Bern University of Applied Sciences | BFH

Department of Engineering and Information Technology

Bachelor's Thesis (BTI7321) Autumn Semester 2020/21

"Planning of the Assignments for Lecturers(PLANA)" Web Application

Author: Kristina SHIRYAGINA (kristina.shiryagina@bfh.ch)
Supervisor: Prof. Marcel PFAHRER (marcel.pfahrer@bfh.ch)
Expert: Dr. Federico FLUECKIGER

October 20, 2020

Contents

Acknowledgments

Abstract

1 Introduction

At our school at the Department of Technology and Computer Science, actual teacher assignment planning is done using Microsoft Excel tool. This plan is handled by one responsible person. The modern world with the rapid growth of new technologies makes it possible to improve various systems, giving them more and more possibilities, automating many functions and saving a lot of time. This work aims to develop an information system that enables assignment planning for lecturers. But unlike the existing system, it should have the following criteria:

- the teachers themselves should be involved in the planning process
- increased planning flexibility

Users of this system are the **study director** and **teachers**. Such planning implies collaborative work. The planning process involves the input of specific data for specific user-defined views for each user and time limits set by the system.

All of these requirements need a more suitable system than Excel. In the previous project, we compared **Microsoft Excel** and **Web Application** according to several criteria. And we concluded that the web application meets the requirements of the conceived system. The web application is designed to involve **many users**, it can have a database that gives us **consistency of the data**. Also, the data is much **safer** in a database. The web application gives the best **overview** of the entire system. Comparing **desktop application** with **web application** web application wins in:

- it is accessible anywhere
- no update needed
- costs less

We decided to make a web application that will meet all the requirements and will be created using suitable modern technologies. Technologies such as ASP.NET Core Blazor Server with Entity Framework Core (EFCore) and MS SQL (Microsoft Structured Query Language) for the database were chosen. ASP.NET Core Blazor is a new framework that is gaining popularity. Interestingly, thanks to it, it becomes possible to do the entire application in C# without using JavaScript.
This work is a continuation of a project that was completed last semester, In which we have prepared the necessary environment for this project. In project 2, we created a prototype. In this work, the system will be detailed and expanded. In particular, the following goals are pursued:

- **Involvement of lecturers in the planning process.** Lecturers can create their medium- and long-term plans in form of requests and proposal for the definite plan, which is then approved by the person responsible for the planning.
- **Manage planning data.** Each teacher can manage his assignments.
- **Grouping of lecturers.** It should be possible to schedule several lecturers for the same module.
  - Teachers who join a group can independently manage their assignments related to their common module.
  - Each teacher can make a group with other lecturers.
- **Grouping of the modules.** The group of lecturers can choose the group of modules and set themselves to it. This can be done in the form of a proposal for the definitive plan, which is then approved by the person responsible for the planning.

In this work, first, we will explain how the project was organized and how we used a SCRUM to manage it, then we carry out an additional analysis of the system in connection with the expansion of the system requirements. We will make changes to the domain analysis. We will expand the topic of System Architecture and System Design. And then we will cover the topics Project Implementation and Testing.

1.1 Acronyms

Acronyms	Words
EF	Entity Framework
CSS	Cascading Style Sheet
KKK	345

Table 1: Caption2

1.2 Glossary

- **FURPS+eeles2005capturing** is a system for classifying requirements.
  - Functionality
  - Usability
  - Reliability
  - Performance
  - Supportability
- **SignalIR** is a free and open-source software library for Microsoft ASP.NET that allows server code to send asynchronous notifications to client-side web applications.
- **Blazor** is a free and open-source web framework that enables developers to create web apps using C# and HTML. It is being developed by Microsoft.
- **EF Core**
- **HTML** HyperText Markup language
- **SQL** Structured Query Language
- **JS** JavaScript
- **CRUD** Create, read, update and delete
- **UI** User Interface
- **API** Application Programming Interface
- **MS** Microsoft
- **BPMN** Business Process Model and Notation

2 Project Management

2.1 Effort Estimation

The Bachelor's Thesis is designed as a 12 ECTS module. This corresponds to a workload of 360 hours. When we are working on a project, we always record our hours of work in an Excel table. At the end of the project, we will compare this time with the time allotted for the project.

