Cloud Computing (IT 4090)
4th Year, 1st Semester

Individual Assignment

Create and deploy a simple application in Docker

Submitted to

Sri Lanka Institute of Information Technology

IT18540536

Lanerolle T.Y

In partial fulfillment of the requirements for the

Bachelor of Science Special Honors Degree in Information Technology

07-11-2021

Declaration

I certify that this report does not incorporate without acknowledgement, any material previously submitted for a degree or diploma in any university, and to the best of my knowledge and belief it does not contain any material previously published or written by another person, except where due reference is made in text.

Table of Content

Table of Figures

01 Background

This is a simple system that helps to work with Employees and Departments which Employee should related to. Application main includes these functions,

- Create New Employee
- Create New Department
- Update Employee Details
- Update Department Details
- View Employee Details
- View Department Details
- Remove Employee
- Remove Department

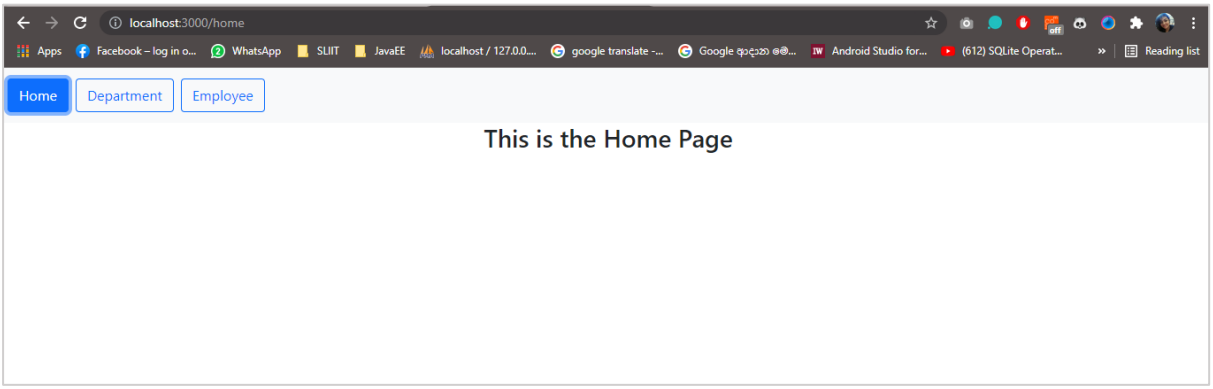

Figure 1.1: Home Page

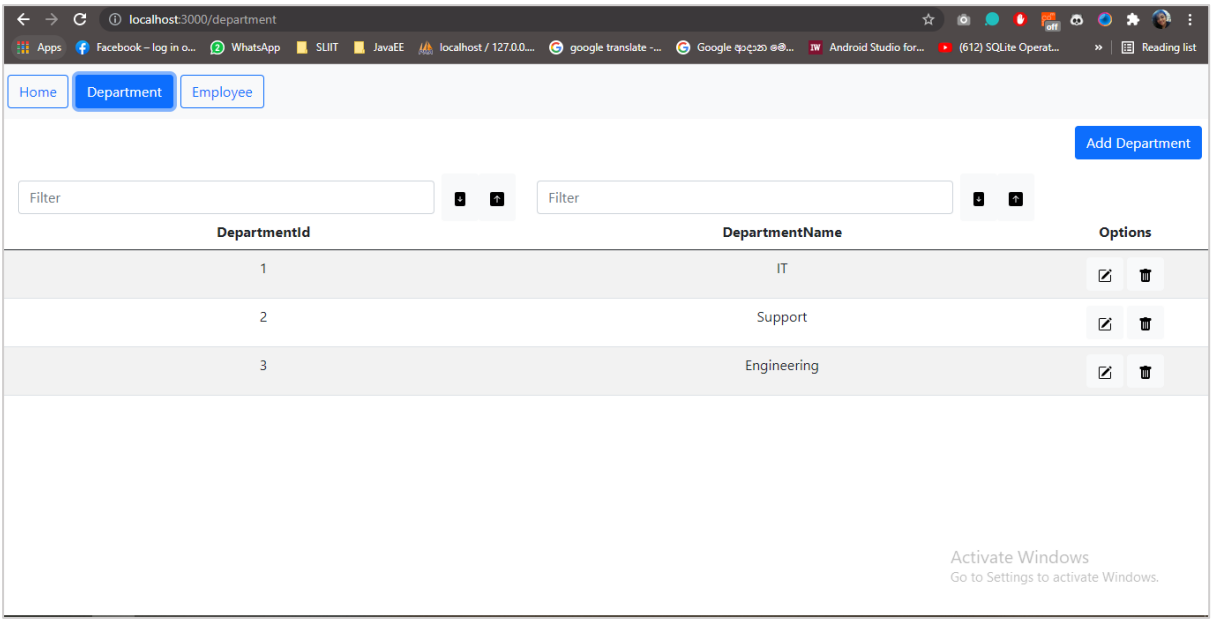

Figure 1.2: Department Details Page

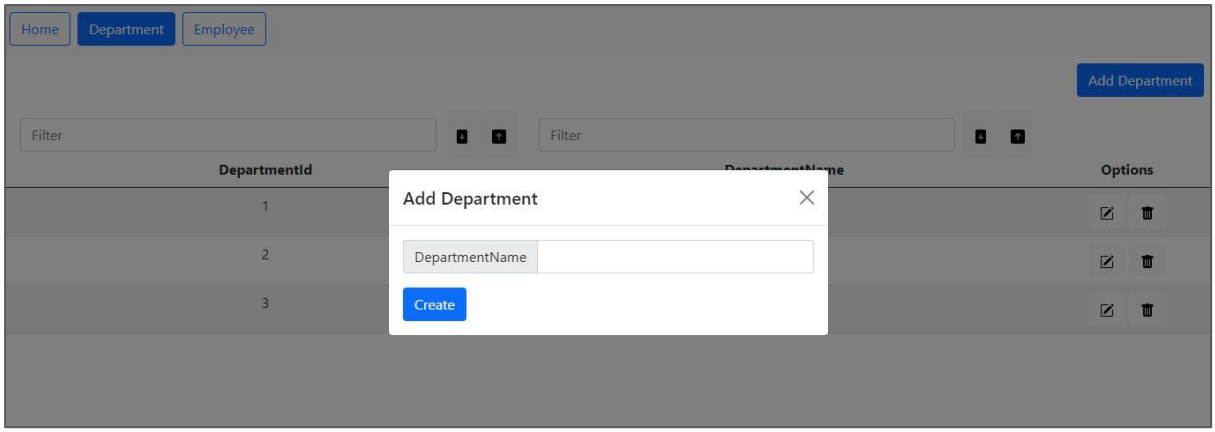

Figure 1.3: Add New Department Page

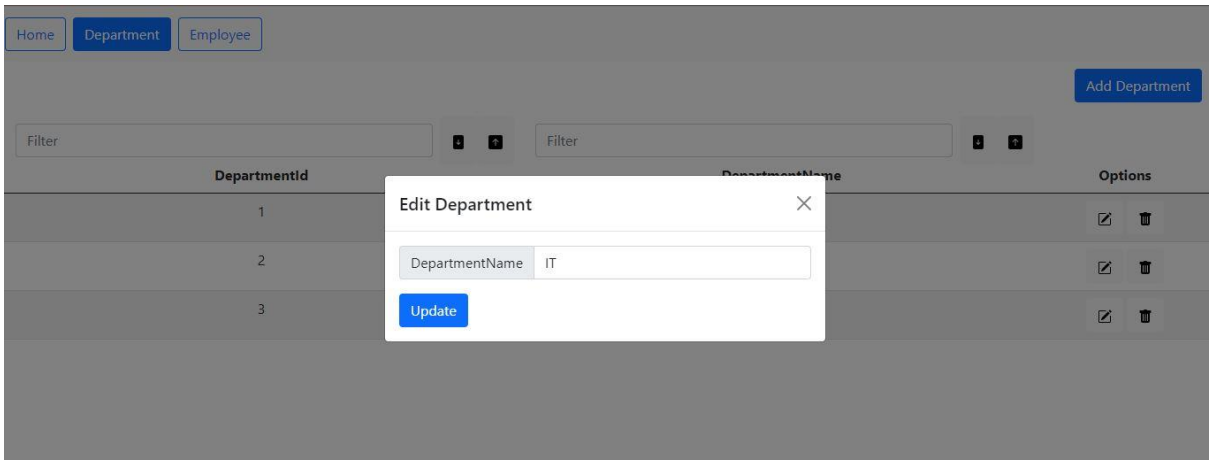

Figure 1.4: Edit Department Details Page

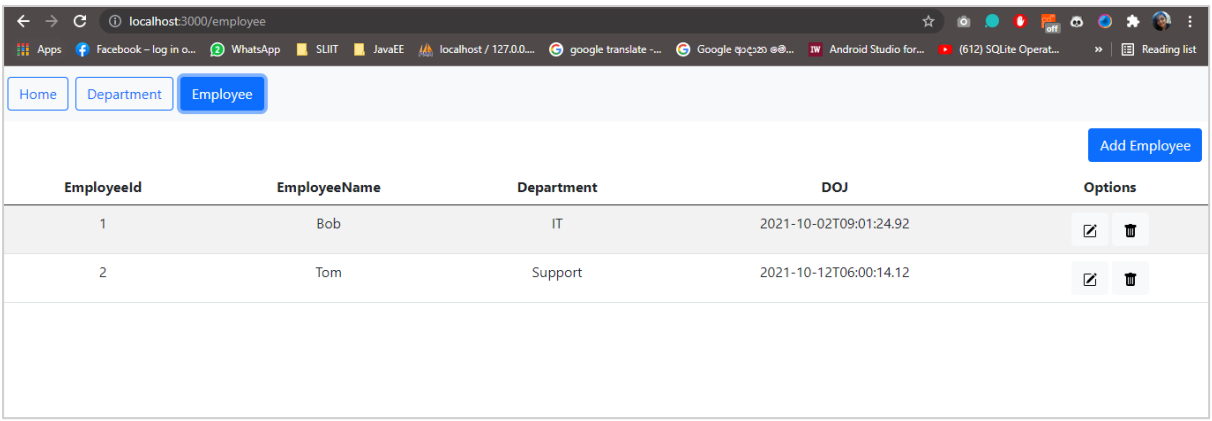

Figure 1.5: Employee Details Page

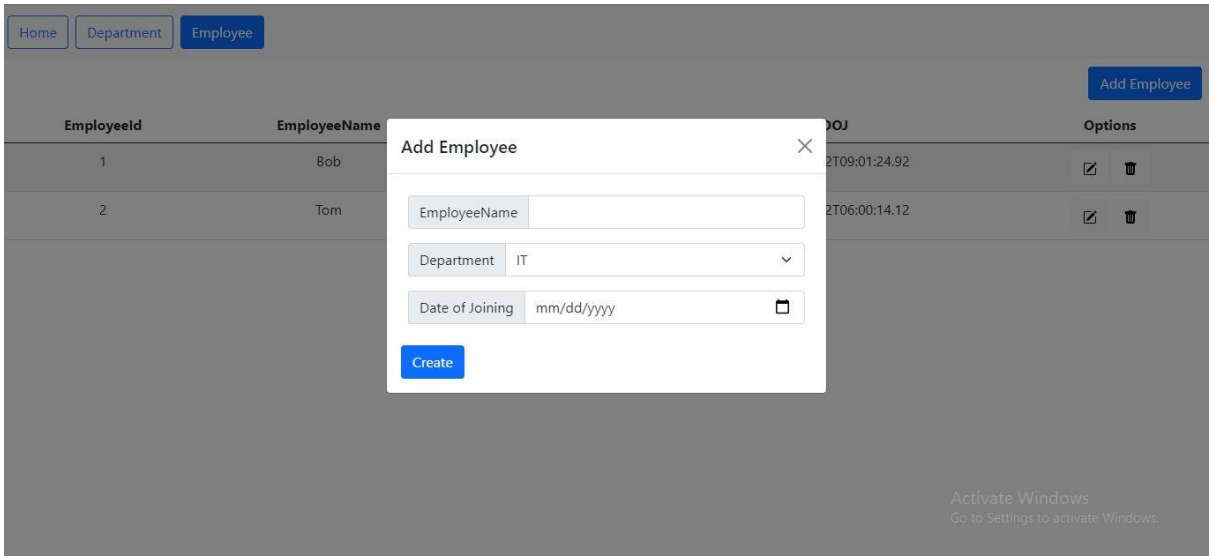

Figure 1.6: Add Employee Page

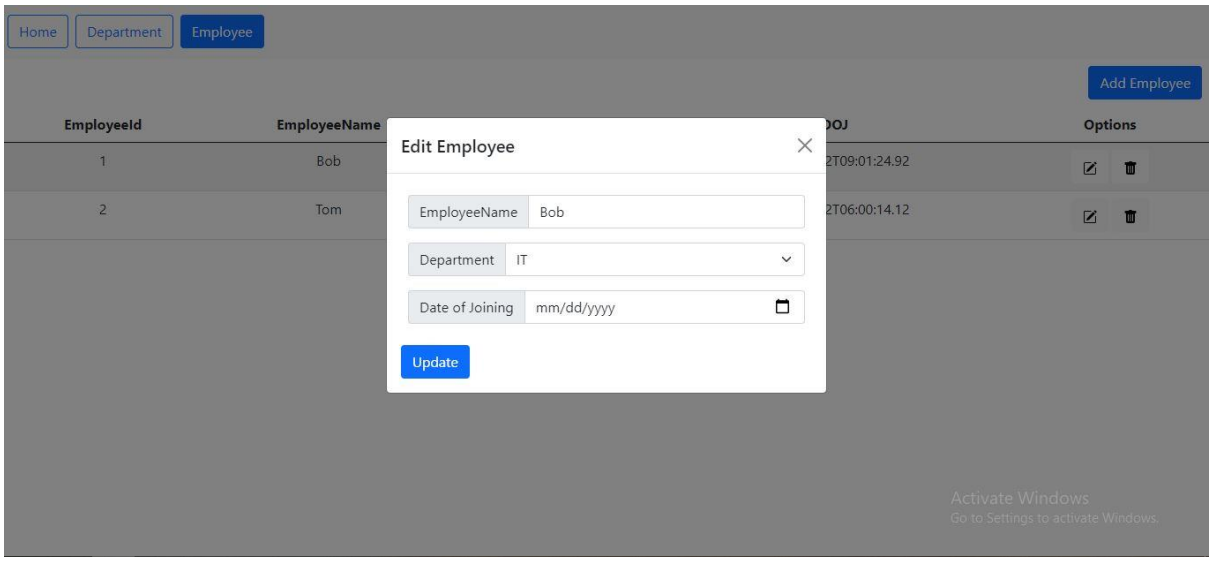

Figure 1.7: Edit Employer Details Page

02 Application Architecture

