|  |  |  |  |
| --- | --- | --- | --- |
| **STUDENT USE** | | **STAFF USE** | |
| Module Name | Web Applications Development | First Marker’s  (acts as signature) |  |
| Module Code | 5COSC017C | Second Marker’s  (acts as signature) |  |
| Lecturer Name | Mikhail Shpirko | Agreed Mark |  |
| UoW Student IDs |  | **For Registrar’s office use only (hard copy submission)** | |
| WIUT Student IDs | 00008488 |
| Deadline date | 30.06.2020 |
| Assignment Type | 🗌Group ☑ Individual |

**COURSEWORK SUBMISSION FORM**

|  |
| --- |
| **MARKERS FEEDBACK (Continued on the next page)** |
|  |

Contents

[Introduction 3](#_Toc75013338)

[Development plan 4](#_Toc75013339)

[Reflection 5](#_Toc75013340)

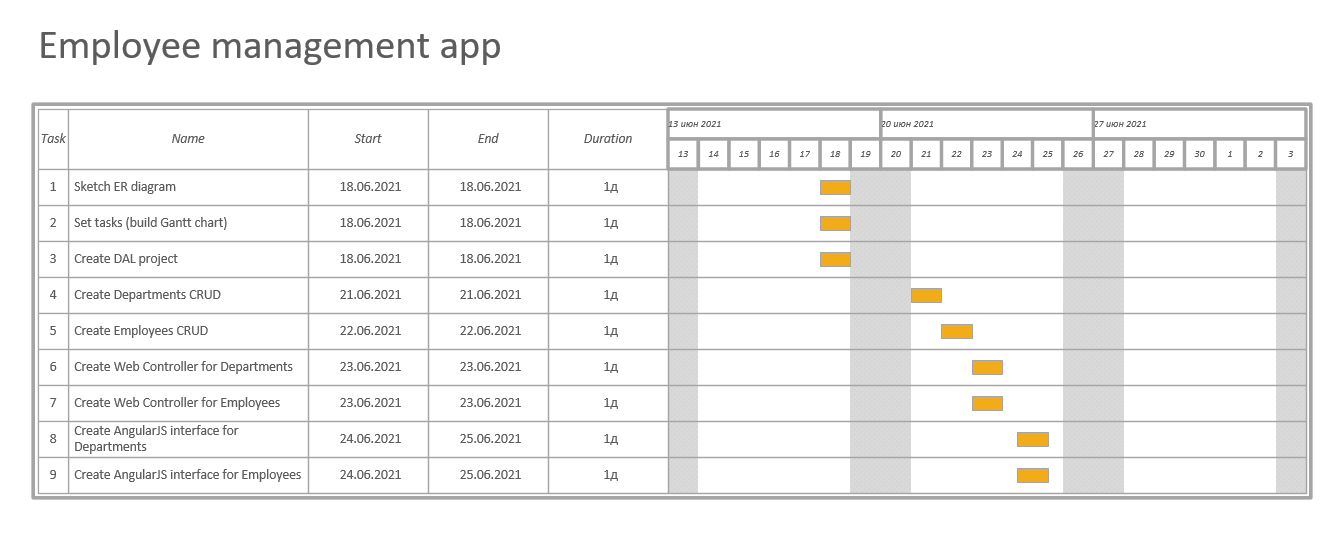
# Introduction

This project involves producing high-fidelity prototype in the view of web application “Employee Management system” based on ASP.NET CORE technology. The application provides CRUD functionalities for *employee* and *department* entities.

Source code of the application is published on GitHub platform reachable at:

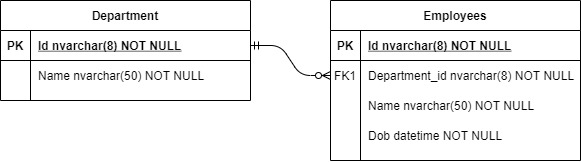
[*https://github.com/00008488/WAD.Application.8488*](https://github.com/00008488/WAD.Application.8488)

# Development plan



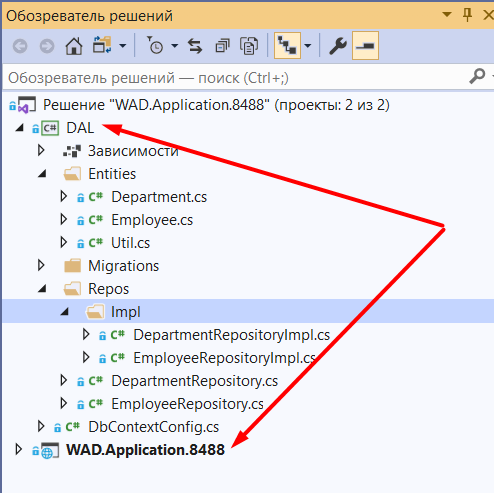
Development of the application was split into iterations that reflected in Gantt chart above. Each sprint involves developing of working functionality of the web application.

Before building the application itself, it was decided to identify major entities within the application boundary. These are reflected in ER diagram below:

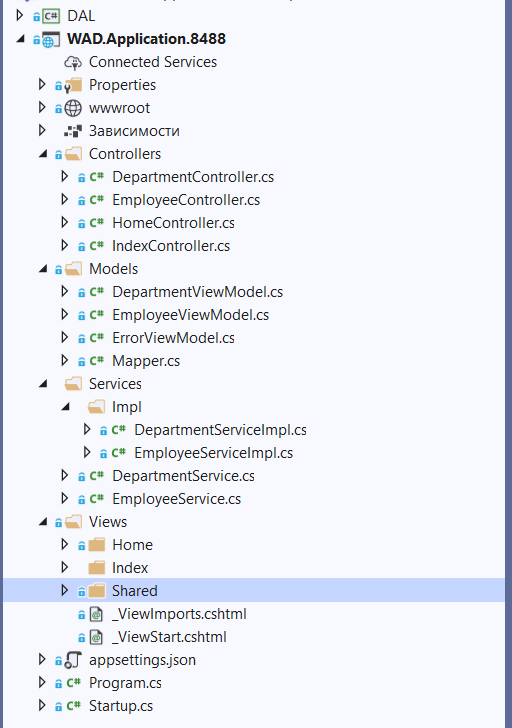


These entities were converted into POCOs inside the program. Moreover, since the application is expected to keep employee data, i.e. data is persisted, it was decided to create database tables sketched as above and to do so, code-first approach was used.

In addition, several programming principles like DRY, SRP and software design patterns were used. Before showing the examples of these, it is apt showing the overall architecture of the application:



* **DAL** module is responsible for working with database:
  + **Entities** – folder responsible to keep database models
  + **Repos** – classes that contain CRUD logic for entities



* **WAD.Application.8488** or simply **UI** – is responsible to work with a user closer:
  + **Controllers** – classes that store API endpoint for serving the SPA
  + **Models** – classes that are used to transfer the same data that entity classes give
  + **Services** – classes that call repository classes from DAL layer
  + **Views** – html files building the UI that is shown for a user

Used software principles:

* DRY – this principle is shown with the example of Mapper.cs class – that is in charge of mapping the entity classes into view model classes and vice versa. This class is used both in Department and Employee Service classes. Separating the logic of mapping into other class helped avoiding simply repeating the same code block both in service classes.
* SRP – overall architecture of the application consists of namespaces enumerated already above where each namespace: *service, repo, entities, controller* and *etc.* is responsible for only one task, in other words, there is none of the classes that is “omnipotent”.
* Adapter design pattern – Mapper class imitates adapter design pattern – where it is responsible for transferring the data from Service -> Repository and vice versa:

