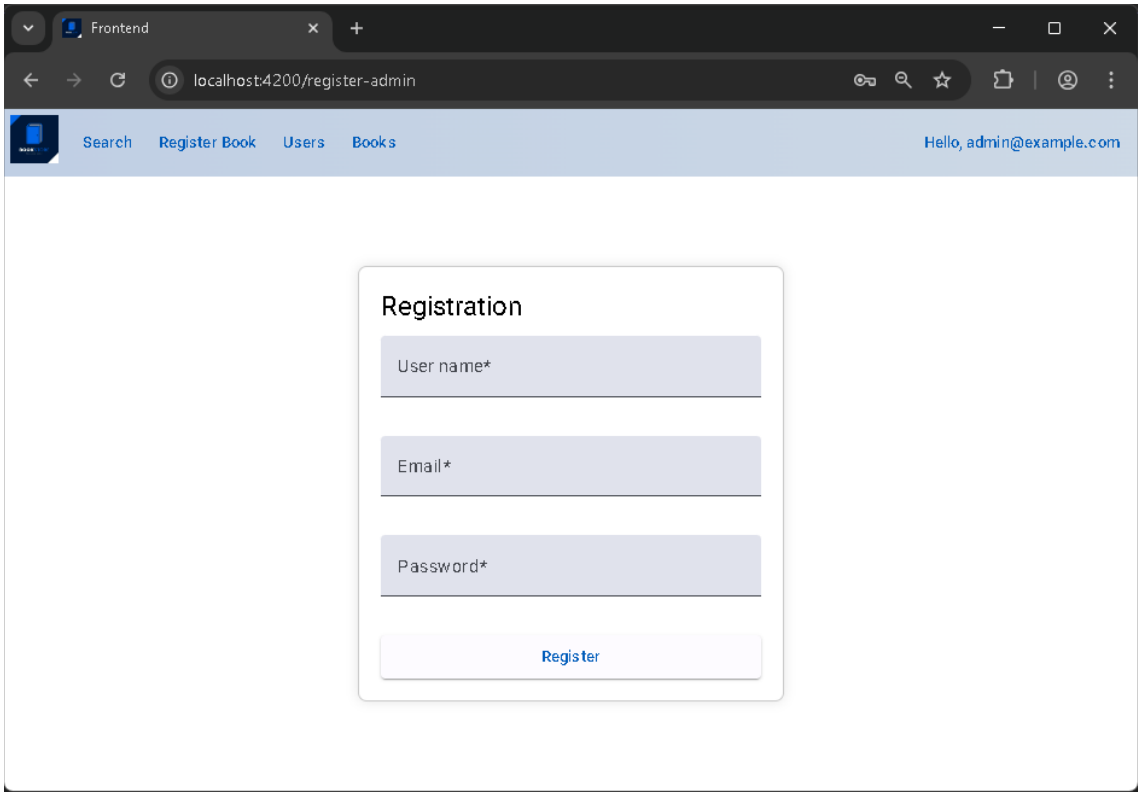

Frontend
localhost:4200/book-details/69550e7d023add36f5a3f04a
Search
Books
Hello, user@example.com
Books details
Id: 69550e7d023add36f5a3f04a
Title: Journey to the Center of the Earth
Ids: 123456787
Authors: Jules Verne
category: Novel
Published: 2025-12-03T08:00:00Z
Publisher: World MEdia
Series: 0
Tags: Fantasy
Formats: EPUB

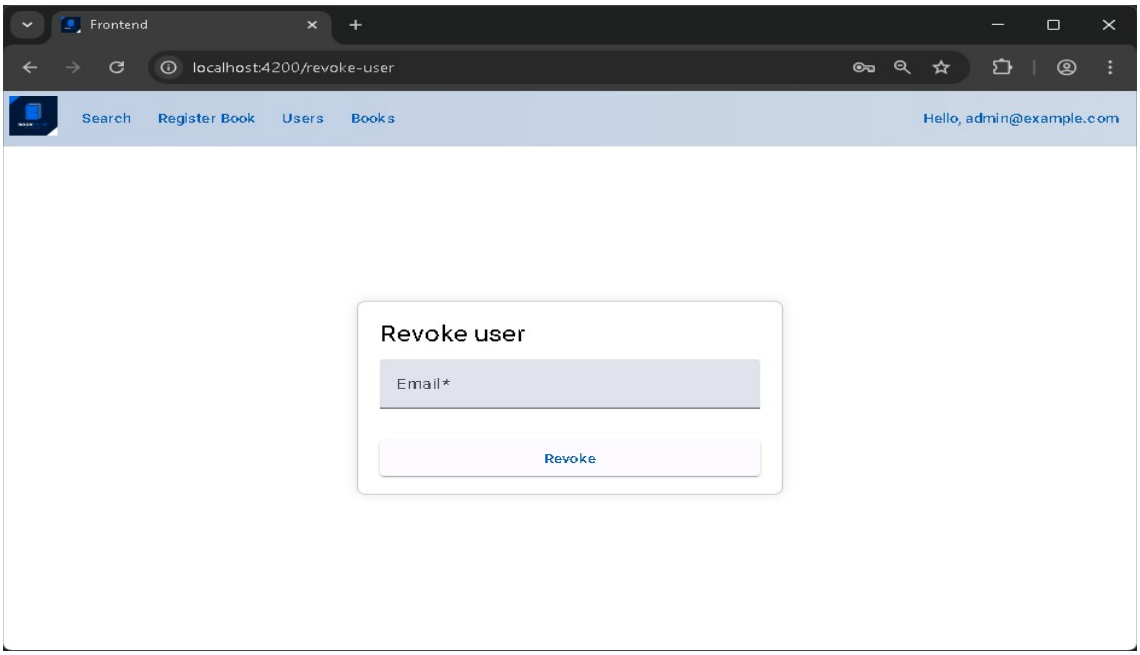

Frontend
localhost:4200/edit-book/69550e7d023add36f5a3f04a
Search
Books
Hello, user@example.com
Edit book
Title*
Journey to the Center of the Earth
Authors
Jules Verne
Ids
123456787
Category
Novel
Series
0
12/2/2025
Publisher
World MEdia
Languages
ENG
Tags
Fantasy
Formats
EPUB
Modified
2025-12-31
Save

