

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

Requirements.....	3
Overview.....	4
Anonymous user screens.....	4
Authenticated user screens.....	6
Administrator user screens.....	9
Scope.....	11
RQ-01 Web API.....	12
Security.....	14
Securitised endpoints.....	14
Initial setup for Admin.....	14
Revoke refresh token.....	14
RQ-02 MongoDB Cluster.....	15
Setup.....	15
RQ-04 Angular App.....	19
Requirements.....	19
Angular Material UI.....	19
State Management.....	19
Security.....	20
Control route access with guards.....	20
Role based view.....	21
Run the App.....	22
RQ-03 Docker.....	23
Containerise an Angular Application.....	23
Containerise a .NET Application.....	26
Deployment.....	28
Software life cycle management.....	30
Annexes.....	31

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

Requirements

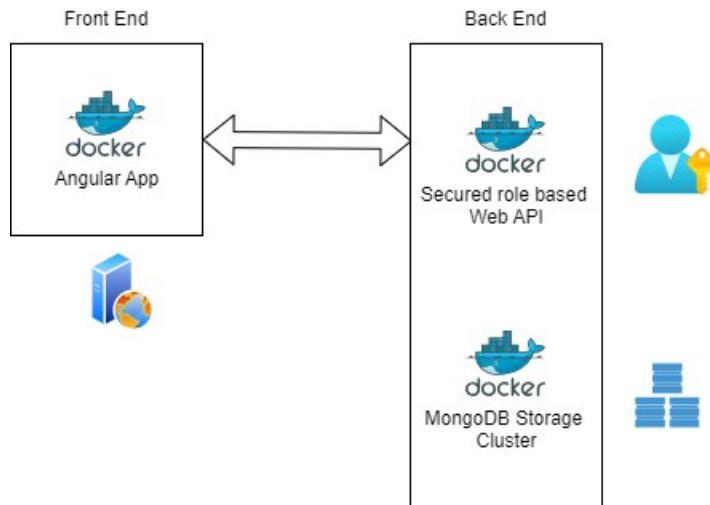
The next table shows the requirements for the solution described.

Requirement	Description
RQ-001 Web API	Back-end web API and all 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 prepermissions 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