I have created the system architecture in a 3-Tiered way. We can separate these 3-tiers as presentation tier, application tier and data tier. The Presentation tier created with the help of ReactJS and Application tier (API) created with the help of ASP.NET Core. The Data tier includes the database tables in a MS SQL Server instance.

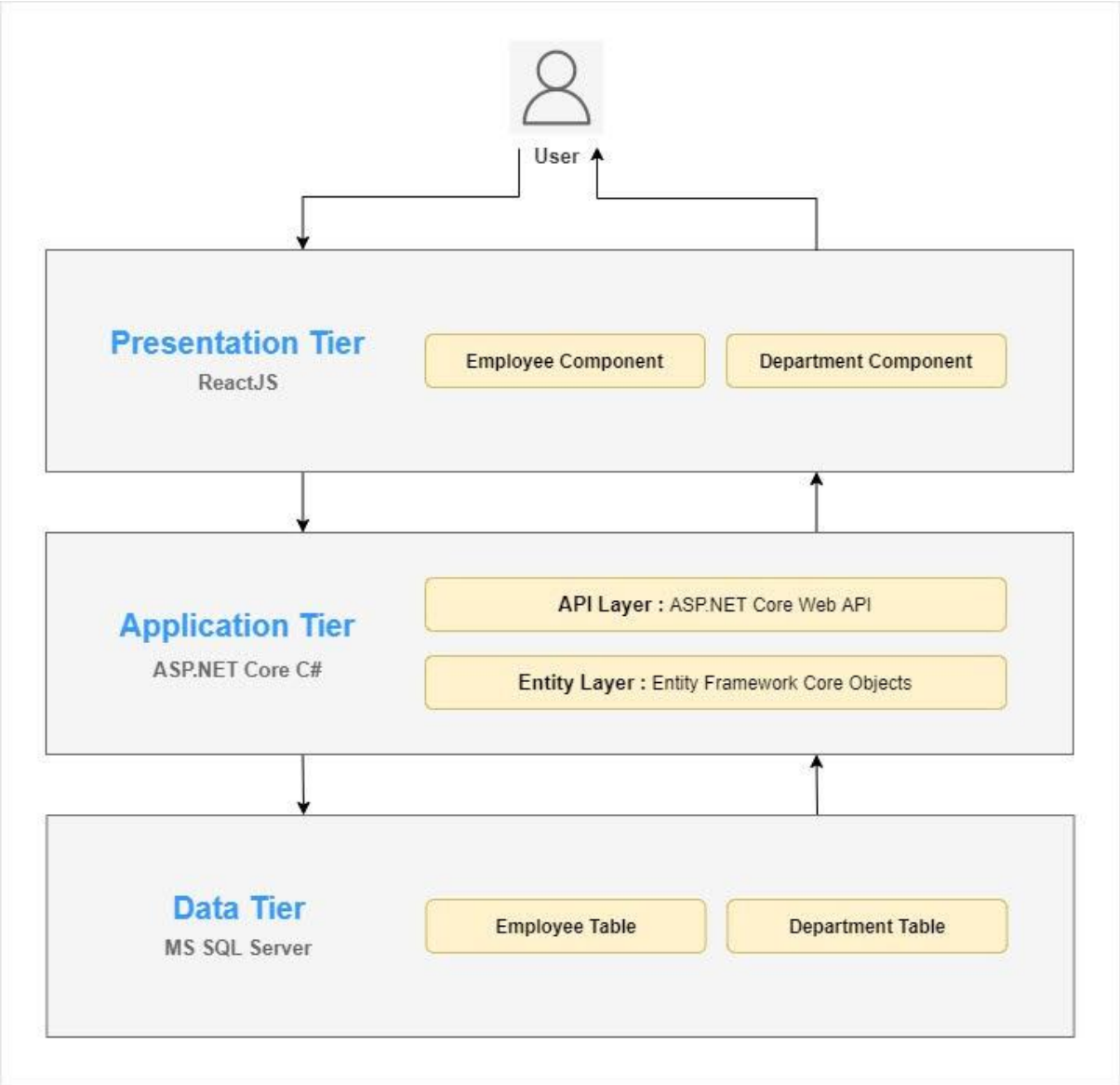

Figure 2.1: Application Architecture Diagram

2.1 Presentation Tier

ReactJS is a JavaScript library for building user interfaces, maintained by Facebook and several individual developers/companies. It is a popular, open-source, front-end JavaScript library that is prominent in the web development domain, especially for single-page applications. It is widely used to create fast and interactive UI elements for web and mobile apps.