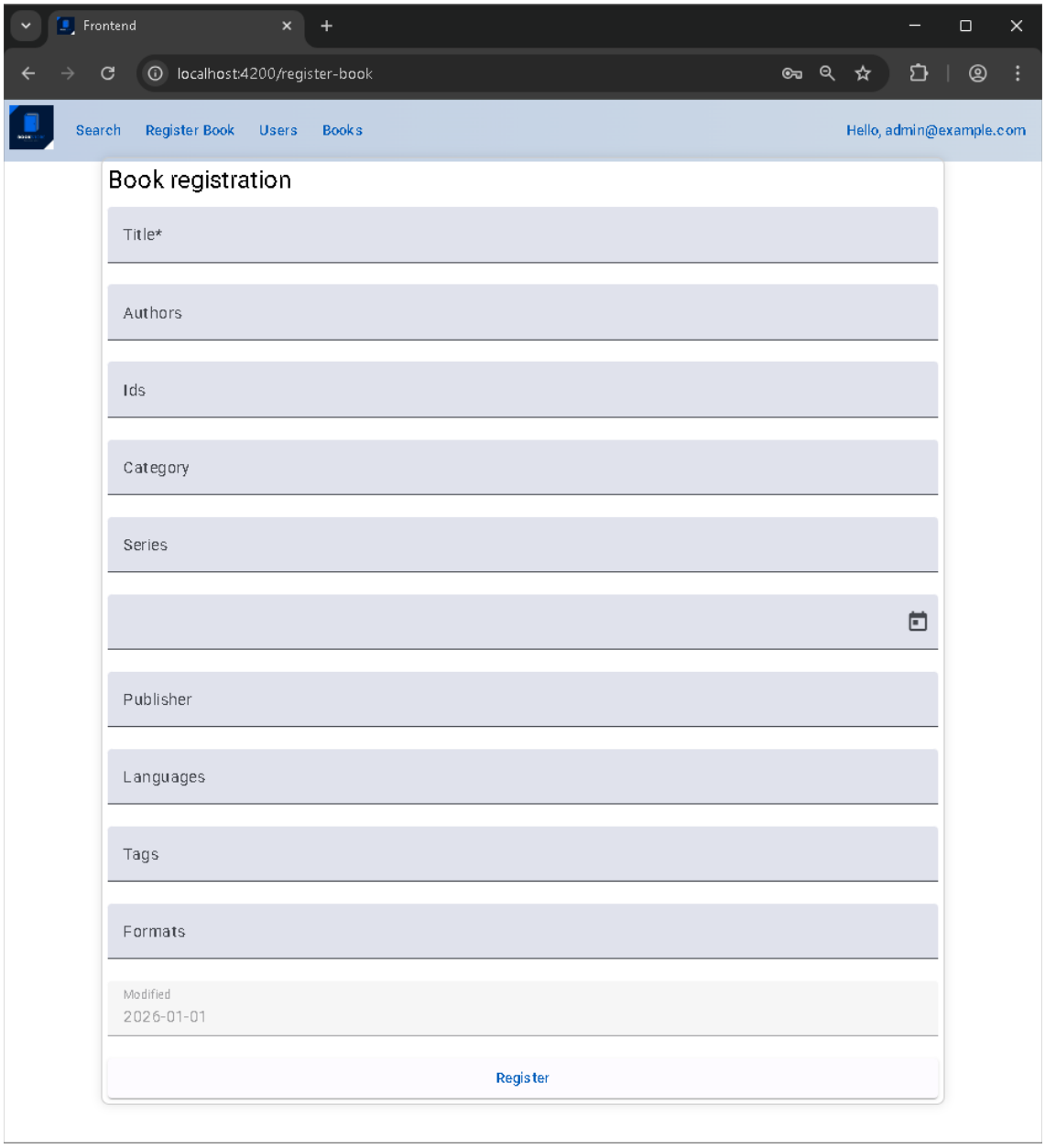
Administrator user screens

Here some screenshots for the administrator user.






Frontend
localhost:4200/revoke-user
Search
Register Book
Users
Books
Hello, admin@example.com
Revoke user
Email*
Revoke

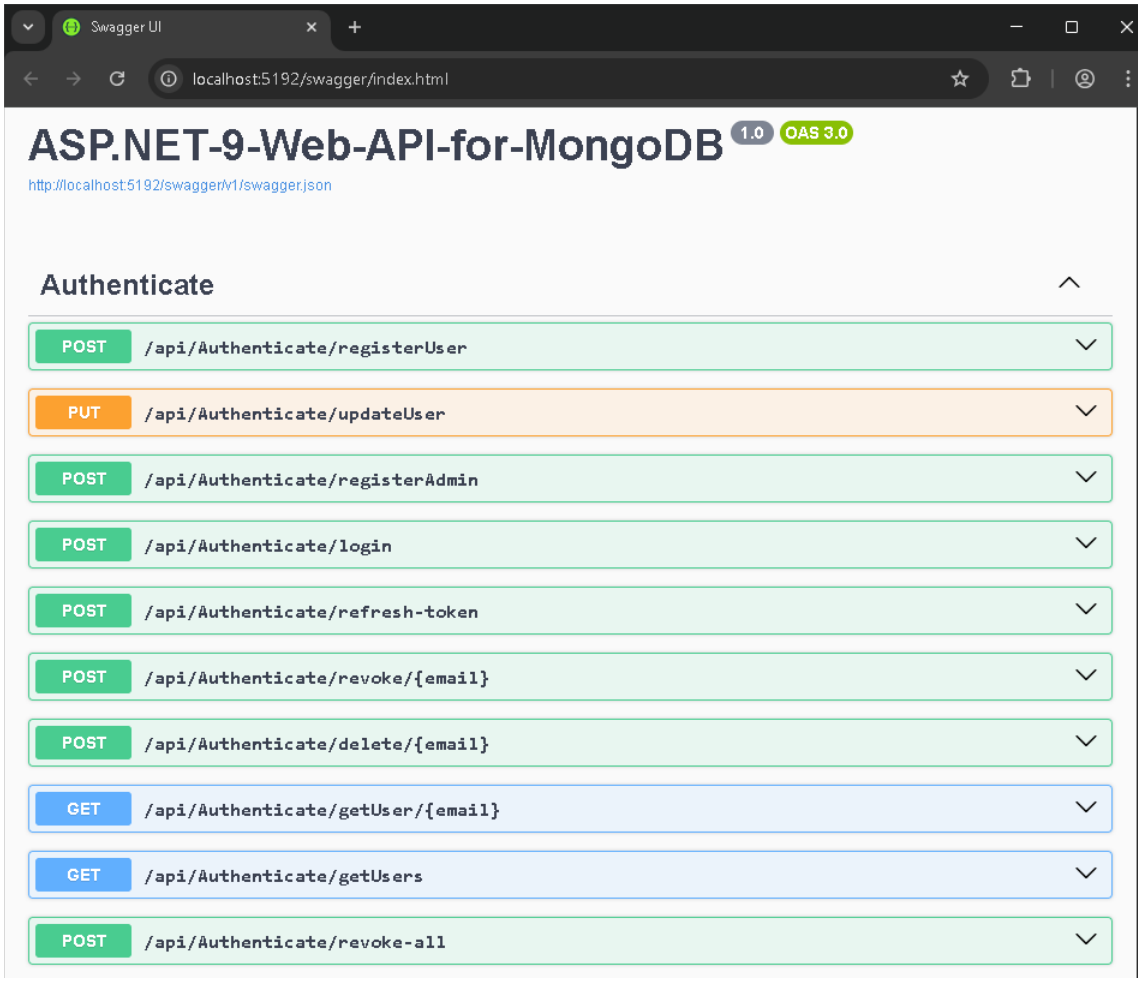

Frontend
localhost:4200/register-book
Search
Register Book
Users
Books
Hello, admin@example.com
Book registration
Title*
Authors
Ids
Category
Series
Publisher
Languages
Tags
Formats
Modified
2026-01-01
Register

Scope

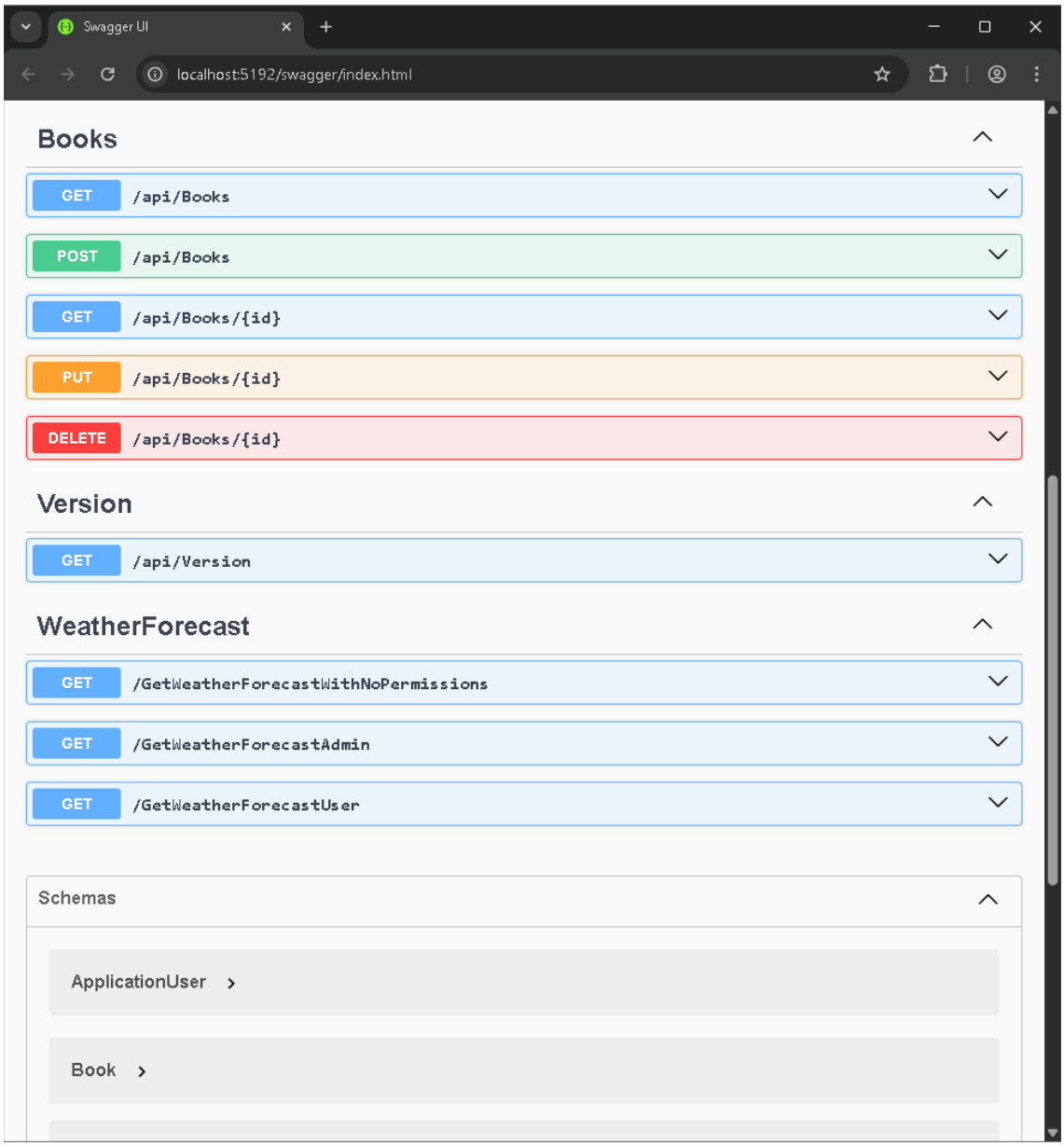
The aim of this document is to describe a full-stack solution for containerised applications to separate them from your infrastructure so you can deliver software independently to it, providing scalability, high availability, and performance, typically configured as either a Replica Set for data redundancy (automated failover) or a Shared Cluster.