2.2 Scrum

The foundation of the project organization was Scrum. Some principles of Scrum could not be achieved since they need a group of more than two people. Our work was based on the principles of Scrum like the Empirical Process of Control, the core of Scrum, self-organization, value-based prioritization, etc. The Empirical Process of Control includes three main ideas, namely transparency, inspection, and adaptation.
Transparency: The work is carried out in full trust of all parties involved. Everyone has the courage to keep each other up to date with both good and bad news.
Inspection: Inspection is carried out by every one in the Scrum Team. The team openly shows the product at the end of each Sprint.
Adaptation: The team asks constant questions about the progress of work, whether we are on the right way. Depending on this, we can adapt an existing product.

At the beginning of the project, we have discussed and estimated all the work that needs to be done. Meetings between supervisor and developer are weekly and sometimes bi-weekly. Each meeting includes a discussion about what has been achieved since the tasks have been assigned, what can be improved, and scheduling of future tasks.

Scrum Roles

- Product Owner: Mr. Pfahrer
- Development Team: Shiryagina Kristina
- Scrum Master: Shiryagina Kristina

Scrum Plan

To discuss the project, were weekly and biweekly meetings held . They included personal meetings or meetings using Microsoft Teams. The meetings consisted of:

- Sprint Review. It includes a show of work and its discussion.
- Sprint Planning. It includes the scheduling of future tasks.
- Sprint Retrospective. It includes discussion about what went well and what went wrong, what we should do differently.

Scrum Artifacts

Sprints

3 System Requirements

3.1 Product Backlog and User Stories

Epic	User Story ID	User Story Name	User Story	Acceptance Criteria
As a Lecturer I want to be able to create my own medium and long-term assignment plans in form of requests and proposal for the definite plan and manage it so that it will be possible a mutual development of the main assignment plan.	01	list modules	As a Lecturer, I want to see the list of modules for concrete semester, so that I can choose the modules I want to plan in my own plan.	User is able to: • have his own plan page • navigate to his own plan page • able to see the module list
	02	add modules to own assignment plan	As a Lecturer, I want to be able to add the modules to my assignment plan I want to teach in specific semester or remove it from my plan so that I can participate in the main planning by making a suggestions or requests.	User is able to: • have his own plan page • navigate to his own plan page • able to select a module and set himself to it • able to remove himself from the module he has added himself • able modify a data of his own planning
	03	manage own plan	As a Lecturer, I want to be able to manage my plan, so I can modify my plan as I like.	User is able to: • have his own plan page • navigate to his own plan page • able to select a module and set himself to it • able to remove himself from the module he has

Epic	ID	User Story Name	User Story	Acceptance Criteria
As a Study Director, I want to be able to make group of lecturers and attach it to a specific module, and also make group of modules so this will increase planning flexibility . As a Study Director, I want to be able to attach a specific group of modules to a specific group of teachers so that further joint planning of these modules will be easier	04	See the requests for the groups and for the modules	As a Study Director, I want to see teachers' suggestions for group work and also their proposal for the selected modules so that it will be easier to approve specific groups.	User is able to: • open the main planning matrix with suggestions for group work and teacher suggestions for teaching modules. • manage matrix page, making the necessary adjustments
	05	Makes groups of lecturers	As a Study Director, I want to be able to make groups of lecturers in the assignment plan.	User is able to: • open the main planning page • select list of teachers • select several teachers and save them as a group. • open module view and attach a specific group of lecturers to a specific module.
	06	Make groups of modules	As a Study Director, I want to be able to make groups of modules in the assignment plan.	User is able to: • open the main planning page • select list of modules • select several modules and save them as a group.
	07	Attach lecturer/group of lecturers to the module	As a Study Director, I want to be able attach lecturer or group of lecturers to the specific module in the assignment plan.	User is able to: • open the main planning page • select list of modules • click add lecturer button or add group of lecturer button

Epic	User Story ID	User Story Name	User Story	Acceptance Criteria
			As a Study Director, I want to be able attach a group of lecturers to the group of modules in the assignment plan.	User is able to: • open the main planning page • select list of modules • select a specific group of modules • click add group of lecturer button • select group of lecturer and save it
As a Lecturer, I can join a group with another lecturers, so that we can use a system as a group.				User is able to: • •
As a Lecturer, who joins a group, I want to be able to independently manage the tasks related to the common module				User is able to: • •

Table 4: Product Backlog

Epic	User Story ID	User Story Name	User Story	Acceptance Criteria
			As a Lecturer and the member of the group, I want to be able build a subgroup with another Lecturer/s so that we can manage assignments independently from another lecturers in this group.	User is able to: •
a group.				User is able to: • •
module				User is able to: • •

Table 5: Product Backlog

3.2 Functional Requirements

4 System Architecture and System Design

In project 2 we have started with describing of System architecture and design. In this work we want go deeper into this topic.