Here I have created the presentation tier using ReactJS. Its mainly focusing on two components such as Employee component and Department component.

2.2 Application Tier

ASP.NET Web API is a framework for building HTTP services that can be accessed from any client including browsers and mobile devices. It is an ideal platform for building RESTful applications on the .NET Framework.

Here I have created the Application tier using ASP.Net Core with the help of Entity Framework Core. API mainly includes Employee Controller and Department Controller for handle incoming requests. Employee Model and Department Model helps the Controller to handle the request. Each Controller includes main action methods POST, GET, PUT, DELETE.

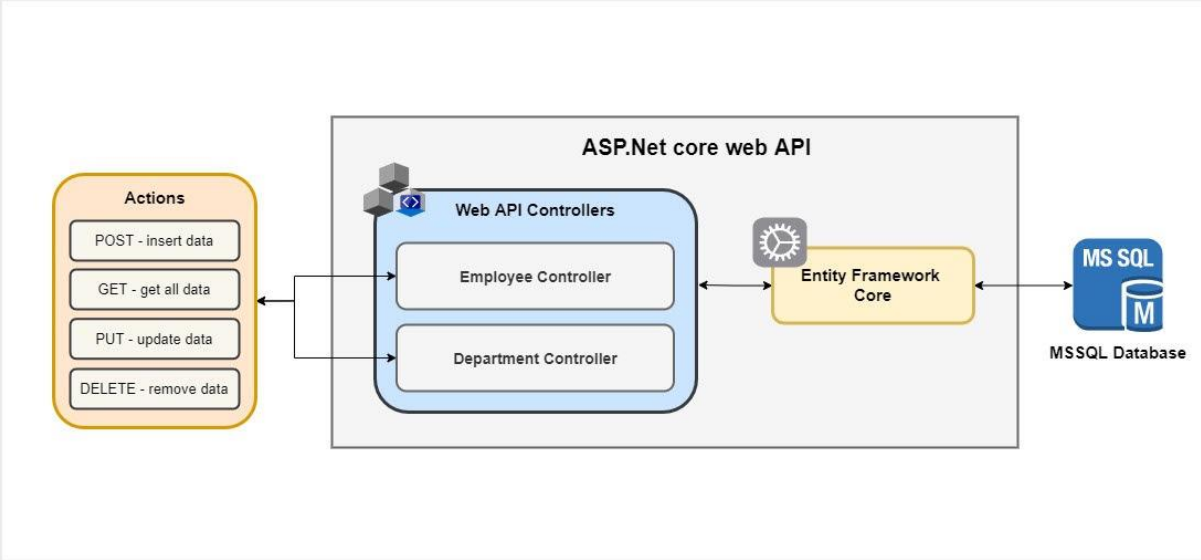

Figure 2.2.1: API Architecture Diagram

2.3 Data Tier

Microsoft SQL Server is a relational database management system (RDBMS) that supports a wide variety of transaction processing, business intelligence and analytics applications in corporate IT environments.

Here I have created database tier with the hep of MS SQL Server. This database includes mainly two tables to proceed data to store, fetch and modify. Employee Table stores the data of Employees and Department Table stores the data of Departments.

3 Docker Architecture

Application works inside docker environment. Here the Presentation Tier works as a separate container. But the application (API) and the data tier works as a composite component. Because the API depends on the database. Both API and Database containers are running as a one unit.

In my system application can access using *http://localhost:3000* and API can access using *http://localhost:8000* . Database can access using localhost, 1445 port.

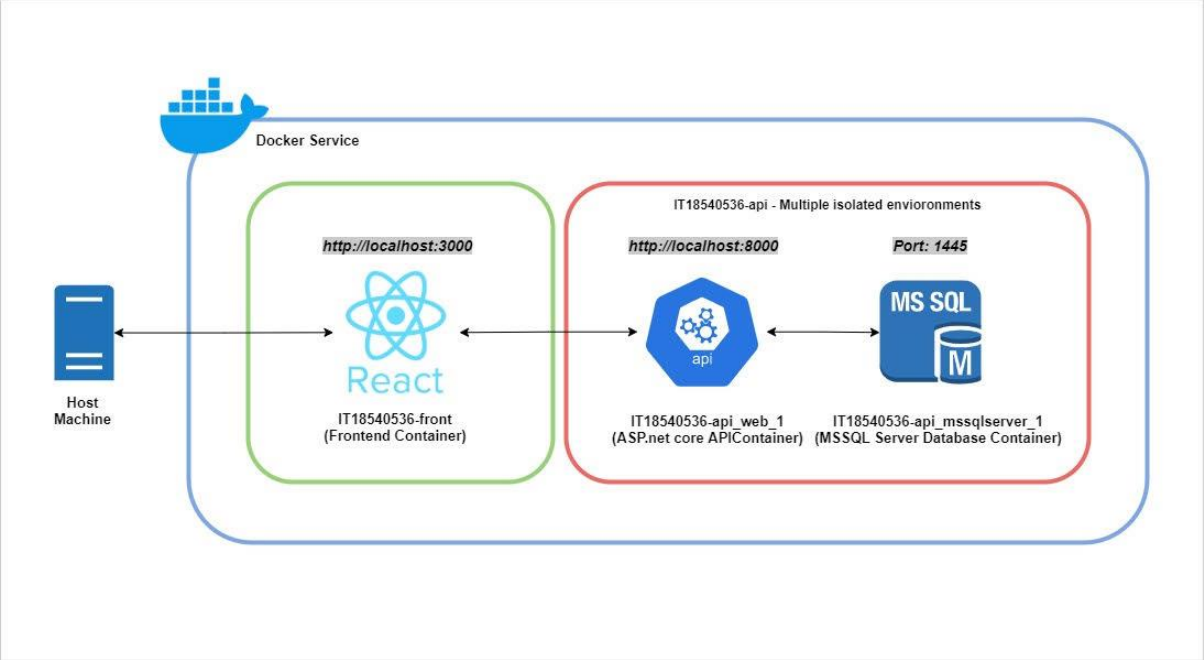

Figure 3.1: Docker Environment Architecture Diagram.

