Contents

[1. General Information 1](#_Toc177671410)

[2. Setup 1](#_Toc177671411)

[3. Sprint 1 1](#_Toc177671412)

[3.1 IDE Setup 1](#_Toc177671413)

[3.2 Docker Containerization 2](#_Toc177671414)

[4. Sprint 2 2](#_Toc177671415)

[4.2 Nginx 2](#_Toc177671416)

[4.3 Project Structure Adjustments 2](#_Toc177671417)

[5. Sprint 3 3](#_Toc177671418)

[5.1 Entities 3](#_Toc177671419)

[5.2 Postgres 3](#_Toc177671420)

[5.3 Postgres Container Issues and Solutions 3](#_Toc177671421)

[5.4 DTOs and Mapping 3](#_Toc177671422)

[5.5 Unit Tests for DTOs and Mapping 3](#_Toc177671423)

# 1. General Information

# 2. Setup

Before starting with the first Sprint, following things had to be checked:

* IDE like Visual Studio or JetBrains Rider installed?
* Docker Installed?
* Git installed?

We both decided to use JetBrains Rider, since we are very familiar with their software and simply prefer using it over Visual Studio.

# 3. Sprint 1

## 3.1 IDE Setup

When creating the project, we opted for using the APS.NET Core Web API template in Rider, which provides a swagger UI out of the box. We ditched the swagger UI and just printed sample documents for Sprint 1.

## 3.2 Docker Containerization

After creating the Dockerfile for this application, the command was used to build the docker image for the container:

*docker build -t document-management-system .*

‘document-management-system’ is the name of the image.

This command is used to run the docker container:

*docker run -d -p 8081:8081 --name dms\_container document-management-system*

*‘-p 8081:8081’*  lets the docker container run on port 8081 and also exposes it to port 8081, where it is being accessible in the browser. ‘dms\_container’ is the name of the container.

Using a ‘*docker-compose.yml’*  file makes coding with Docker a lot easier. Since you can just type ‘*docker-compose build’* and ‘*docker-compose up’* to practically build and run the containerized application.

# 4. Sprint 2

## 4.2 Nginx

## 4.3 Project Structure Adjustments

After taking a look at the sample code from our lecturer, we decided to apply their project structure to ours to some degree. We added following folders:

* **Controllers**: MVC architecture, Controllers are basically the URL
* **DTOs**: Data Transfer Objects, carries data between ‘processes’
* **Entities**: Model-classes for the data
* **Repositories** (later in DAL): Defines queriers following a pattern for the API

# 5. Sprint 3

## 5.1 Entities

Entities define the data structure / model and properties of objects used in the application. For example, ‘user’ consists of:

* (Id)
* Firstname
* Lastname
* Email

Validation rules for the properties of the entity classes were defined afterwards. In case of the ‘user’ entity, firstname, lastname and email are all required in order to pass the validation.

## 5.2 Postgres

In our *.yml* file we added the postgres container with following parameters:

* **User**: dms\_user
* **Password**: dms\_password
* **Database**: dms\_db

The connection string for the postgres connection with the application is stored in the *appsettings.json* file.

## 5.3 Postgres Container Issues and Solutions

Upon starting the container the first time, everything went well. After the initialization we changed the parameters to the ones specified above. We didn’t delete the volume for postgres, that’s why the initial parameters were still stored in postgres and our new user wasn’t created.

## 5.4 DTOs and Mapping

## 5.5 Unit Tests for DTOs and Mapping