5 Domain Analysis

5.1 Domain Model

The domain model (Figure 1) shows us the important concept classes, associations and multiplicities between them. The model made in the previous project is shown in black. And other colors show new concepts and associations associated with new tasks.

In the definition of concepts, new concepts, associations between them and other concepts are highlighted in blue.

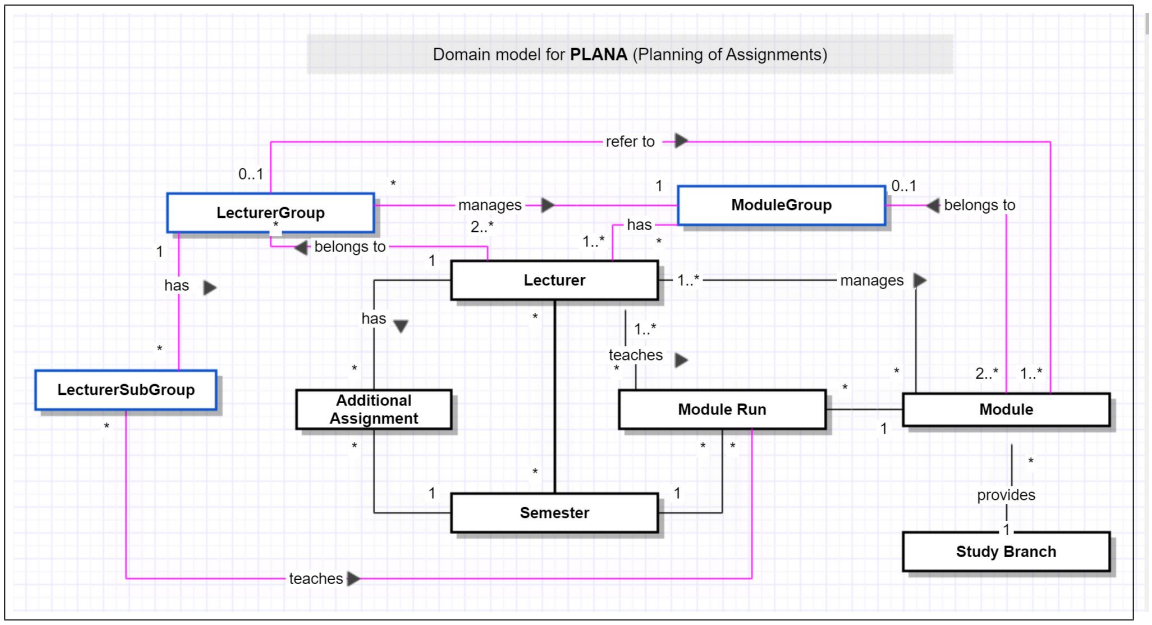

Figure 1: Domain Model for PLANA

i. Concept Definition

- Concept class **Lecturer** models a person who teaches in a school.
- Concept class **Study Branch** models a conceptual subdivision of subjects that form a study programme.
- Concept class **Module** models a set of independent units that form a course at the school.
- Concept class **Module Run** models executions of a course in different languages.
- Concept class **Additional Assignment** models a set of independent units that form an additional task for the assignment's plan for the lecturer.
- Concept class **Semester** models the periods in the year, during which the lecturer is present in the school.
- Concept class **LecturerGroup** models group of teachers who will jointly participate in one or more modules.
- Concept class **ModuleGroup** models several modules collected in groups for further effective use.
- Concept class **LecturerSubGroup** models subdivision of the main group of teachers to carry out the module in different student groups.