3.1 Docker Dashboard – Docker Images

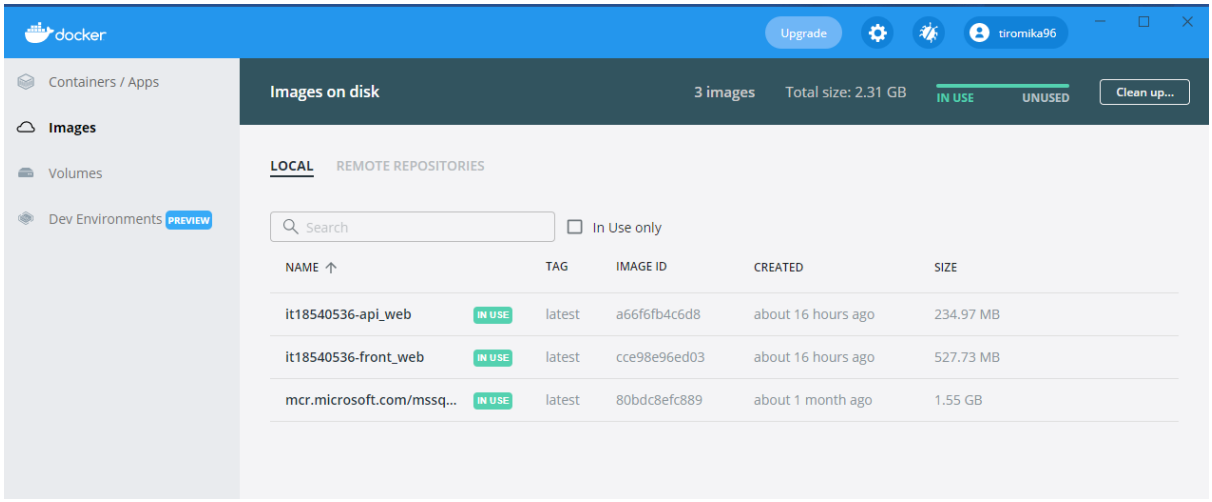

Figure 3.1.1: Docker Dashboard – Docker Images

3.2 Docker Dashboard – Docker Containers

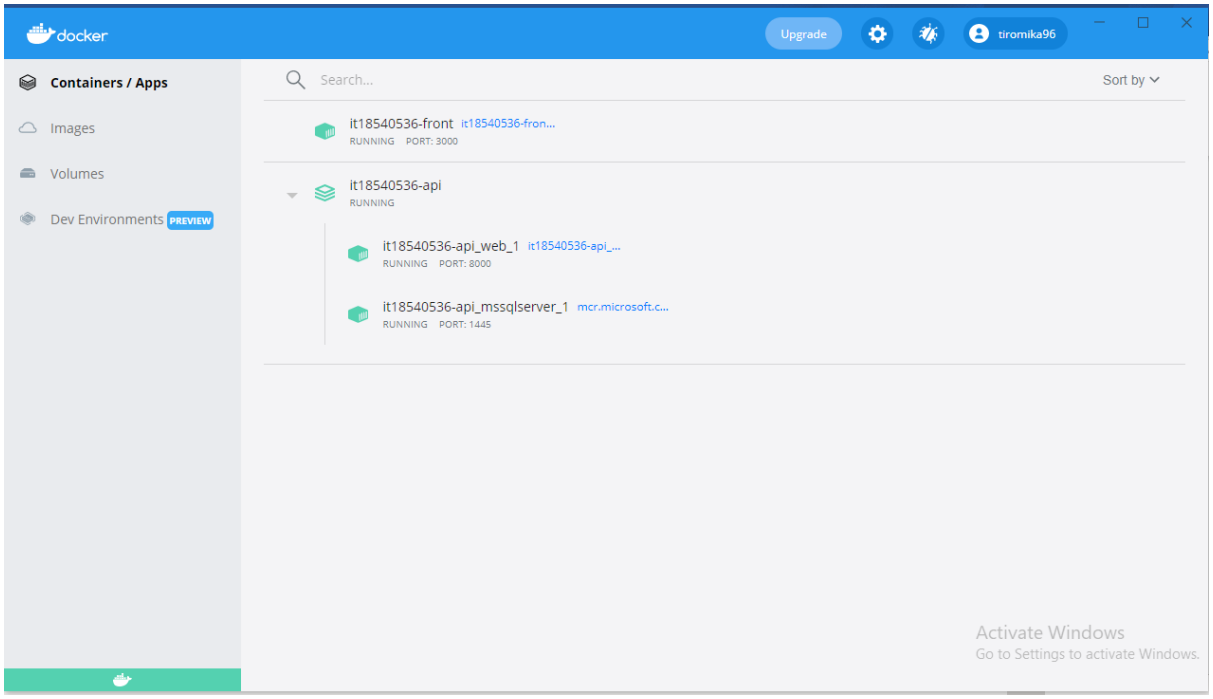

Figure 3.2.1: Docker Dashboard – Docker Containers

3.3 Dockerfile – Backend

After creating the ASP.NET Core web API need to create the Dockerfile inside the project folder. Dockerfile should contains the dependencies and packages need to be installed in the environment that we are going to create.