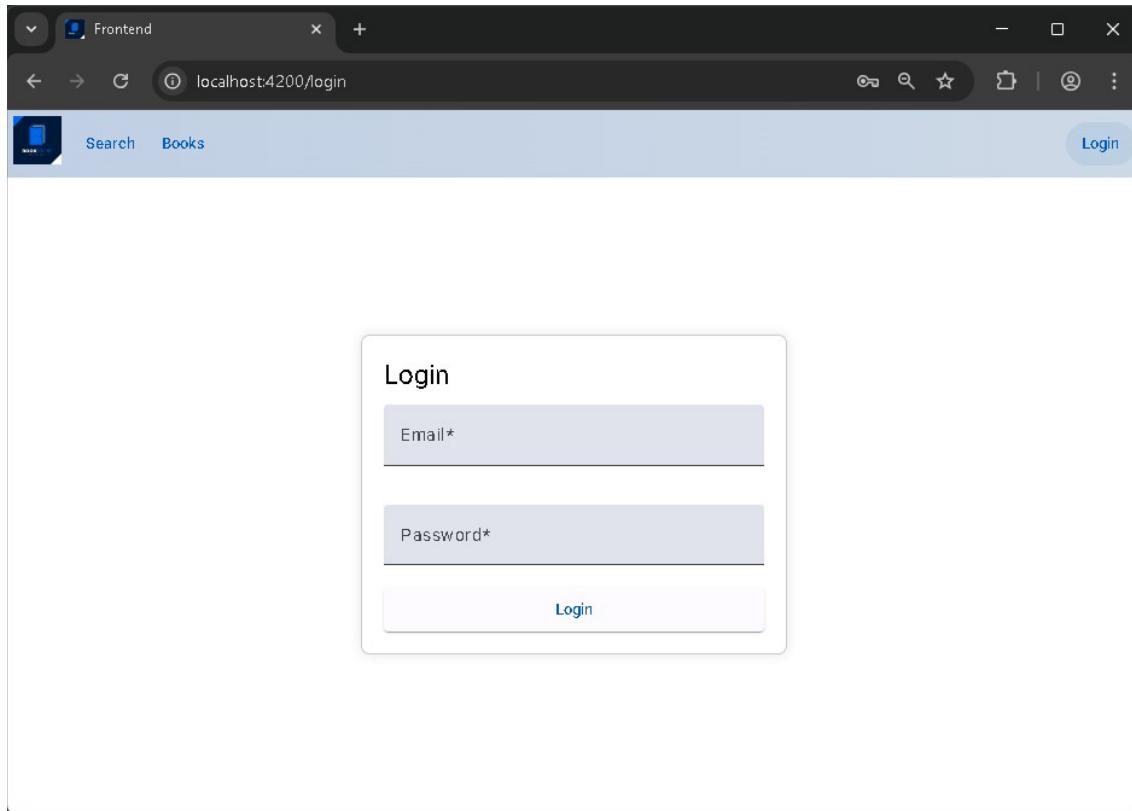
localhost:4200/list-books

Search Books Login

Books

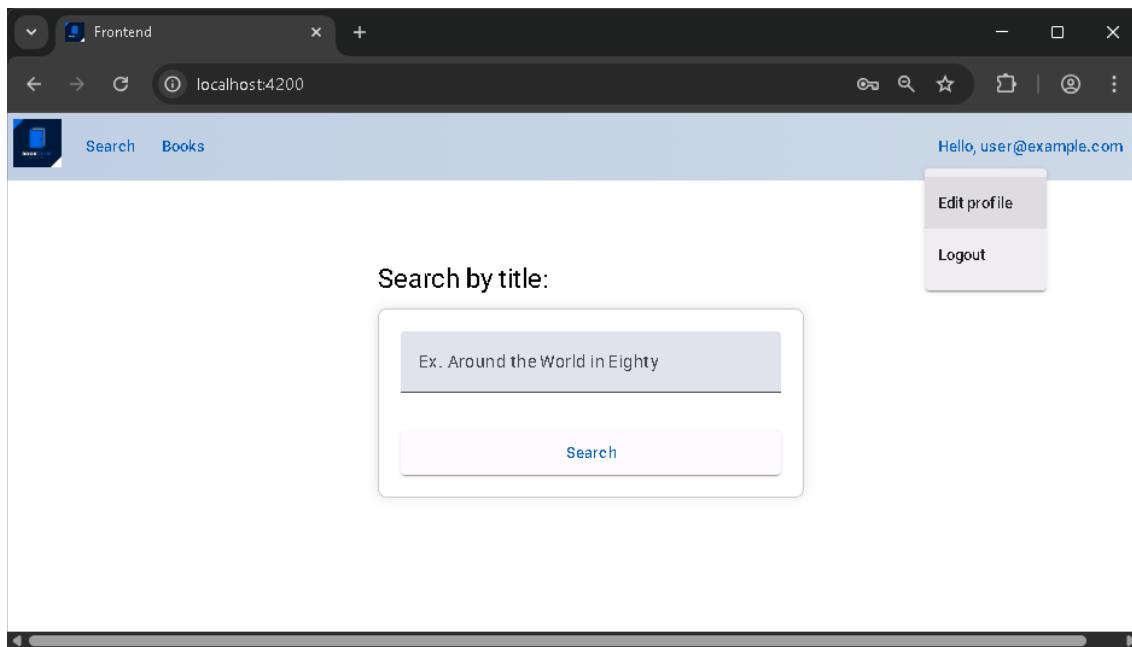
Title	Authors	Category	Published
Journey to the Center of the Earth	Jules Verne	Novel	2025-12-03T08:00:00Z
Twenty Thousand Leagues Under the Seas	Jules Verne	Novel	2025-12-04T08:00:00Z
The Mysterious Island	Jules Verne	Novel	2025-12-05T08:00:00Z
From the Earth to the Moon	Jules Verne	Novel	2025-12-06T08:00:00Z
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

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

Hello, user@example.com

Books

Title	Authors	Category	Published	Modified	Ids		
Journey to the Center of the Earth	Jules Verne	Novel	2025-12-08T08:00:00Z	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 < >

A screenshot of a web browser window titled "Frontend". The address bar shows "localhost:4200/book-details/69550e7d023add36f5a3f04a". The page header includes a logo, a search bar, and a user greeting "Hello, user@example.com". The main content area is titled "Books details" and displays the following book information:

- 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

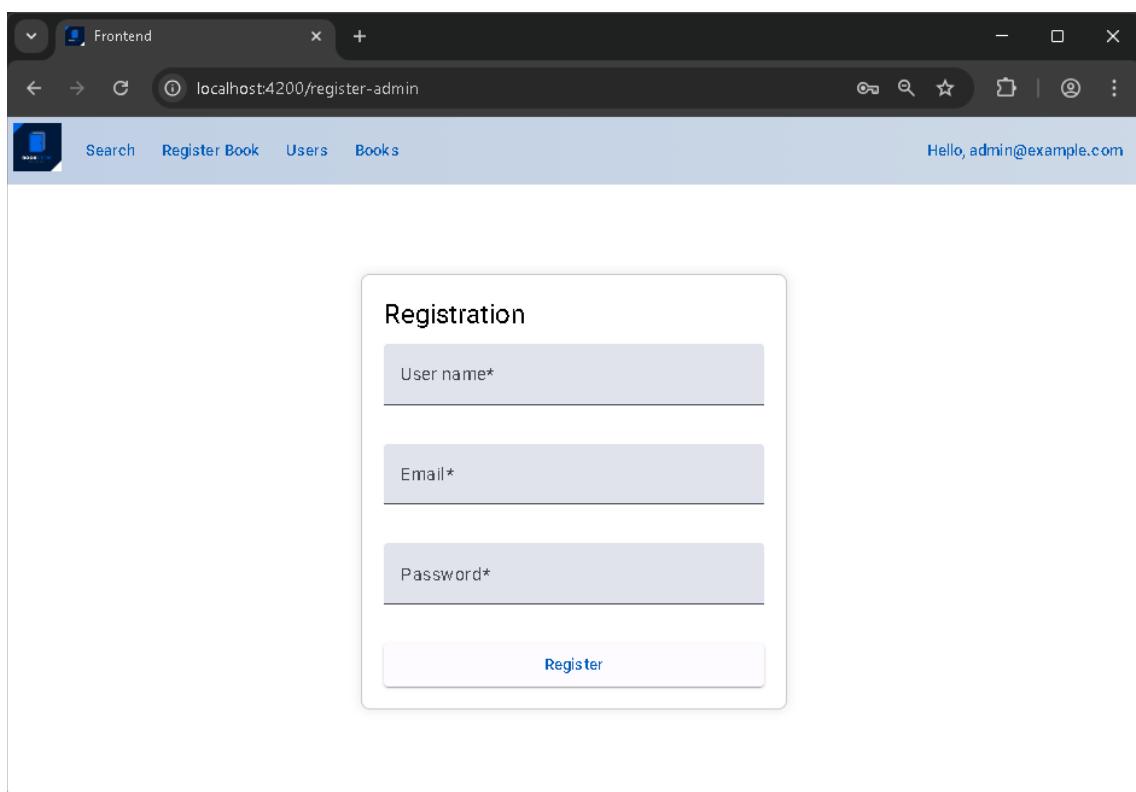
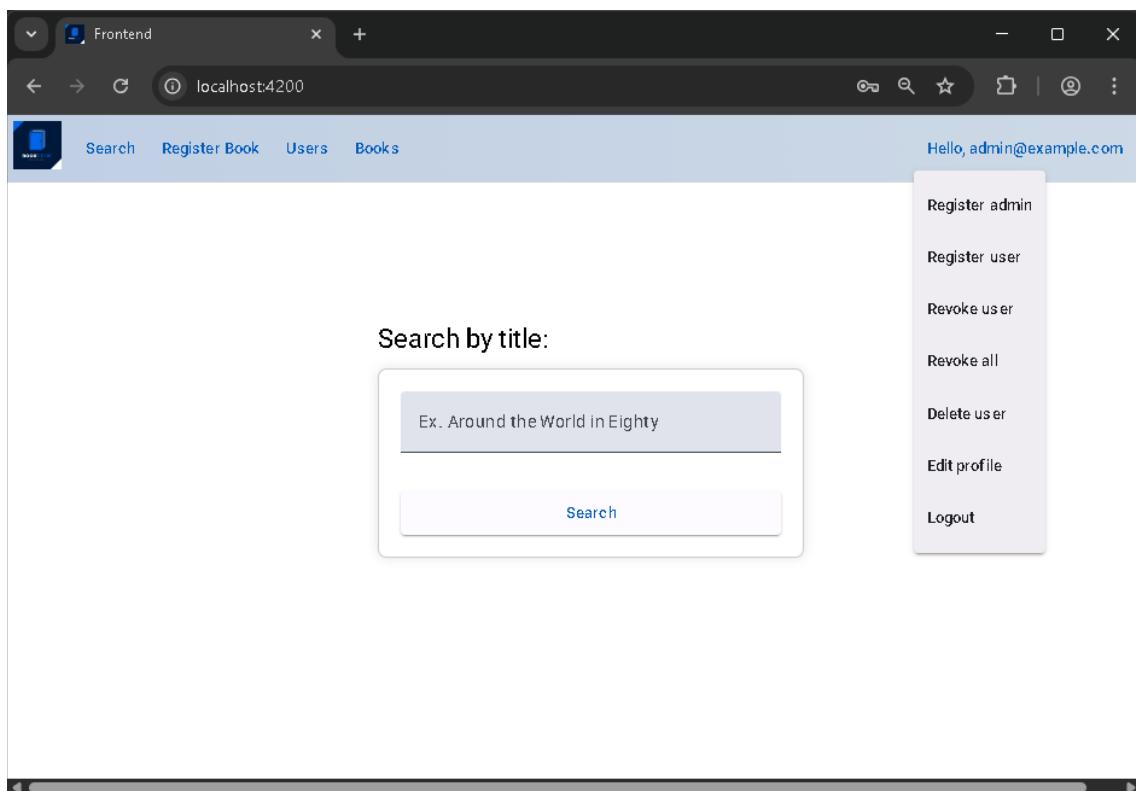
A screenshot of a web browser window titled "Frontend". The address bar shows "localhost:4200/edit-book/69550e7d023add36f5a3f04a". The page header includes a logo, a search bar, and a user greeting "Hello, user@example.com". The main content area is titled "Edit book" and contains the following form fields:

Title*	Journey to the Center of the Earth
Authors	Jules Verne
Ids	123456787
Category	Novel
Series	0
Published	12/2/2025 <input type="button" value="Calendar"/>
Publisher	World MEdia
Languages	ENG
Tags	Fantasy
Formats	EPUB
Modified	2025-12-31

At the bottom right of the form is a "Save" button.

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

Calendar icon

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.

The screenshot shows the Swagger UI interface for an ASP.NET 9 Web API. The title bar reads "Swagger UI" and the address bar shows "localhost:5192/swagger/index.html". The main content area is titled "Authenticate" and lists various API endpoints:

- POST /api/Authenticate/registerUser
- PUT /api/Authenticate/updateUser
- POST /api/Authenticate/registerAdmin
- POST /api/Authenticate/login
- POST /api/Authenticate/refresh-token
- POST /api/Authenticate/revoke/{email}
- POST /api/Authenticate/delete/{email}
- GET /api/Authenticate/getUser/{email}
- GET /api/Authenticate/getUsers
- POST /api/Authenticate/revoke-all

Here the books API.

The screenshot shows the Swagger UI interface for a .NET Core application. The URL is `localhost:5192/swagger/index.html`. The interface is organized into sections:

- Books**:
 - GET /api/Books
 - POST /api/Books
 - GET /api/Books/{id}
 - PUT /api/Books/{id}
 - DELETE /api/Books/{id} (highlighted with a red border)
- Version**:
 - GET /api/Version
- WeatherForecast**:
 - GET /GetWeatherForecastWithNoPermissions
 - GET /GetWeatherForecastAdmin
 - GET /GetWeatherForecastUser
- Schemas**:
 - ApplicationUser >
 - Book >

And the schemas.

The screenshot shows the Swagger UI interface running at `localhost:5192/swagger/index.html`. The top navigation bar includes tabs for 'Swagger UI' and 'API'. Below the address bar, there are buttons for back, forward, and search, along with a refresh icon.

The main content area displays two API endpoints:

- GET /GetWeatherForecastAdmin**
- GET /GetWeatherForecastUser**

Below the endpoints is a section titled "Schemas" which lists various data models:

- ApplicationUser >
- Book >
- LoginModel >
- MongoClaim >
- Token >
- TokenModel >
- User >
- UserLoginInfo >
- WeatherForecast >

Security

The security aspects to take in account are described below.

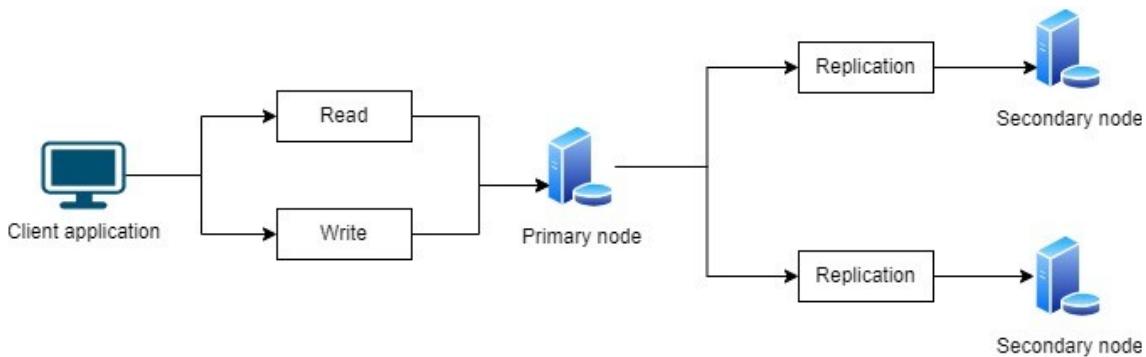
Securitised endpoints

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.