RQ-01 | Web API

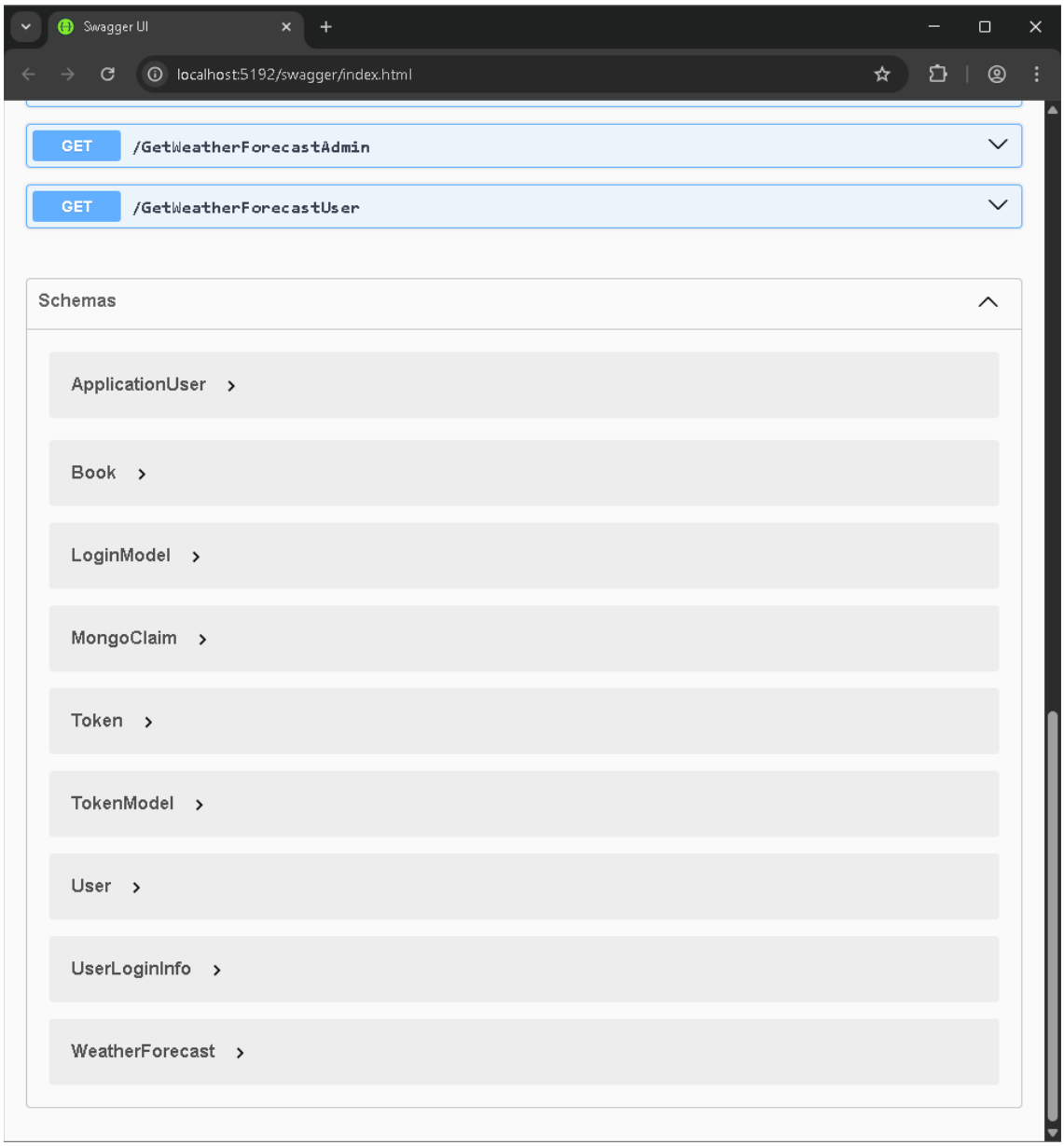
Here the authentication API offered by Swagger.



Here the books API.



And the schemas.



Security

The security aspects to take in account are described below.

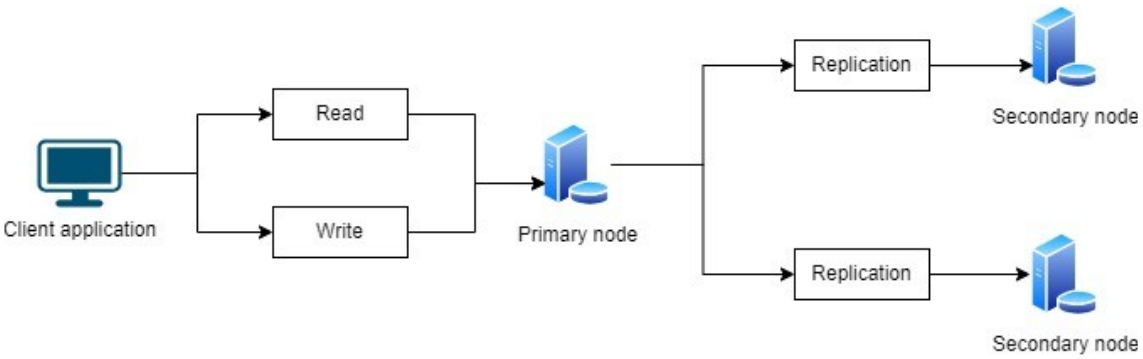
Securitised endpoints

Initial setup for Admin

Revoke refresh token

RQ-02 | MongoDB Cluster

MongoDB is a general-purpose database that was built with the web in mind. Amongst other things, it offers high availability when used in clusters, also called replica sets. A replica set is a group of MongoDB servers, called nodes, containing an identical copy of the data. If one of the servers fails, the other two will pick up the load while the crashed one restarts.



There are many different ways to create a MongoDB cluster. The easiest way is by using MongoDB Atlas, the database-as-a-service offering by MongoDB. With just a few clicks, you can create your free cluster, hosted in the cloud.

However, if you need to experiment with MongoDB clusters, you can use Docker to create a cluster on your personal computer. Below I will provide you with the necessary instructions to create your own MongoDB Docker cluster locally.

If you do not want to install MongoDB on your laptop, you can use Docker to run your cluster. Docker is an application to launch containers, packages that contain applications along with all the required dependencies necessary to run them.

Setup

Here the steps to create MongoDB Cluster with initial data.

We run Docker Desktop application, and after that, we pull the MongoDb official image from Docker Hub to our local host.

Create a Docker network

The first step is to create a Docker network. This network will let each of your containers running in this network see each other. To create a network, run the docker network create command

```
docker network create mongoCluster
```

Start MongoDB Instances

You are now ready to start your first container with MongoDB. To start the container, use the docker run command.