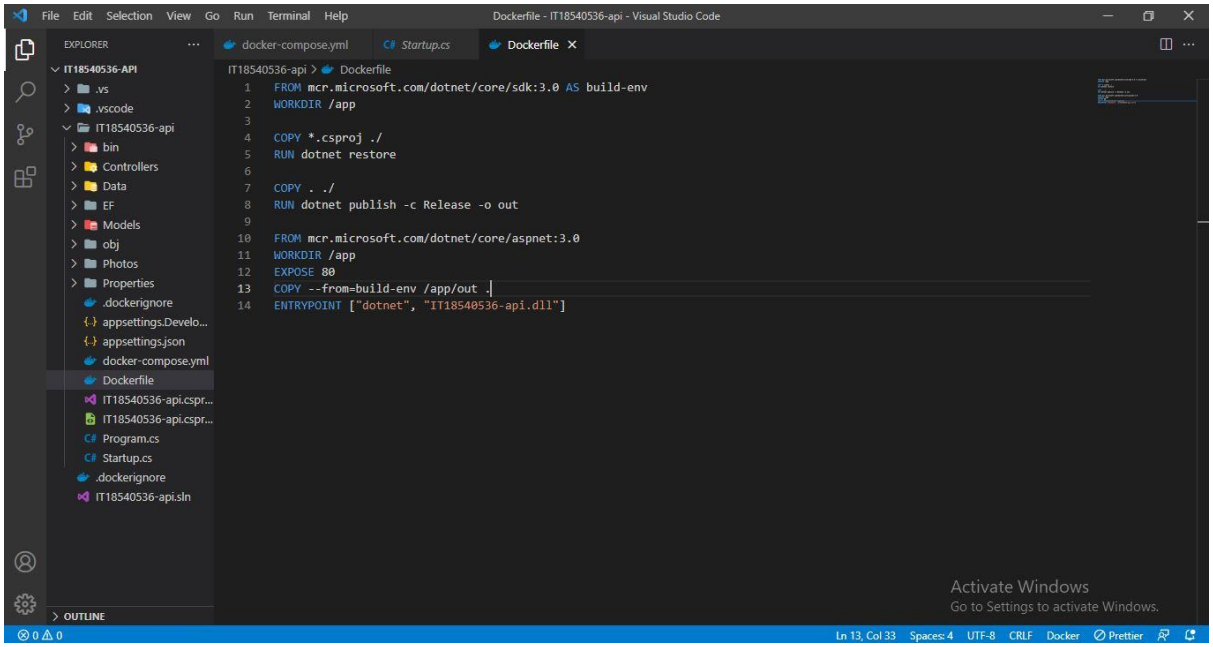

Figure 3.3.1: Dockerfile – Backend

3.4 Docker-compose file – Backend

Docker Compose is a tool that was developed to help define and share multi-container applications. With Compose, we can create a YAML file to define the services and with a single command, can spin everything up or tear it all down. In the compose file, we'll start off by defining the schema version. In most cases, it's best to use the latest supported version. You can look at the Compose file reference for the current schema versions and the compatibility matrix.

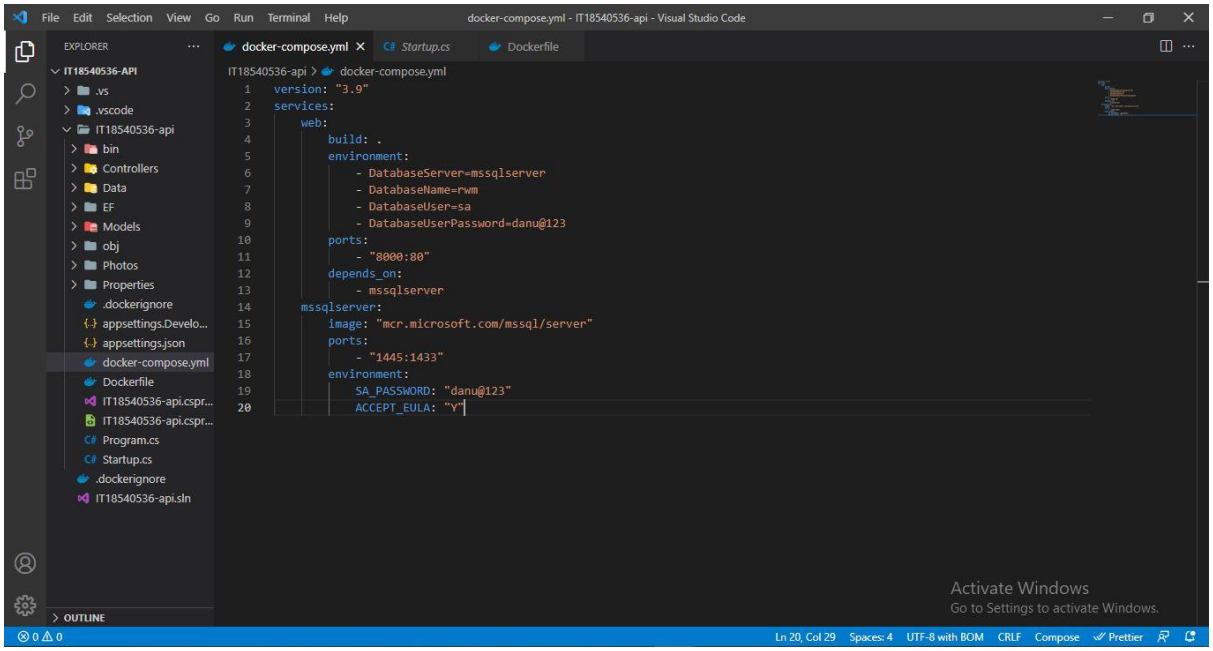

Figure 3.4.1: docker-compose.yml– Backend

3.5 Builds, (re)creates, starts, and attaches to containers for services

docker-compose up

This command should type in package manager console inside the project folder which contains the docker-compose and Dockerfile. To create the required images and pull the predefined images and create containers according to the images we can use docker-compose up. This will create the API related image according to the Dockerfile and use docker-compose file for the other configurations. And also pull the mssql server related docker image from the docker repository and creates the container according to it and configuration change according to the docker-compose file.

Unless they are already running, this command also starts any linked services. The docker-compose up command aggregates the output of each container. If there are existing containers for a service, and the service's configuration or image was changed after the container's creation,

docker-compose up picks up the changes by stopping and recreating the containers (preserving mounted volumes).