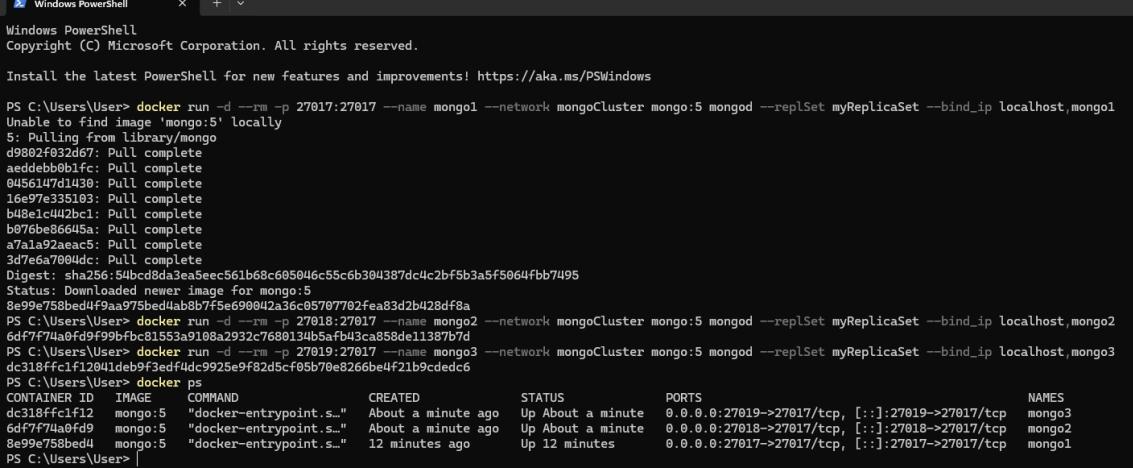
```
docker run -d --rm -p 27017:27017 --name mongo1 --network mongoCluster
mongo:latest mongod --replSet myReplicaSet --bind_ip localhost,mongo1
```

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 --replicaSet myReplicaSet --bind_ip localhost,mongo2
```

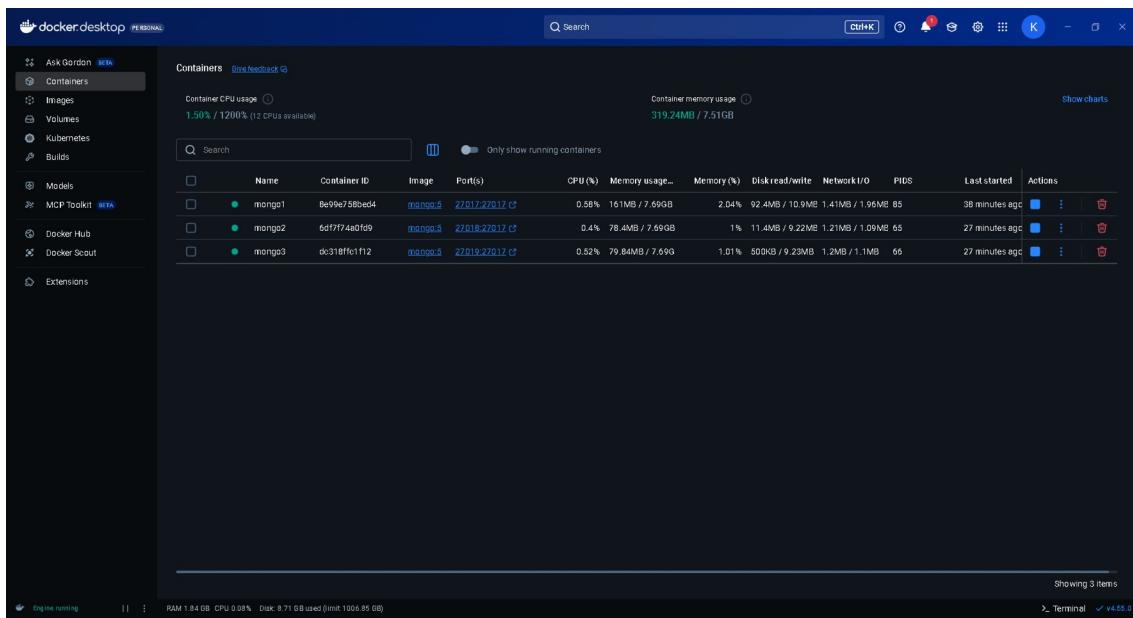
```
docker run -d --rm -p 27019:27017 --name mongo3 --network mongoCluster  
mongo:5 mongod --replicaSet myReplicaSet --bind_ip localhost,mongo3
```

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



```
PS C:\Users\User> docker run -d --rm -p 27017:27017 --name mongo1 --network mongoCluster mongo:5 mongod --replicaSet myReplicaSet --bind_ip localhost,mongo1  
Unable to find image 'mongo:5' locally  
5: Pulling from library/mongo  
d9802f932d67: Pull complete  
aeddeb0b1fc: Pull complete  
0456147d1438: Pull complete  
16e97e335103: Pull complete  
b48e1c442bc1: Pull complete  
b076be86645a: Pull complete  
a7a1a92aeac5: Pull complete  
3d7e6a7094dc: Pull complete  
Digest: sha256:54bcd8da3e5ee561b68c695946c55c6b304387dc4c2bf5b3a5f5064fb7495  
Status: Downloaded newer image for mongo:5  
5e99e758bed4f9aa975bed4ab0b7f5e699042a36c05707702fea83d2b428df8a  
PS C:\Users\User> docker run -d --rm -p 27018:27017 --name mongo2 --network mongoCluster mongo:5 mongod --replicaSet myReplicaSet --bind_ip localhost,mongo2  
6df7f74af0fd9f996fb81553a9108a2932c7680134b5afbf43ca858de11387b7d  
PS C:\Users\User> docker run -d --rm -p 27019:27017 --name mongo3 --network mongoCluster mongo:5 mongod --replicaSet myReplicaSet --bind_ip localhost,mongo3  
dc318ffcf1f12041debf9f3edf4dc9925e9f82d5cf05b70e8266be4f21b9cdedc6  
PS C:\Users\User> docker ps  
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES  
dc318ffcf1f12 mongo:5 "docker-entrypoint.s..." About a minute ago Up About a minute 0.0.0.0:27019->27017/tcp, [::]:27019->27017/tcp mongo3  
6df7f74af0fd9 mongo:5 "docker-entrypoint.s..." About a minute ago Up About a minute 0.0.0.0:27018->27017/tcp, [::]:27018->27017/tcp mongo2  
5e99e758bed4 mongo:5 "docker-entrypoint.s..." 12 minutes ago Up 12 minutes 0.0.0.0:27017->27017/tcp, [::]:27017->27017/tcp mongo1  
PS C:\Users\User> |
```

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.

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

What's next:
  Try Docker Debug for seamless, persistent debugging tools in any container or image → docker debug mongo1
  Learn more at https://docs.docker.com/go/debug-cli/
PS C:\Users\User> |
```

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 command, we will see that by default we will be connected to the primary node.