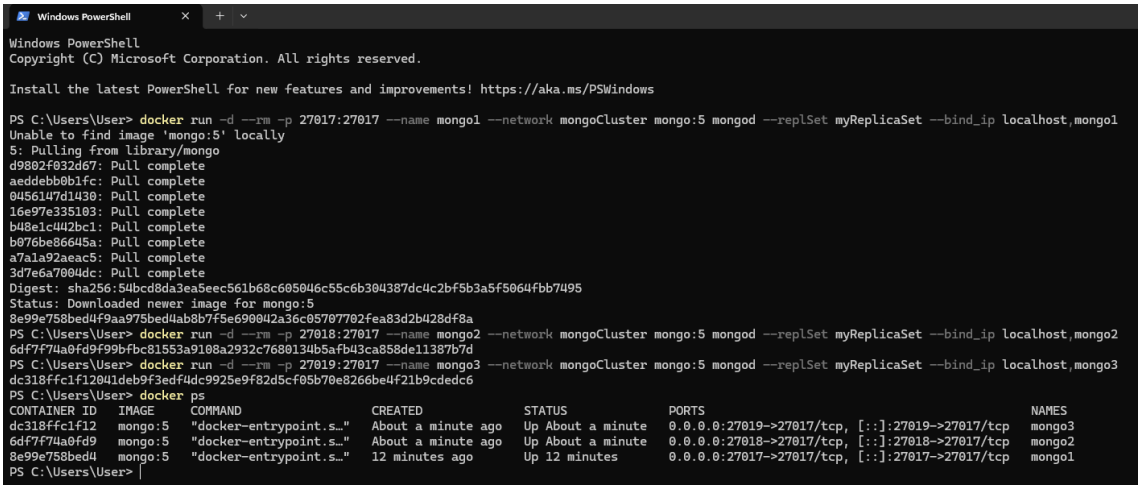
```
docker run -d --rm -p 27017:27017 --name mongo1 --network mongoCluster
mongo:latest mongod --replSet myReplicaSet --bind_ip localhost,mongo1
```

The command that will be executed once to start the container is started, it will create a new mongod instance ready for a replica set. Start two other containers. You will need to use a different name and a different port for those two.

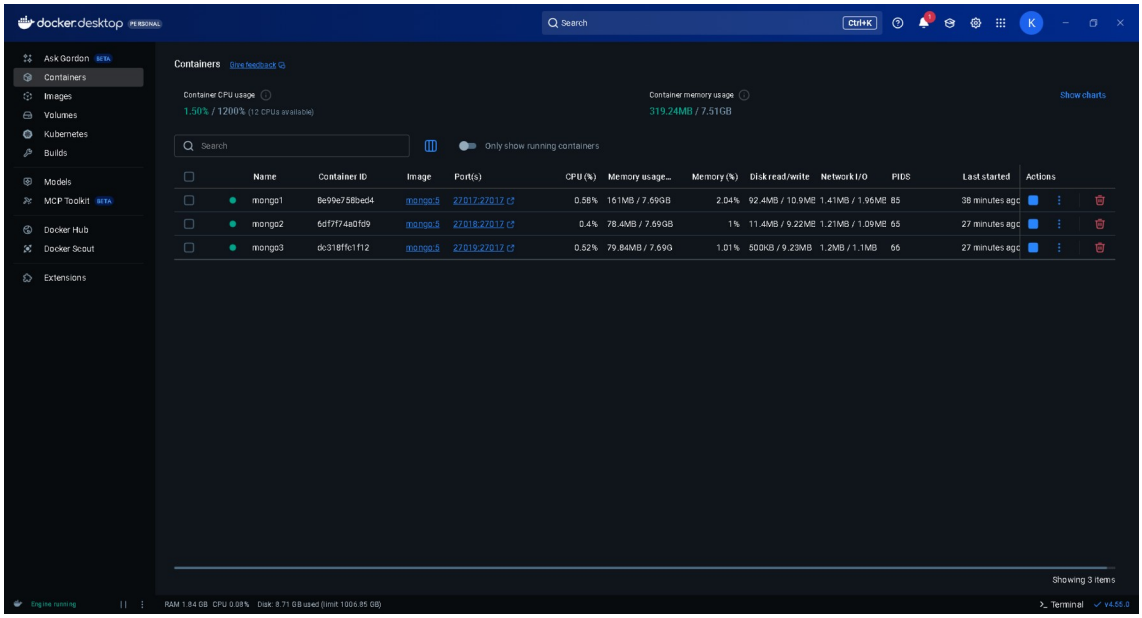
```
docker run -d --rm -p 27018:27017 --name mongo2 --network mongoCluster
mongo:5 mongod --replSet myReplicaSet --bind_ip localhost,mongo2

docker run -d --rm -p 27019:27017 --name mongo3 --network mongoCluster
mongo:5 mongod --replSet myReplicaSet --bind_ip localhost,mongo3
```

You can use docker ps to validate that they are running.



Or open the Docker application and check the containers.



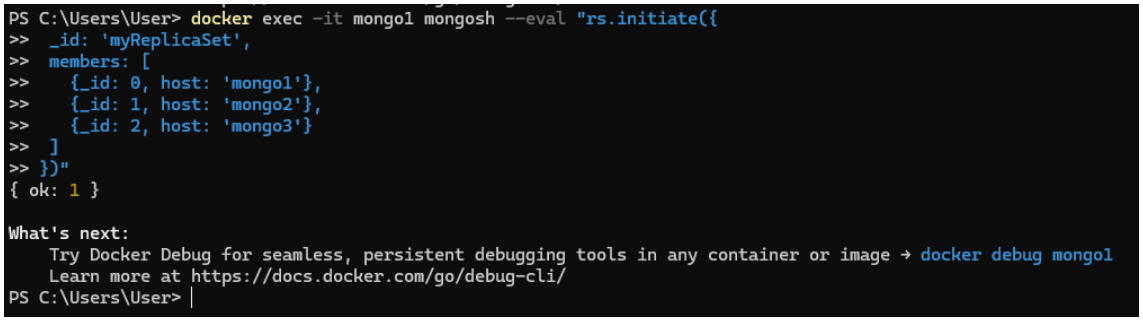
Initiate the Replica Set

The next step is to create the actual replica set with the three members. To do so, you will need to use the MongoDB Shell. This CLI (command-line interface) tool is available with the default MongoDB installation or installed independently.

However, if you don’t have the tool installed on your laptop, it is possible to use mongosh available inside containers with the docker exec command.

```
docker exec -it mongo1 mongosh --eval "rs.initiate({
 _id: 'myReplicaSet',
 members: [
   {_id: 0, host: 'mongo1'},
   {_id: 1, host: 'mongo2'},
   {_id: 2, host: 'mongo3'}
 ]
})"
```

NOTE: If you get an error because of the single quotes, try executing the command with double quotes instead or execute mongosh without the --eval parameter and then type the rs.initiate() command directly in the CLI.



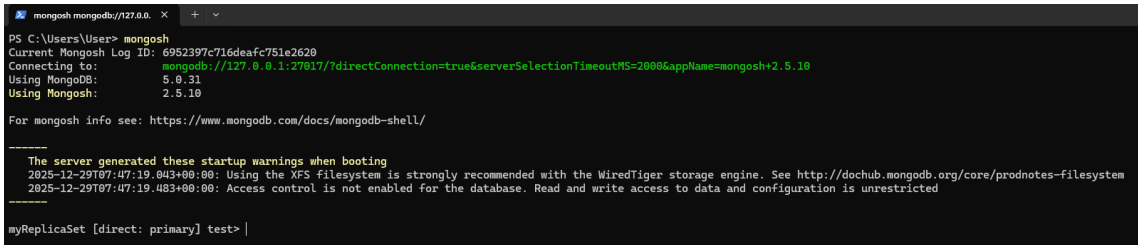
This command tells Docker to run the mongosh tool inside the container named mongo1. mongosh will then try to evaluate the rs.initiate() command to initiate the replica set. If the command was successfully executed, you should see a message from the mongosh CLI indicating the version numbers of MongoDB and Mongosh, followed by a message indicating:{ ok: 1 }.

Test and Verify the Replica Set

You should now have a running replica set. If you want to verify that everything was configured correctly, you can use the mongosh CLI tool to evaluate the rs.status() instruction. This will provide you with the status of your replica set, including the list of members.

```
docker exec -it mongo1 mongosh --eval "rs.status()"
```

If we launch mongosh comand, we will see that by default we will be connected to the primary node.