ii. Association Definition

Concept Pair	Association Definition	Association Name
Lecturer -> Module Run	Lecturer can teach zero or more Module Runs . Each Module Run can be taught by one or more Lecturers	teaches
Semester -> Module Run	Semester can include zero or more Module Runs .	includes
Semester -> Additional Assignment	Semester can include zero or more Additional Assignments	includes
Lecturer -> Additional Assignment	Lecturer can have zero or more Additional Assignments	has
Lecturer -> Module	Lecturer can manage zero or more Modules. A module can be managed by one or more Lecturers	manages
Module -> Module Run	Module is executed as many as there are module runs or not executed at all. A Module run is executed for one Module.	executes
Study Branch -> Module	Each Study Branch has many modules. These modules belong to exactly one study branch.	has
Semester -> Lecturer	In each Semester, there are many lecturers that are teaching, and these teachers are teaching in more than one Semester	includes
LecturerGroup -> Module	Each Module can have maximal one LecturerGroup. LecturerGroup can refer to multiple Modules.	refer to
Lecturer -> LecturerGroup	Each Lecturer can belongs to zero or more group. Each LecturerGroup must consist of two or more lecturers.	belongs to
LecturerGroup -> LecturerSubGroup	Each LecturerGroup can have from zero to many LecturerSubGroups. LecturerSubGroup can refer to exactly one LecturerGroup	has
LecturerSubGroup -> ModuleRun	Each LecturerSubGroup can teach zero to many Module Run.A Module Run can be taught by zero or several LecturerSubGroup.	teaches
Module -> ModuleGroup	A Module can belong to zero or one ModuleGroup. ModuleGroup can have from two to many Modules.	belongs to
LecturerGroup -> ModuleGroup	A LecturerGroup can manage exactly one ModuleGroup. A ModuleGroup can have zero to many LecturerGroup. .	manages

Table 6: Association Definition

6 Technologies

7 Creating the Projects

7.1 Structure of Projects and Folders

7.2 Data Models

7.3 Entity Framework Core Packages

7.4 Connection String

7.5 Creating the Database Context Class

7.6 Entity Framework Core Configuration

7.7 Database and Entity Framework Core

Entity Framework(EF) Core is an object-relational mapper (O /RM). It is designed to make writing code for accessing a database quick and intuitive. There are many good reasons to use EF Core. It supports LINQ queries, change tracking, updates, and schema migrations. EF Core works with many databases, including SQL Database, SQLite, MySQL, PostgreSQL, and Azure Cosmos DB. book **efa ef**

7.8 The PLANA App's Relational Database

Our database has many types of relationships we can have in EF Core. The types are: One-to-many: Lecturer Many-to-many: One-To-Many Relationship : Lecturer to an Additional Assignment Semester to a Additional Assignment Semester to a Module Run Module to a Module Run Study Branch to a Module Many-To-Many Relationship : Lecturers to Semester Lecturers to Module Lecturers to Module Run

7.9 Modeling Types of Database Relationships

Many-to-many Relationship

Creating many-to-many relationship is little bit different from the one-to-many and one-to-one. We will take as example relation between Lecturer and Module.

In EF Core database doesn't directly implement this kind of relationships. First we have to create class Lecturer and class Module. Then we have to create one more class, we call it LecturersModules. This class links lecturers to their modules.
At the LecturerModules class there are two properties, LecturerId and ModuleId. There are both - primary keys and foreign keys, known as a composite key.**efa**