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.5.10
PS C:\Users\User> mongosh
Current Mongosh Log ID: 6952397c716deafc751e2620
Connecting to:      mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.5.10
Using MongoDB:     5.0.31
Using Mongosh:     2.5.10
For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/
The server generated these startup warnings when booting
2025-12-29T07:47:19.043+00:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongodb.org/core/prodnotes-filesystem
2025-12-29T07:47:19.483+00:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
myReplicaSet [direct: primary] test> |
```

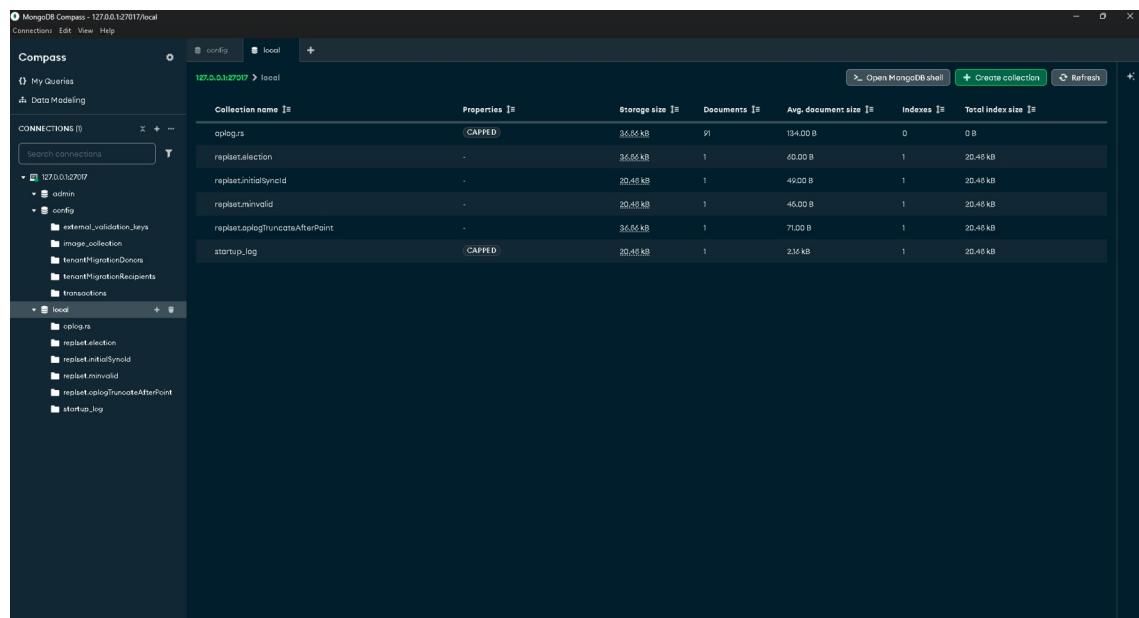
As we can notice, the connection string to test from visual studio is:

```
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.



The screenshot shows the MongoDB Compass interface connected to the 'config' database at '127.0.0.1:27017'. The left sidebar lists connections, queries, and data modeling tools. The main area displays the 'local' database, which contains several collections related to a replica set configuration. A table provides details for each collection:

Collection name	Properties	Storage size	Documents	Avg. document size	Indexes	Total index size
clog.rs	CAPPED	36.85 kB	91	134.00 B	0	0 B
repset.election	-	36.85 kB	1	40.00 B	1	20.45 kB
repset.initialSyncId	-	20.45 kB	1	49.00 B	1	20.45 kB
repset.invalid	-	20.45 kB	1	46.00 B	1	20.45 kB
repset.oplogTruncateAfterPoint	-	36.85 kB	1	71.00 B	1	20.45 kB
startup.log	CAPPED	20.45 kB	1	236 kB	1	20.45 kB

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.

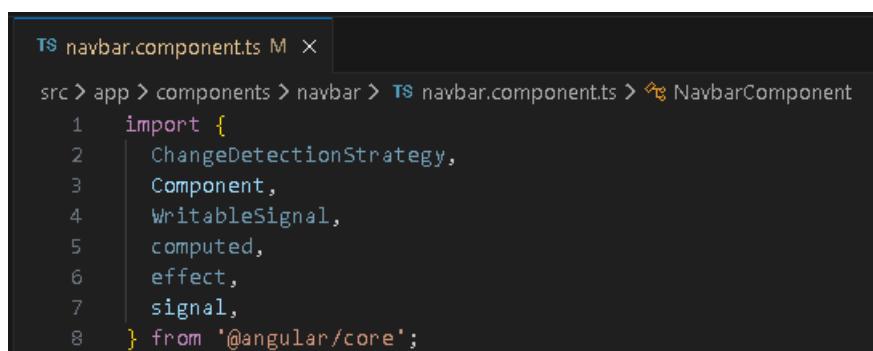
Angular Material UI

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.



```
TS navbar.component.ts M X
src > app > components > navbar > TS navbar.component.ts > NavbarComponent
1 import {
2   ChangeDetectionStrategy,
3   Component,
4   WritableSignal,
5   computed,
6   effect,
7   signal,
8 } from '@angular/core';
```

And here an example of implementation.

```
TS navbar.component.ts M ×
src > app > components > navbar > TS navbar.component.ts > NavbarComponent
  26   export class NavbarComponent {
  27     ...
  28     ngOnInit() {
  29       this.username$.subscribe((response) => {
  30         this.username = signal<string>(response || '');
  31       });
  32       this.username = signal<string>(
  33         localStorage.getItem('Template_email') || ''
  34       );
  35     }
  36   }
```

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.

```
TS app.routes.ts M × TS auth.guard.ts M
src > app > TS 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 TS auth.guard.ts M X
src > app > guards > TS auth.guard.ts > ...
1 import { CanActivateFn } from '@angular/router';
2 import { UserService } from '../services/user.service';
3 import { inject } from '@angular/core';
4
5 export const authGuard: CanActivateFn = (route, state) => {
6   const authService = inject(UserService);
7   return authService.isAuthenticated();
8 };
9
```

And the implementation of the function in the service.

```
TS app.routes.ts M TS auth.guard.ts M TS user.service.ts M X
src > app > services > TS user.service.ts > UserService > isAuthenticated
11 export class UserService {
41   isAuthenticated(): [
42     if(this.isAuthenticatedSubject.value){
43       return true;
44     }
45     else{
46       this.router.navigate(['/login']);
47       return false;
48     }
49   ]
50 }
```

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.

```
25 export class NavbarComponent {
26   isLoggedIn$ = this.userService.isAuthenticated$;
27   isAdmin$ = this.userService.isAdmin$;
28   username$ = this.userService.username$;
29   username: any;
```