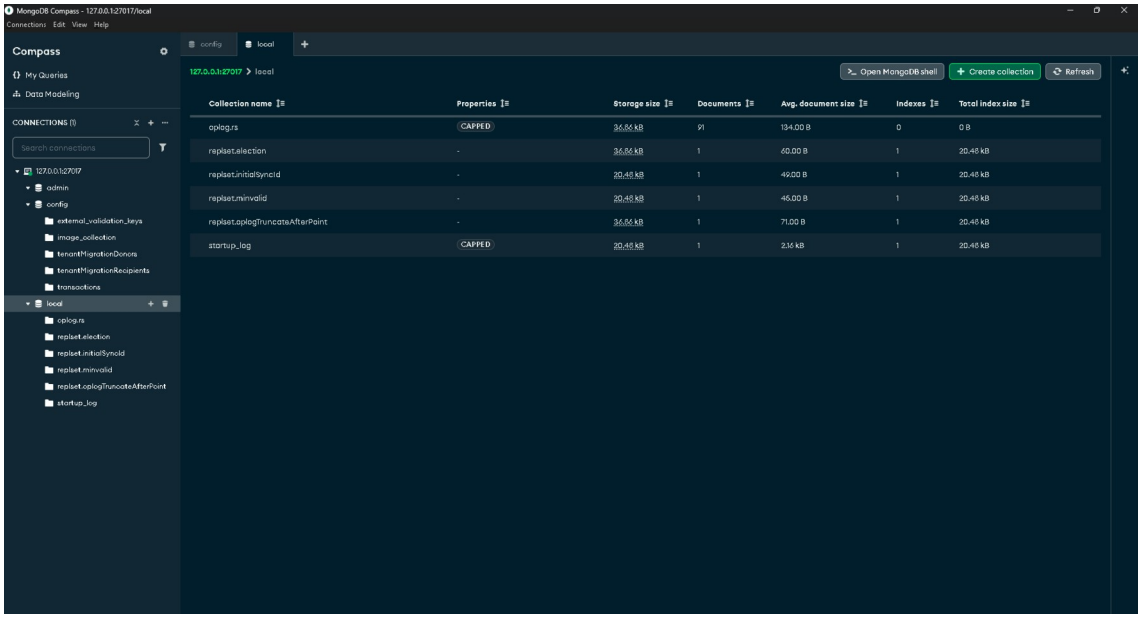
As we can notice, the connection string to test from visual studio is:

```
mongodb://127.0.0.1:27017/?
directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.5.10
```

And to test inside the container in Docker is:

```
"mongodb://host.docker.internal:27017/?
directConnection=true&serverSelectionTimeoutMS=6000&appName=mongosh+2.5.10",
"Identity"
```

If we connect with Compass we will see the initial database configuration settings for the replica set.



RQ-04 | Angular App

Angular 18 is a significant update focused on performance, developer experience, and modernising the framework, highlighted by the move towards zoneless change detection using signals, faster server-side rendering (SSR) with partial hydration, stable deferrable views, better micro-frontend integration, improved routing with function-based redirects, and enhanced Material/CDK compatibility, all leading to smaller bundles, quicker load times, and simpler development.

Requirements

```
npm install -g @angular/cli
```

```
npm install @angular/cli
```

Add *"C:\Users\User\AppData\Roaming\npm\node_modules\@angular\cli\bin"* to the PATH environment parameters.

Angular Material UI

The UI is built using Material components.

State Management

In Angular 18, signals (introduced in v16) are used for fine-grained, efficient state management and reactivity, acting as wrappers around values that notify interested parts of the application when they change, optimising rendering by only updating what's necessary, reducing boilerplate, and simplifying reactive data flows for things like component state, forms, and even global stores.

Here the imports.

And here an example of implementation.

navbar.component.ts M

src > app > components > navbar > navbar.component.ts > NavbarComponent

```
26  export class NavbarComponent {
38    ngOnInit() {
39      this.username$.subscribe((response) => {
40        this.username = signal<string>(response || '');
41      });
42      this.username = signal<string>(
43        localStorage.getItem('Template_email') || ''
44      );
45    }
```

Security

The security aspects to take in account are described below.

Control route access with guards

CRITICAL: Never rely on client-side guards as the sole source of access control. All JavaScript that runs in a web browser can be modified by the user running the browser. Always enforce user authorization server-side, in addition to any client-side guards.

Route guards are functions that control whether a user can navigate to or leave a particular route. They are like checkpoints that manage whether a user can access specific routes. Common examples of using route guards include authentication and access control.

To create a new guard execute the next command.

```
ng generate guard auth
```

The implementation for the routes.

app.routes.ts M | auth.guard.ts M

src > app > app.routes.ts > ...

```
1   import { Routes } from '@angular/router';
2   import { RegisterAdminComponent } from './components/register-admin/registerAdmin.component';
3   import { RegisterUserComponent } from './components/register-user/registerUser.component';
4   import { LoginComponent } from './components/login/login.component';
5   import { HomeComponent } from './components/home/home.component';
6   import { EditUserComponent } from './components/edit-user/edit-user.component';
7   import { authGuard } from './guards/auth.guard';
8
9   export const routes: Routes = [
10    { path: 'login', component: LoginComponent },
11    { path: 'home', component: HomeComponent, canActivate: [authGuard]},
12    { path:  '', component: HomeComponent, canActivate: [authGuard] },
13    { path: 'register-admin', component: RegisterAdminComponent, canActivate: [authGuard] },
14    { path: 'register-user', component: RegisterUserComponent, canActivate: [authGuard] },
15    { path: 'edit-user', component: EditUserComponent, canActivate: [authGuard] },
16
17  ];
18
```

The implementation of the guards.

```ts
// app.routes.ts M | auth.guard.ts M
// src > app > guards > auth.guard.ts > ...
import { CanActivateFn } from '@angular/router';
import { UserService } from '../services/user.service';
import { inject } from '@angular/core';

export const authGuard: CanActivateFn = (route, state) => {
  const authService = inject(UserService);
  return authService.isAuthenticated();
};

```

And the implementation of the function in the service.

```ts
// app.routes.ts M | auth.guard.ts M | user.service.ts M
// src > app > services > user.service.ts > UserService > isAuthenticated
  export class UserService {
    isAuthenticated() {
      if(this.isAuthenticatedSubject.value){
        return true;
      }
      else{
        this.router.navigate(['/login']);
        return false;
      }
    }
```

Role based view

For certain components of the application the role based view is implemented. In this case we will distinguish the regular users form administrators.

```ts
export class NavbarComponent {
  isLoggedIn$ = this.userService.isAuthenticated$;
  isAdmin$ = this.userService.isAdmin$;
  username$ = this.userService.username$;
  username: any;
```

So the actions available for administrators is not only secured in the endpoints but also in the UI.

navbar.component.ts M | navbar.component.html

src > app > components > navbar > navbar.component.html > nav

```
<nav>
    <div class="left-side">
      <div class="logo" (click)="homeRedirect()">
      </div>
      <button mat-button color="primary" (click)="homeRedirect()">Home</button>
      <button mat-button color="primary">Users</button>
    </div>
    <div class="right-side">
        @if (isLoggedIn$ | async){
            <button mat-button color="primary" [matMenuTriggerFor]="menu">Hello, {{username()}}</button>
            <mat-menu #menu="matMenu">
              @if (isAdmin$ | async){
                <button mat-menu-item color="primary" (click)="registerAdmin()">Register admin</button>
                <button mat-menu-item color="primary" (click)="registerUser()">Register user</button>
                <button mat-menu-item color="primary" (click)="revokeUser()">Revoke user</button>
                <button mat-menu-item color="primary" (click)="revokeAll()">Revoke all</button>
                <button mat-menu-item color="primary" (click)="deleteUser()">Delete user</button>
              }
              <button mat-menu-item color="primary" (click)="editUser()">Edit profile</button>
              <button mat-menu-item color="primary" (click)="logout()">Logout</button>
            </mat-menu>
        } @else {
            <button mat-button color="primary" (click)="loginRedirect()">Login</button>
        }
    </div>
</nav>
```

Run the App

```
npm install
```

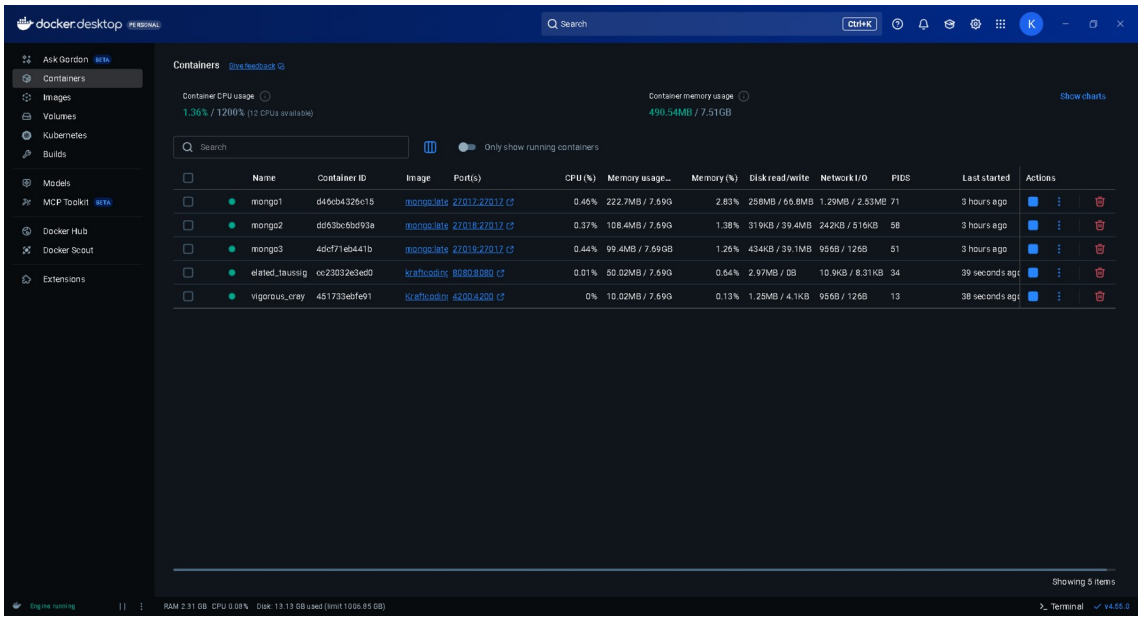
```
npm start
```

```
npm run build
```

RQ-03 | Docker

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code, you can significantly reduce the delay between writing code and running it in production.