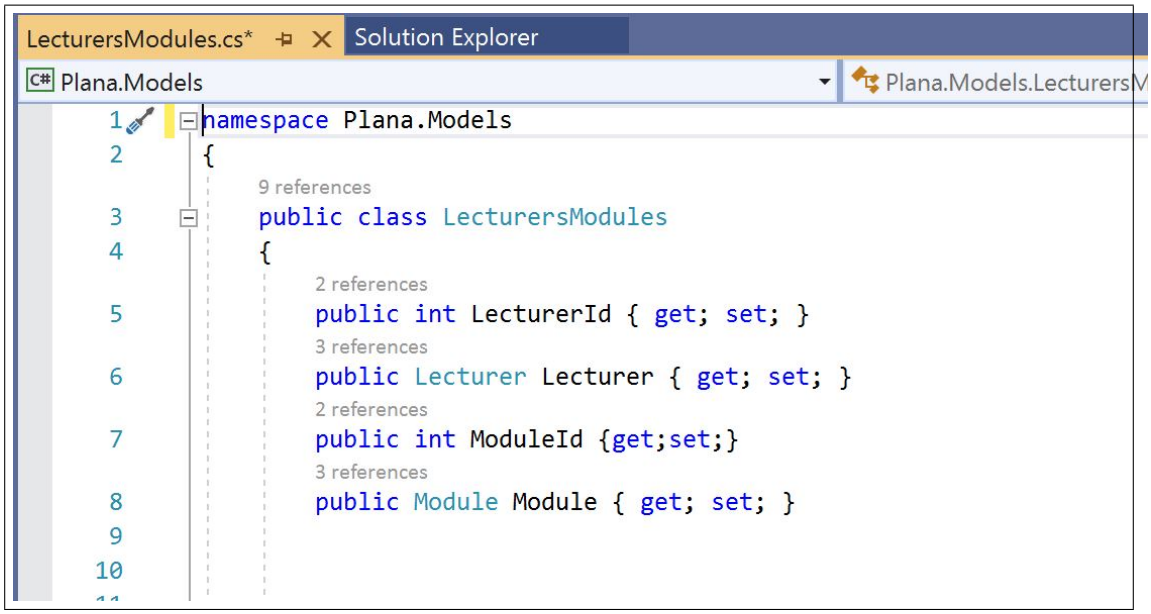

Figure 2: The LecturersModules entity class

The next step is adding necessary code to the AppDbContext class. We add

- **DbSet<Lecturer>**
- **DbSet<Module>**
- **DbSet<LecturersModules>**

The Figure 2 below shows this process. In the OnModelCreating method we add

- **modelBuilder.Entity<LectuerersModules>().HasKey(x=> new x.LecturerId, x.ModuleId**;

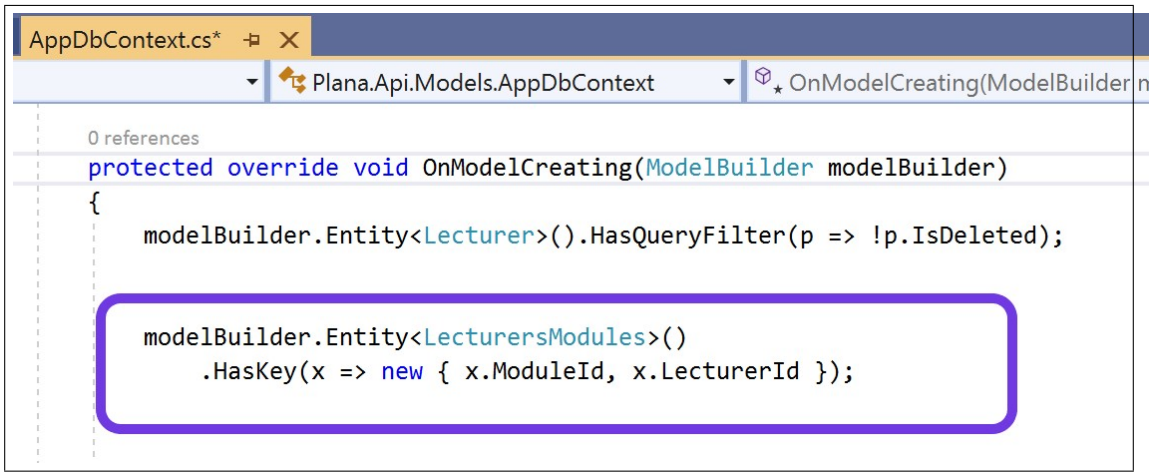

Figure 3: Adding Entity LecturersModules to the AppDbContext class

We need write just this and Entity Framework Core will do the correct implementation that we can see then in the migration files. **patrick**

```
migrationBuilder.CreateTable(
    name: "LecturersModules",
    columns: table => new
    {
        LecturerId = table.Column<int>(nullable: false),
        ModuleId = table.Column<int>(nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_LecturersModules", x => new { x.ModuleId, x.LecturerId });
        table.ForeignKey(
            name: "FK_LecturersModules_Lecturers_LecturerId",
            column: x => x.LecturerId,
            principalTable: "Lecturers",
            principalColumn: "LecturerId",
            onDelete: ReferentialAction.Cascade);
        table.ForeignKey(
            name: "FK_LecturersModules_Modules_ModuleId",
            column: x => x.ModuleId,
            principalTable: "Modules",
            principalColumn: "ModuleId",
            onDelete: ReferentialAction.Cascade);
    });
```

Figure 4: Initial Migration File

From Figure 3 we can see that one many-to-many relationship has transformed in two one-to-many and many-to-one relationships.