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

The screenshot shows a code editor with two tabs: 'navbar.component.ts' and 'navbar.component.html'. The 'navbar.component.html' tab is active, displaying the following HTML code:

```
src > app > components > navbar > navbar.component.html > nav
  1  <nav>
  2    <div class="left-side">
  3      <div class="logo" (click)="homeRedirect()">
  4        </div>
  5        <button mat-button color="primary" (click)="homeRedirect()">Home</button>
  6        <button mat-button color="primary">Users</button>
  7      </div>
  8      <div class="right-side">
  9        @if (isLoggedin$ | async){
10          <button mat-button color="primary" [matMenuTriggerFor]="menu">Hello, {{username()}}</button>
11          <mat-menu #menu="matMenu">
12            @if (isAdmin$ | async){
13              <button mat-menu-item color="primary" (click)="registerAdmin()">Register admin</button>
14              <button mat-menu-item color="primary" (click)="registerUser()">Register user</button>
15              <button mat-menu-item color="primary" (click)="revokeUser()">Revoke user</button>
16              <button mat-menu-item color="primary" (click)="revokeAll()">Revoke all</button>
17              <button mat-menu-item color="primary" (click)="deleteUser()">Delete user</button>
18            }
19            <button mat-menu-item color="primary" (click)="editUser()">Edit profile</button>
20            <button mat-menu-item color="primary" (click)="logout()">Logout</button>
21          </mat-menu>
22        } @else {
23          <button mat-button color="primary" (click)="loginRedirect()">Login</button>
24        }
25      </div>
26    </div>
27  </nav>
```

Run the App

`npm install`

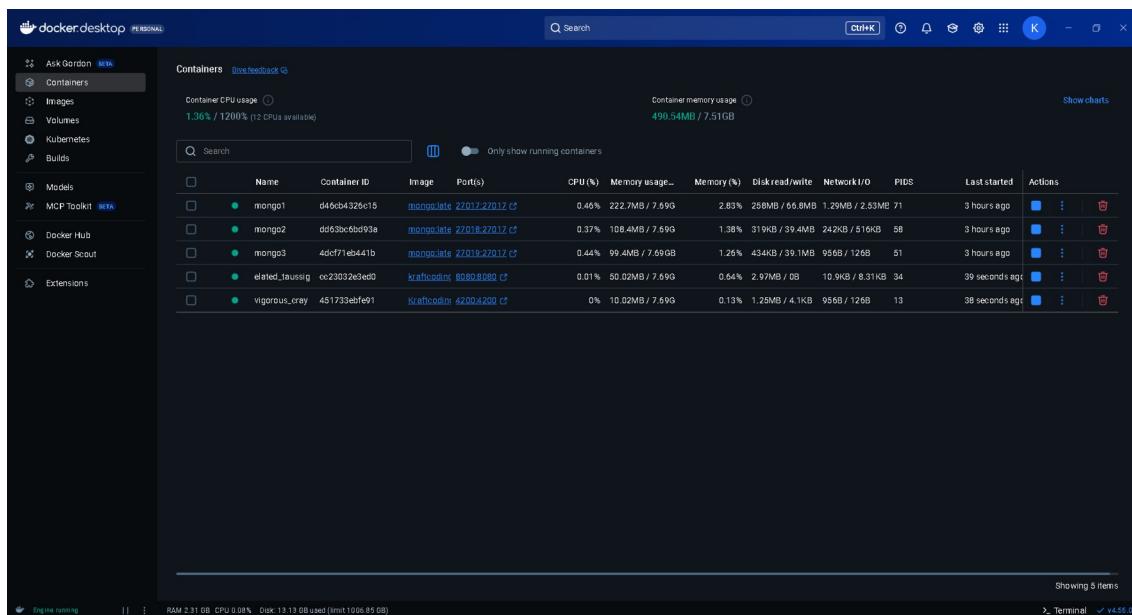
`npm start`

`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      PID(S)
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
```

```
Developer PowerShell
+ Developer PowerShell | ⌂ | ⌂ | ⌂
PS D:\SRC\PROJECTS\BookStore-v1.x\Angular18-Frontend> docker init

Welcome to the Docker Init CLI!

This utility will walk you through creating the following files with sensible defaults for your project:
- .dockerignore
- Dockerfile
- compose.yaml
- README.Docker.md

Let's get started!

? What application platform does your project use? Node
? What version of Node do you want to use? (22.12.0)

? What version of Node do you want to use? 22.12.0
? Which package manager do you want to use? npm
? Do you want to run "npm run build" before starting your server? Yes
? What directory is your build output to? (comma-separate if multiple) dist

? What directory is your build output to? (comma-separate if multiple) dist
? What command do you want to use to start the app? [tab for suggestions] (npm start)

? What command do you want to use to start the app? npm start
? What port does your server listen on? 4200

Created → .dockerignore
Created → Dockerfile
Created → compose.yaml
Created → README.Docker.md

→ Your Docker files are ready!
Review your Docker files and tailor them to your application.
Consult README.Docker.md for information about using the generated files.