The next image shows the containers needed to run the Book Store solution.



To get the stats of memory usage execute:

```
docker stats --no-stream
```

```
C:\Users\User>docker stats --no-stream
CONTAINER ID   NAME             CPU %     MEM USAGE / LIMIT     MEM %     NET I/O           BLOCK I/O         PIDS
451733ebfe91   vigorous_cray    0.00%     10.02MiB / 7.689GiB   0.13%     956B / 126B       1.25MB / 4.1kB    13
cc23032e3ed0   elated_taussig   0.00%     52.95MiB / 7.689GiB   0.67%     82.4kB / 31.2kB   2.97MB / 0B       29
4dcf71eb441b   mongo3           0.35%     99.41MiB / 7.689GiB   1.26%     956B / 126B       434kB / 42.2MB    51
dd63bc6bd93a   mongo2           0.34%     108.4MiB / 7.689GiB   1.38%     282kB / 597kB     319kB / 42.7MB    58
d46cb4326c15   mongo1           0.45%     222.7MiB / 7.689GiB   2.83%     1.4MB / 2.77MB    258MB / 80.9MB    70

C:\Users\User>
```

To purge HDD resources execute:

```
docker system prune
```

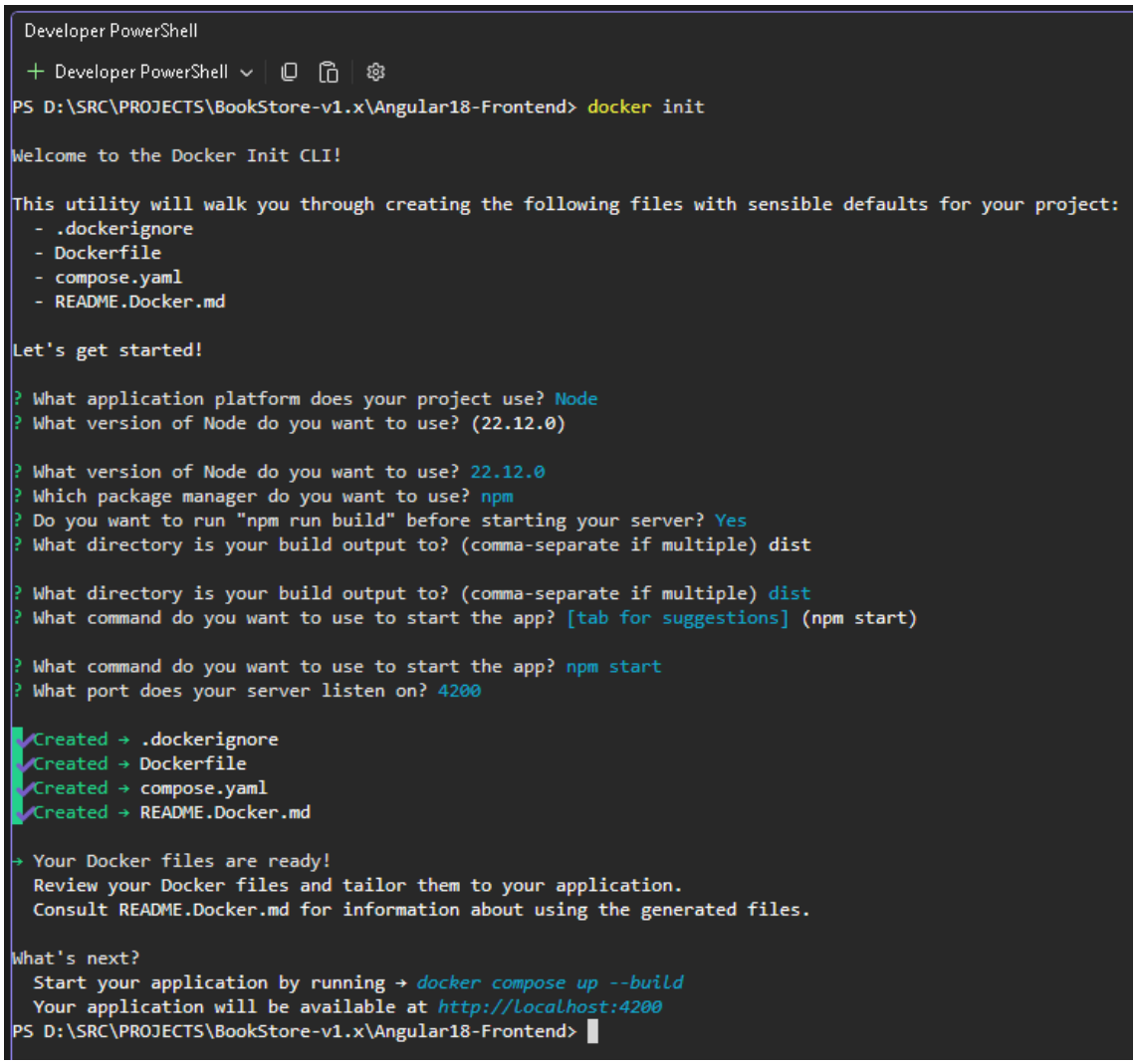
Containerise an Angular Application

Docker provides an interactive CLI tool called docker init that helps scaffold the necessary configuration files for containerising your application.

Setup

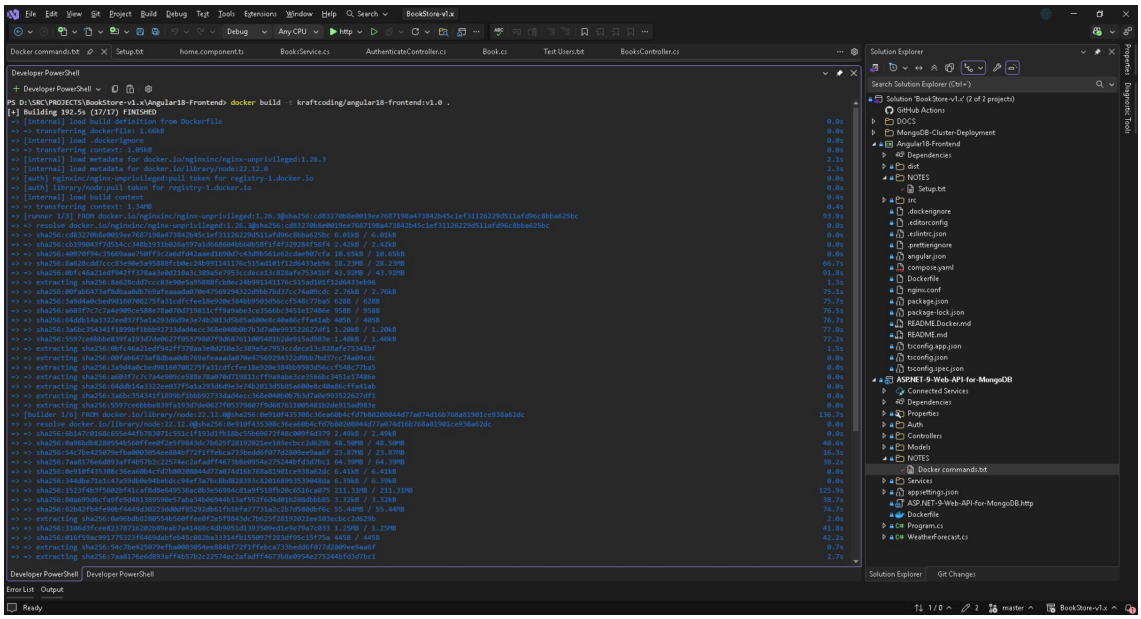
To create the needed files execute:

```
docker init
```