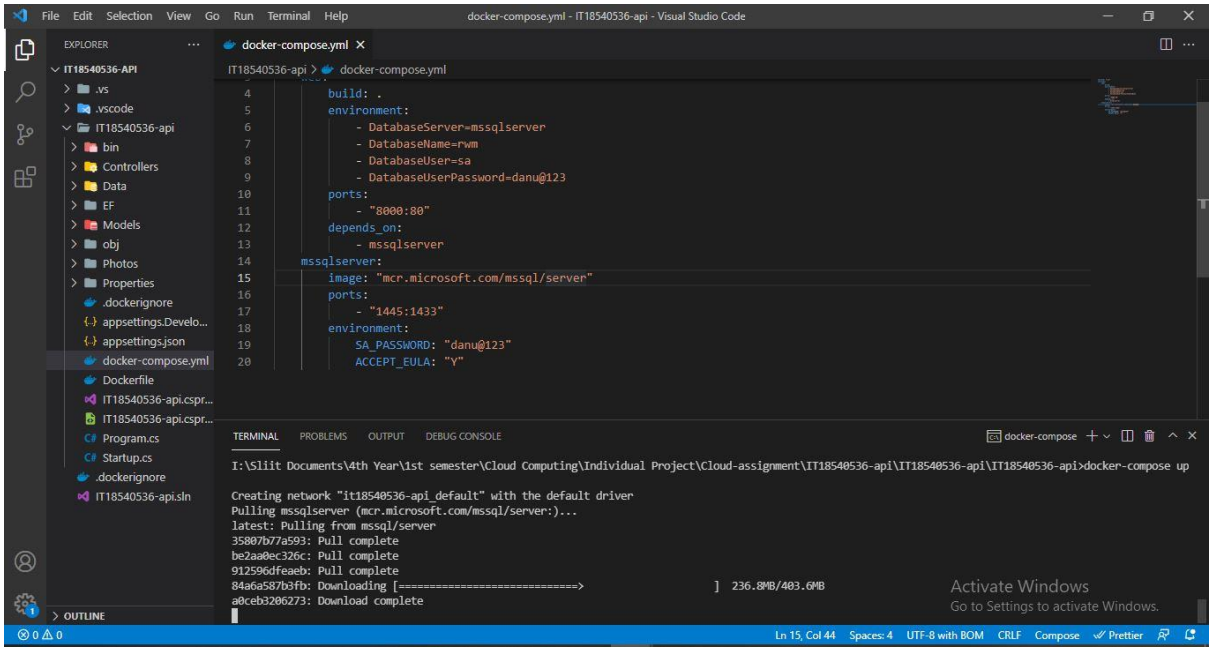

Figure 3.5.1: docker-compose up

3.6 Dockerfile - frontend

Create the Dockerfile inside the project folder. It may contain the configuration related the docker image.

- Pull official base image.
- Set working directory.
- Add `/app/node_modules/.bin` to $PATH.
- Install app dependencies.
- Add App.
- Start App.

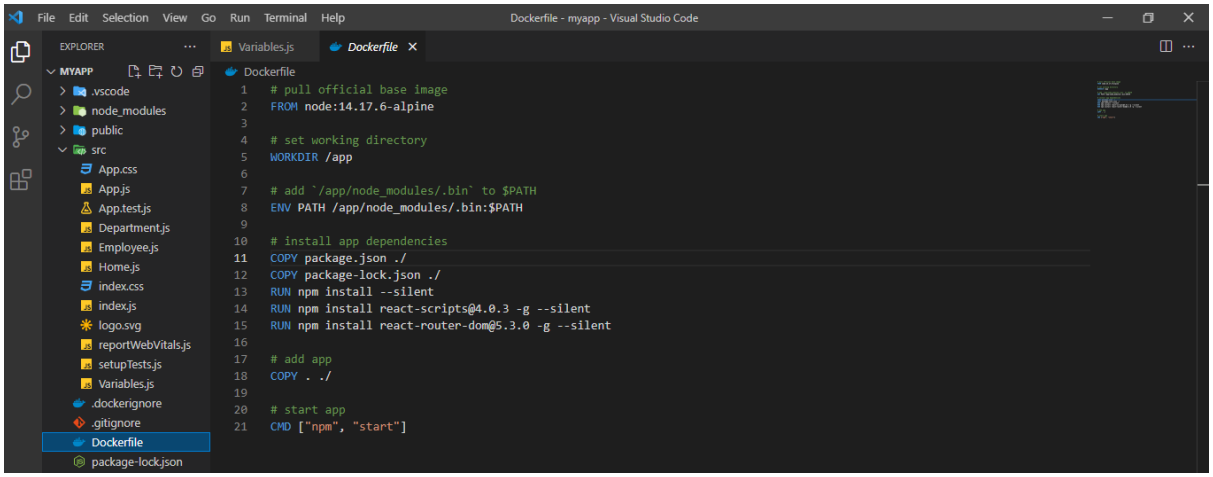

Figure 3.6.1: Dockerfile - Frontend

3.7 Build Docker Image - Frontend

docker build -t it18540536-front_web .

Build with PATH. Tag an image. Image name will be "it18540536-front_web".

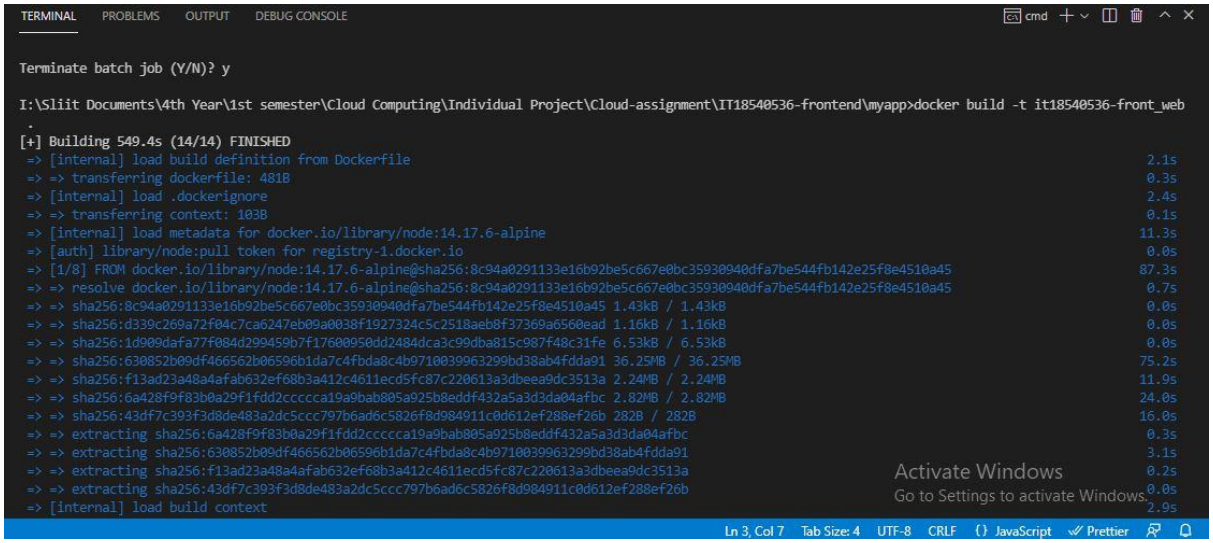

Figure 3.7.1: Create docker image - Frontend

3.8 Run Docker Container - Frontend

docker run -d -it -p 3000:3000/tcp --name it18540536-front it18540536-front_web:latest