What's next?
Start your application by running → docker compose up --build
Your application will be available at http://localhost:4200
PS D:\SRC\PROJECTS\BookStore-v1.x\Angular18-Frontend>
```

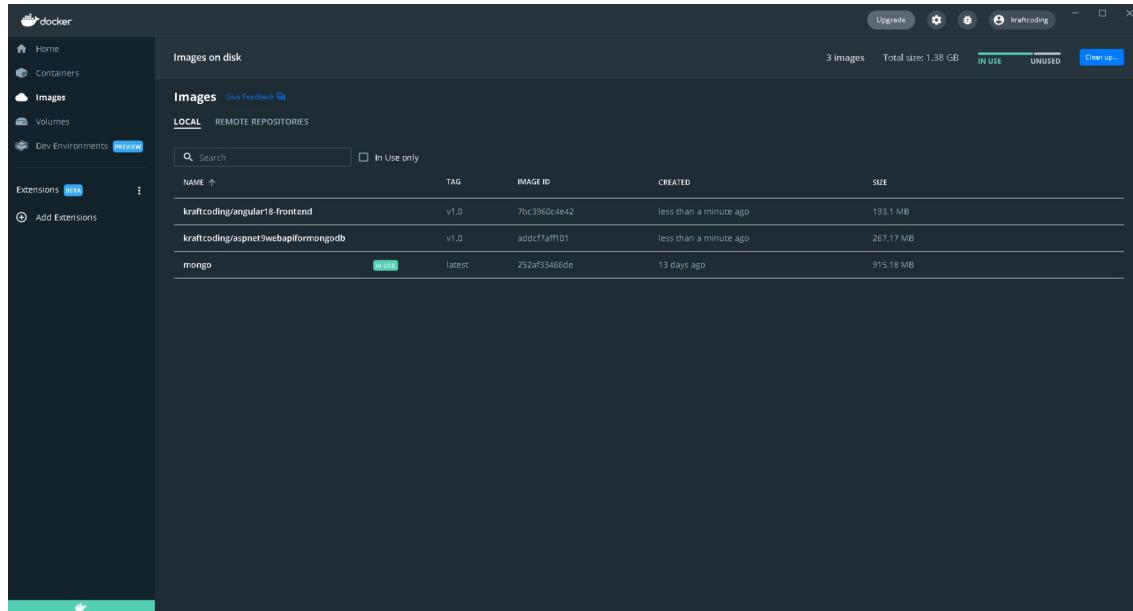
Then we will need to customise de Dockerfile and create nginx.config before building the image.

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

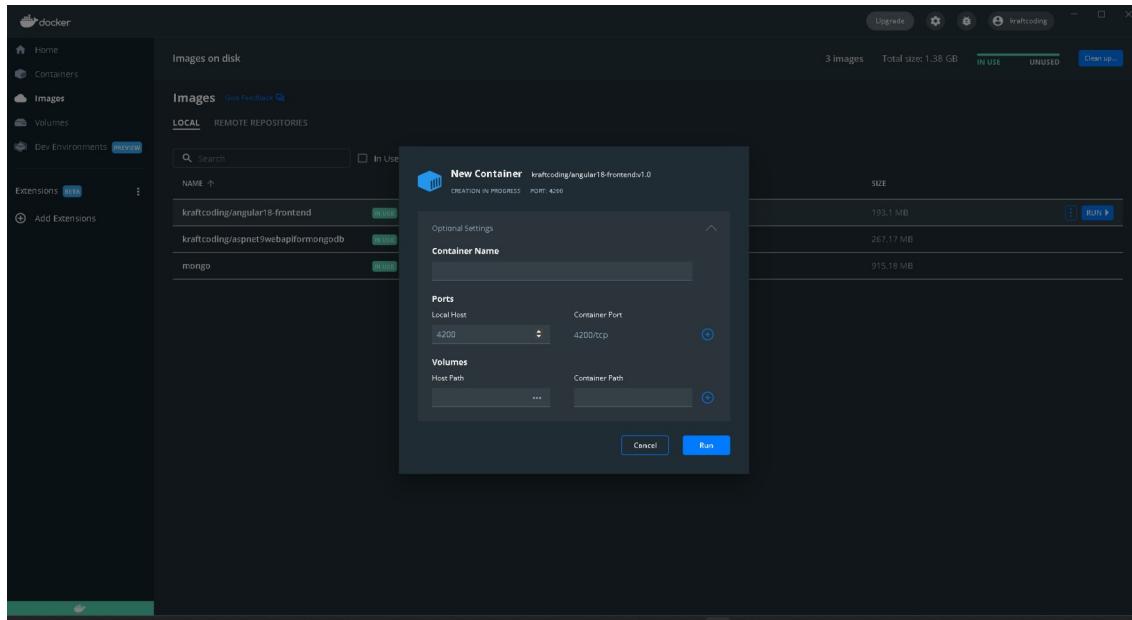
The screenshot shows a Windows desktop with several application windows open, illustrating a multi-environment development setup:

- File Explorer**: Shows the project structure for "BookStore-v1.x".
- Docker command.txt**: A text editor showing Docker command lines.
- Setup.txt**: A text editor showing configuration or deployment scripts.
- home.components.ts**, **BooksService.cs**, **AuthenticateController.cs**, **Book.cs**, **TextUsers.txt**, **BooksController.cs**: Code editors for various .NET and Angular files.
- Developer PowerShell**: A terminal window showing the output of a "docker build" command for the "kraftcoding/angular18-frontend:v1.0" image.
- Solution Explorer**: Shows the solution structure with projects like "MongoDB-Cluster-Deployment" and "Angular18-Frontend".
- Search Solution Explorer**: A search bar for the solution explorer.
- Task List**: Shows build errors and warnings.
- Output**: Shows logs from the "Developer PowerShell" and "Docker command.txt" windows.
- Git Changes**: Shows recent changes in the repository.

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

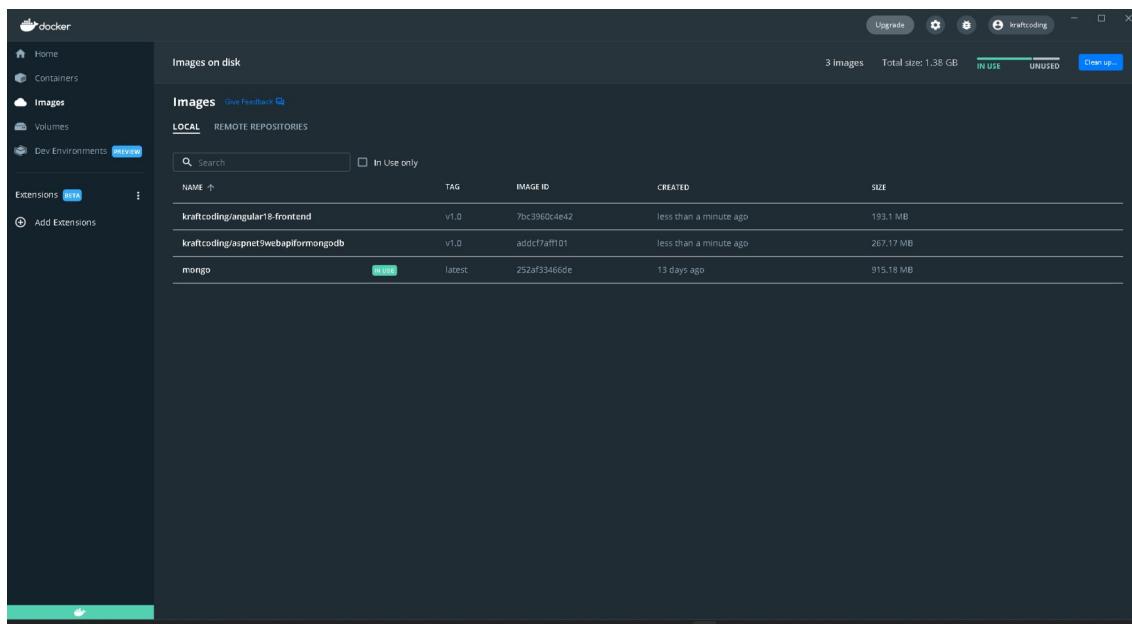
Docker provides an interactive CLI tool called docker init that helps scaffold the necessary configuration files for containerising your application.

Setup

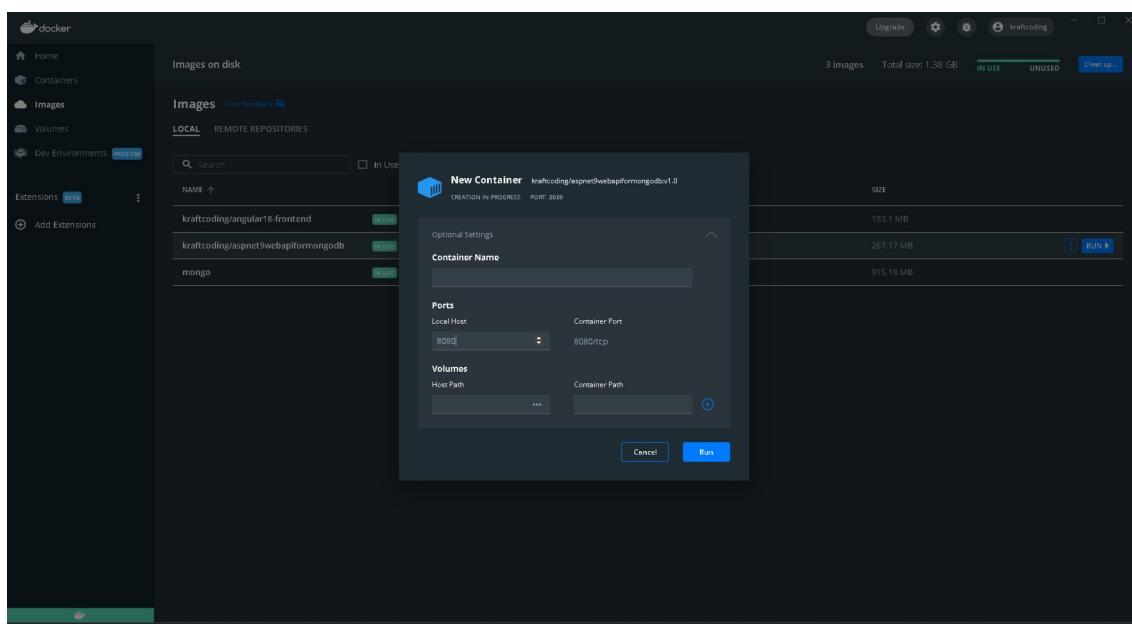
To create the needed files execute:

```
docker build -t kraftcoding/aspnet9webapiformmongodb:v1.0 .
```

This will create the image in Docker:



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



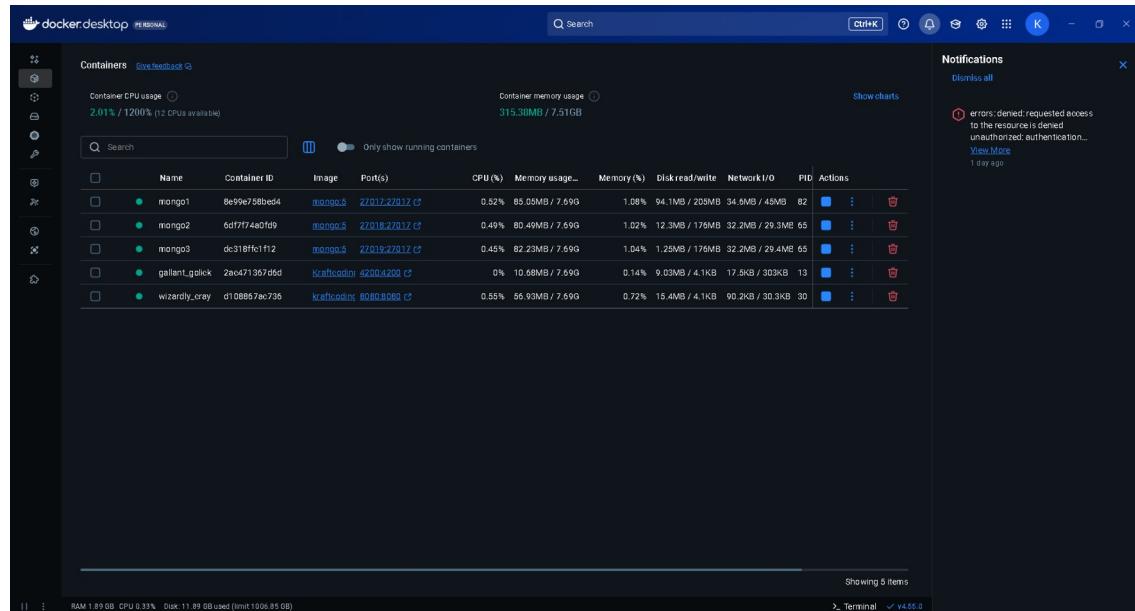
Deployment

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')
```

```
PS C:\Users\User> mongosh  
Current Mongosh Log ID: 6952397c716deafc751e2620  
Connecting to: mongod://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=20000&appName=mongosh+2.5.10  
Using MongoDB: 5.0.31  
Using Mongosh: 2.5.10  
  
For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/  
  
The server generated these startup warnings when booting  
2025-12-29T07:47:19.803Z+00:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongodb.org/core/prodnotes-filesystem  
2025-12-29T07:47:19.483+00:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted  
  
myReplicaSet [direct: primary] test> show databases  
admin 80.00 KiB  
config 120.00 KiB  
local 404.00 KiB  
myReplicaSet [direct: primary] test> use Identity  
switched to db Identity  
myReplicaSet [direct: primary] Identity> db.createCollection('Users')  
[ ok: 1 ]  
myReplicaSet [direct: primary] Identity> db.createCollection('Roles')  
[ ok: 1 ]  
myReplicaSet [direct: primary] Identity> show databases  
Identity 16.00 KiB  
admin 80.00 KiB  
config 168.00 KiB  
local 412.00 KiB  
myReplicaSet [direct: primary] Identity> |
```

2. Open MongoDB Compass and connect to the cluster. We can find the connection string in the “Docker Cluster 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  
password: Password123!
```

Software life cycle management

TO DO.

Annexes

Annexes go here...