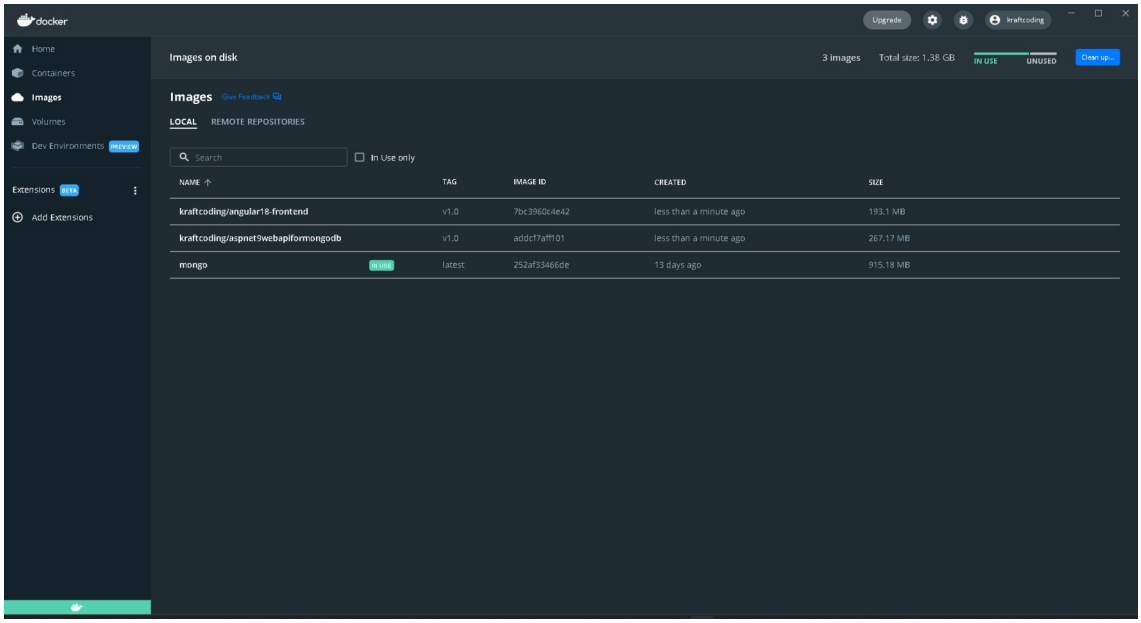


Then we will need to customise de Dockerfile and create nginx.config before building the image.

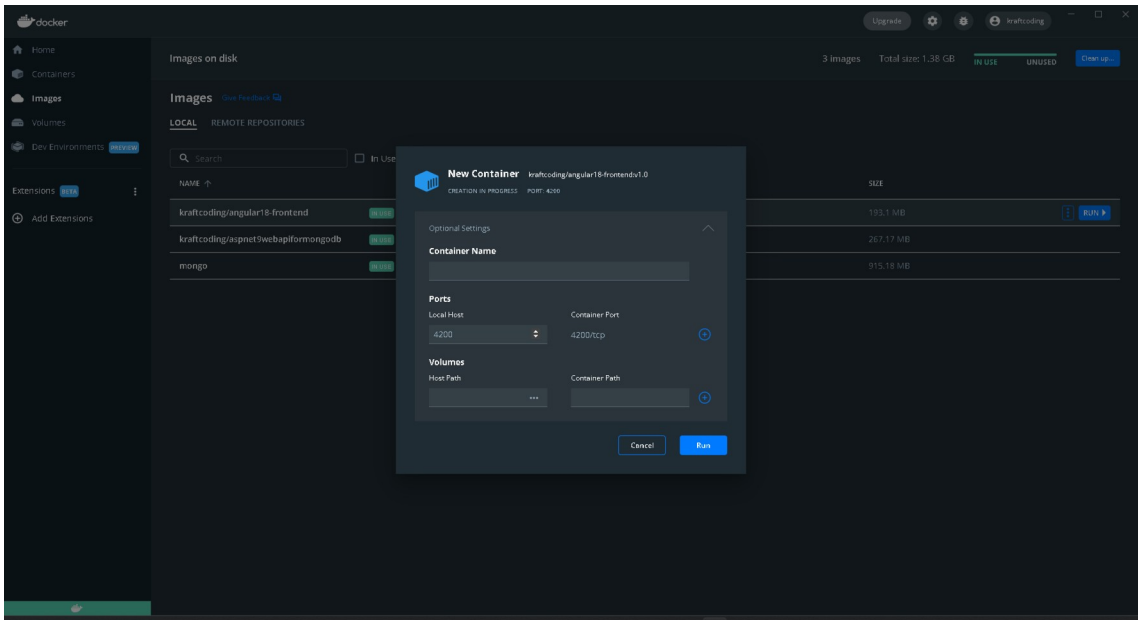
```
docker build -t kraftcoding/angular18-frontend:v1.0 .
```



This will create the image in Docker:



Click run to create the container and access the app in http://localhost:4200.



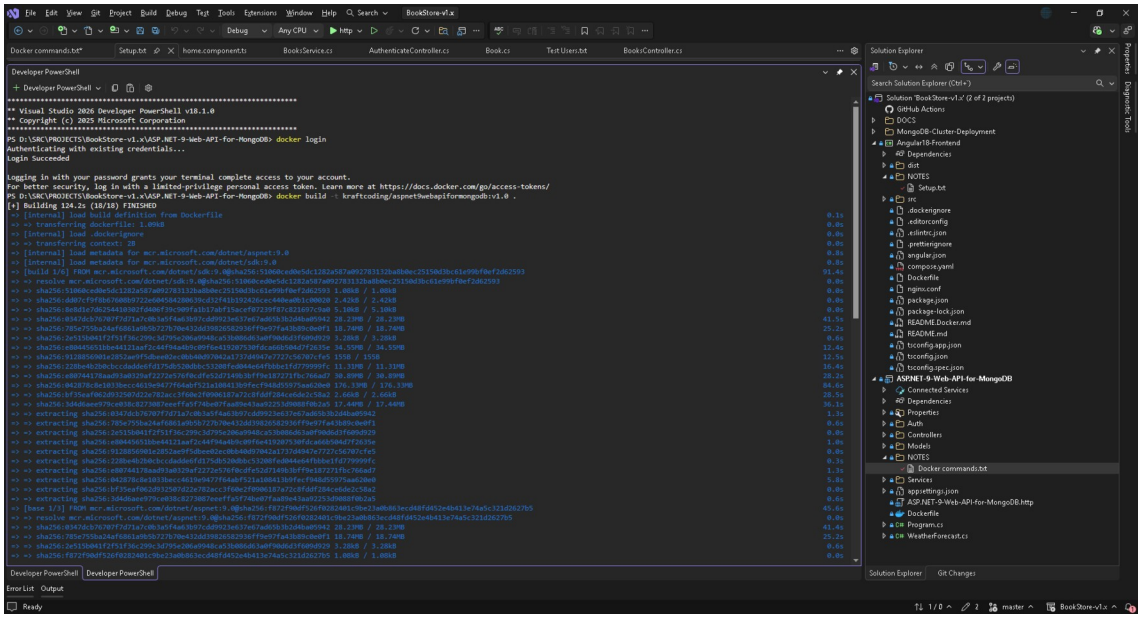
Containerise a .NET Application

Docker provides an interactive CLI tool called docker init that helps scaffold the necessary configuration files for containerising your application.

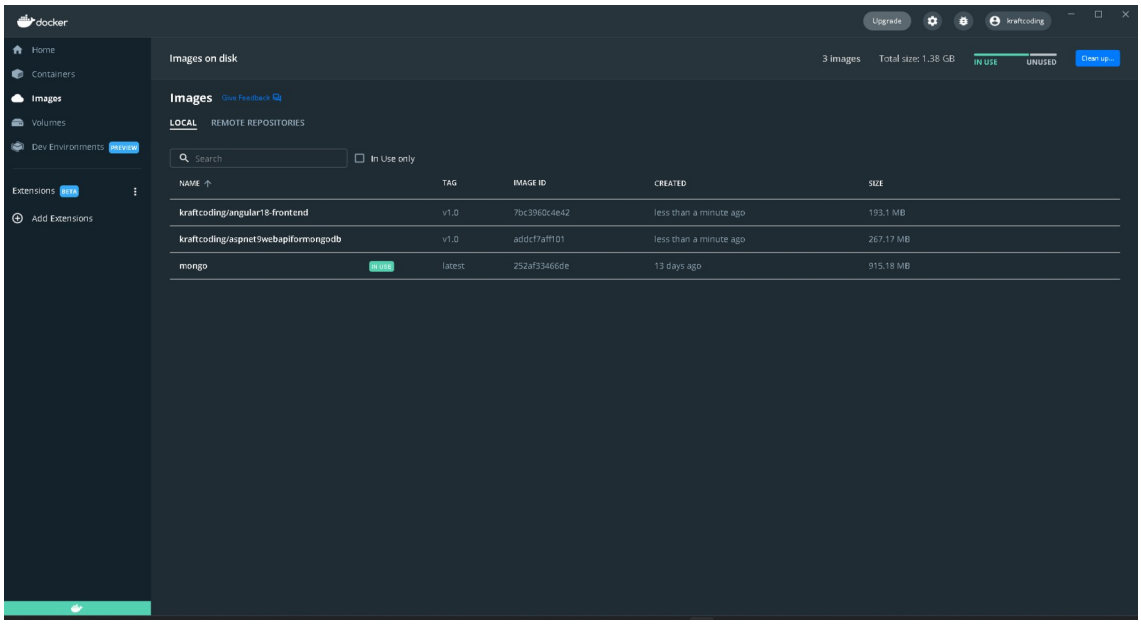
Setup

To create the needed files execute:

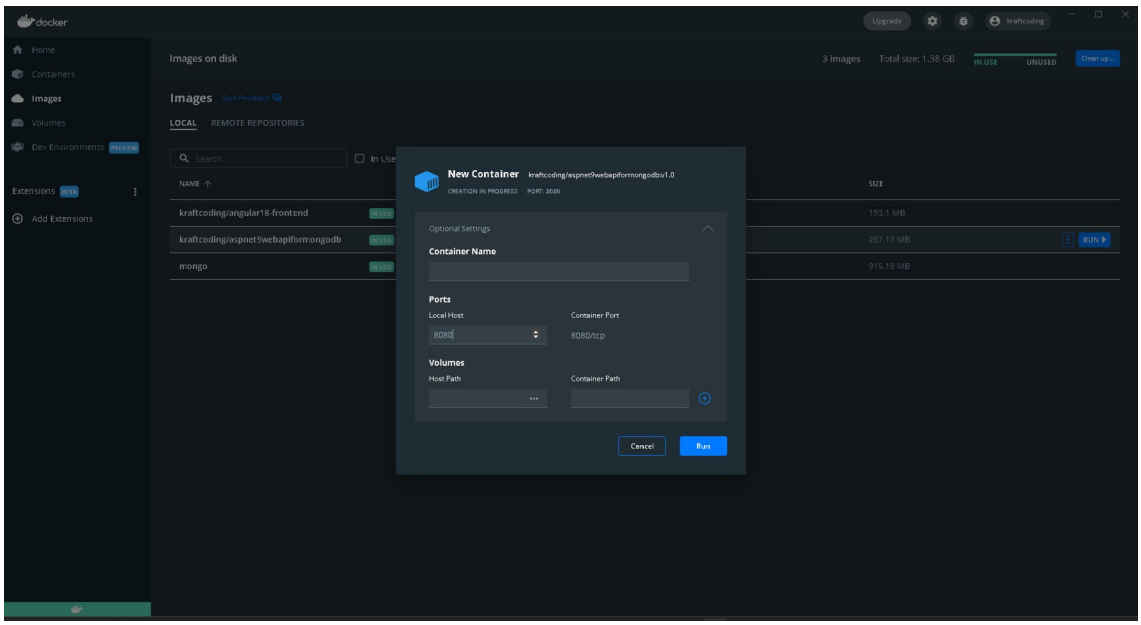
```
docker build -t kraftcoding/aspnet9webapiformongodb:v1.0 .
```



This will create the image in Docker:



Click run to create the container and access the Web API:



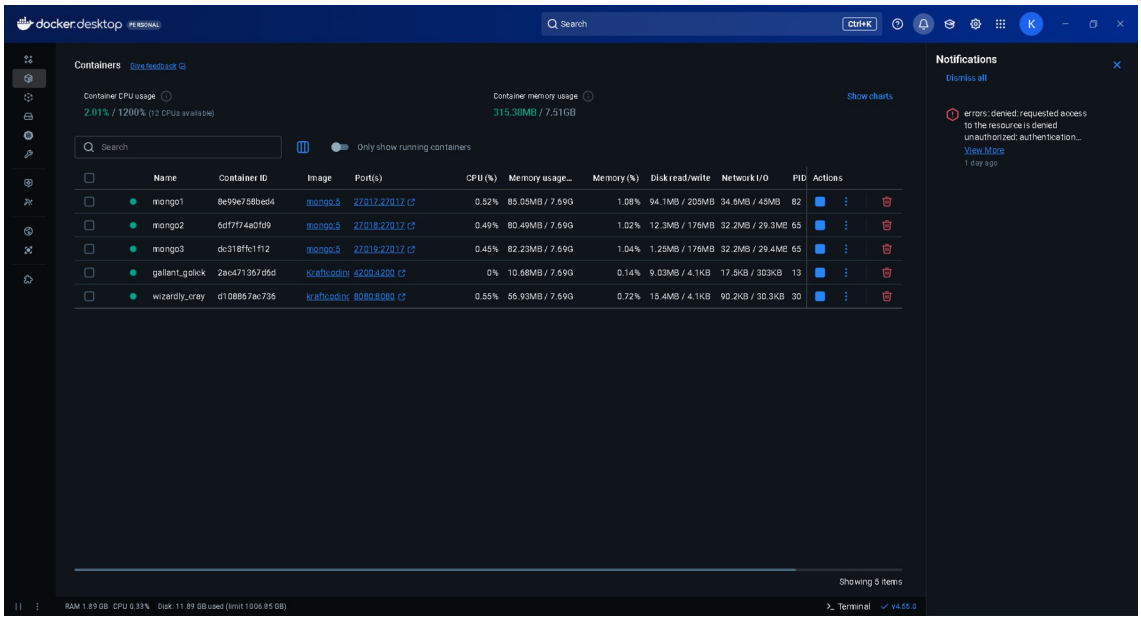
Deployment

To deploy the solution first add the images and deploy the containers for the angular application and the Web API as it has been detailed in the correspondent document sections (RQ3).

Notice that the connection strings in docker internally changes so it must be uncommented like this, both in program.cs and appsettings.json:

```
mongodb://host.docker.internal:27017/?
directConnection=true&serverSelectionTimeoutMS=6000&appName=mongosh+2.5.10
```

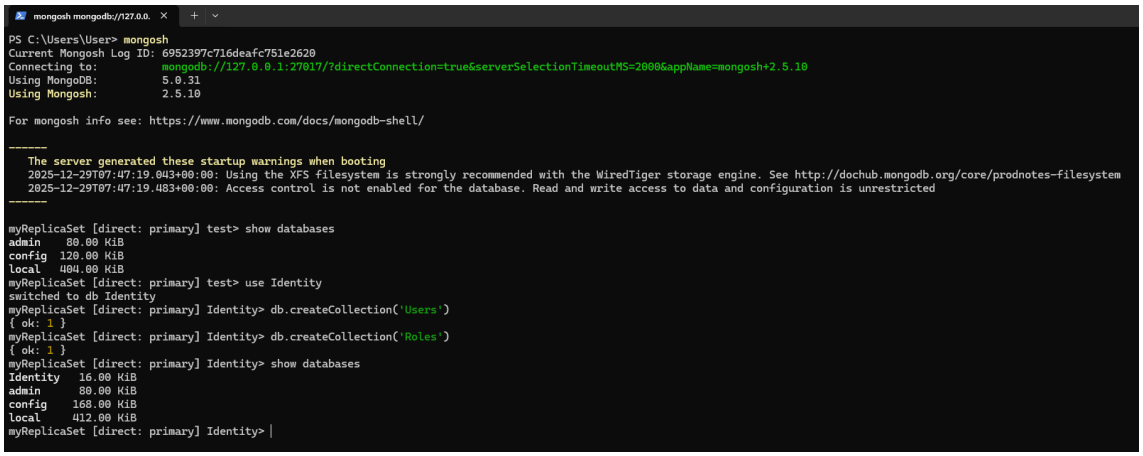
Once all is deployed to Docker it should look like this.



Follow the next steps to add the admin user and initial data to data base.

1. Execute mongosh then launch the next commands

   ```
   use Identity
   db.createCollection('Users')
   db.createCollection('Roles')
   ```



2. Open MongoDB Compass and connect to the cluster. We can find the connection string in the “Docker Custer Setup” section of this document.
3. Import initial data with the admin user and roles from Identity.Users.json and Identity.Roles.json files into the 'Identity' collection using MongoDB Compass.

   CREDENTIALS:

   username: admin
   email: admin@example.com
   pasword: Password123!

Software life cycle management

TO DO.

Annexes

Annexes go here...