The docker run command must specify an IMAGE to derive the container from. Running docker run -d will set the value to true, so your container will run in “detached” mode, in the background. Container name will be "it18540536-front". Image name it18540536-front_web

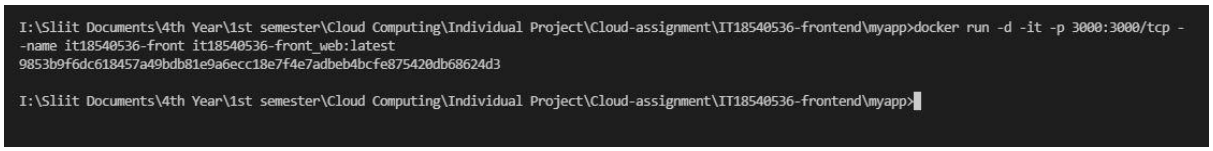

Figure 3.8.1: Run Docker Container – Frontend

4 Application Structure

4.1 Application Structure - Frontend

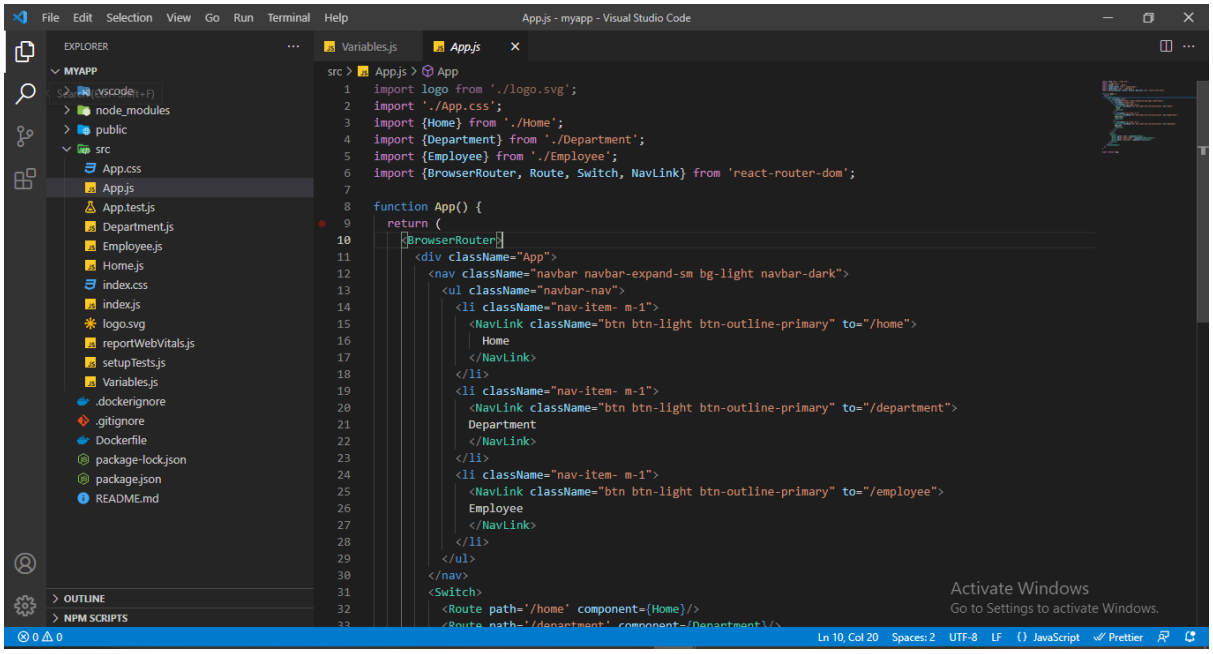

Figure 4.1.1: Application Structure – Frontend

4.2 Application Structure - Backend

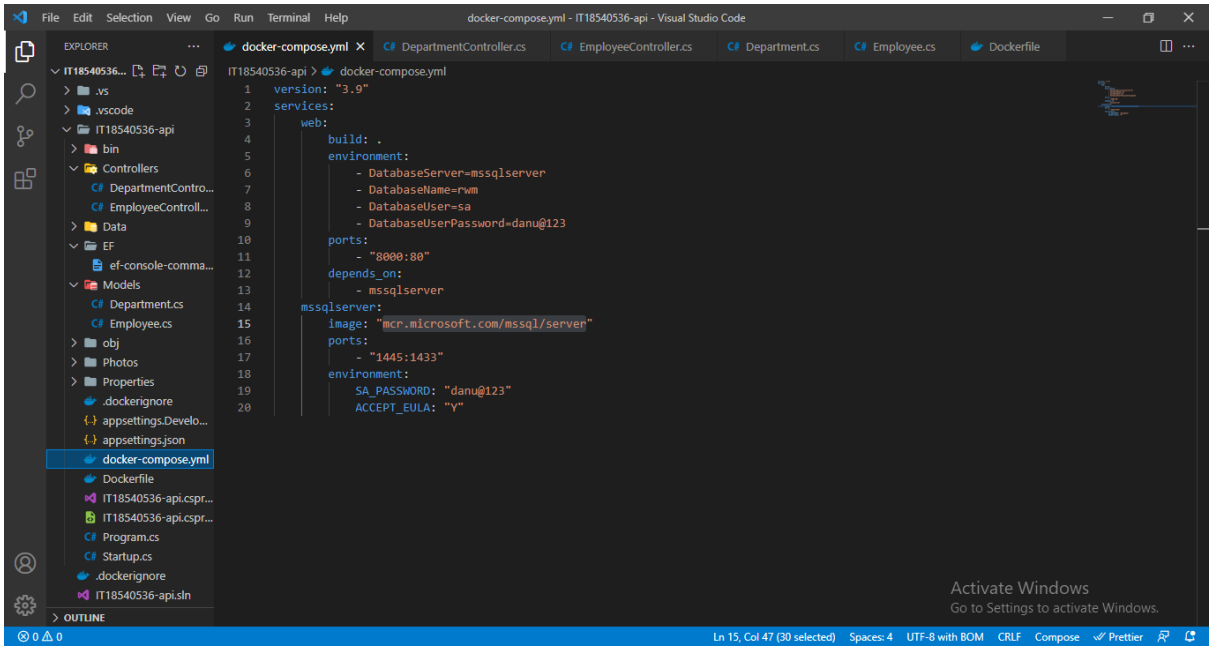

Figure 4.2.1: Application Structure - Backend