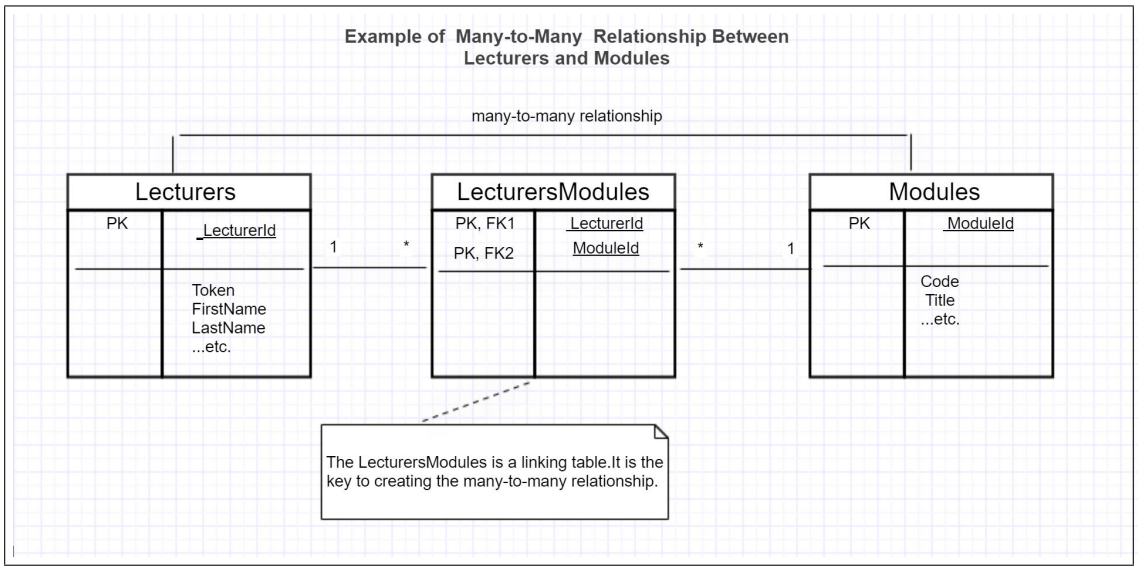

Figure 5: Creating Many-To-Many Relationship

7.10 Creating Database

7.11 Creating a Repository

todo: photo of repository

In our project we create a repository interfaces and implementation classes.
We use **IQueryable<T> and IEnumarable<T>** interfaces. With IQueryable<T> interface the objects can be queried in more efficient way.
For example: **public IQueryable<ModuleRun> ModuleRuns => appDbContext.ModuleRuns;** the ModuleRuns property in the context class returns a DbSet<ModuleRun> object, which implements the IQueryable<T> interface.

```
using Plana.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace Plana.Api.Models
{
    public class ModuleRunRepository : IModuleRunRepository
    {
        private readonly AppDbContext appDbContext;

        public ModuleRunRepository(AppDbContext appDbContext)
        {
            this.appDbContext = appDbContext;
        }
        public IQueryable<ModuleRun> ModuleRuns => appDbContext.ModuleRuns;
```

Figure 6: The ModuleRunRepository.cs file in the Plana.Api/Models folder

Then we create the Repository Service in the Startup.cs file.

```
public class Startup
{
    0 references
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    2 references
    public IConfiguration Configuration { get; }

    // This method gets called by the runtime. Use this method to add services to the
    0 references
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddDbContext<AppDbContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("DbConnection")));

        services.AddScoped<ILecturerRepository, LecturerRepository>();
        services.AddScoped<IModuleRepository, ModuleRepository>();
        services.AddScoped<IModuleRunRepository, ModuleRunRepository>();

        services.AddControllers();

    }
```

Figure 7: Creating Services in Startup.cs File

core3

Creating the Database Migration, Code-First Migration

Entity Framework Core makes it possible to generate schema for the database from the data model classes using **migrations dotnet em migrations add Initial core3**

7.12 Creating Seed Data

The seed data is the data that is used to populate the database. For seed data we add class SeedData.cs in the Models folder in Plana.Api project **core3**. By default, Entity Framework Core uses cascade deletes for depend relationships with non-nullable foreign keys. **efa**
..add photo of seed class (give it name The contents of the SeedData.cs class

Configuration of Core Services and Entity Framework

It is necessary to make changes in Startup.cs class in Plana.Api project - configure Entity Framework Core and set up the services that will be used to access the database **efa**. The figure below shows all these configurations.

```
using Microsoft.AspNetCore.Builder;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Plana.Api.Models;

namespace Plana.Api
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }
        public IConfiguration Configuration { get; set;}


        public void ConfigureServices(IServiceCollection services)
        {
            services.AddDbContext<AppDbContext>(options =>
            options.UseSqlServer(Configuration.GetConnectionString("DbConnection")));

            services.AddScoped<ILecturerRepository, LecturerRepository>();
            services.AddScoped<IModuleRepository, ModuleRepository>();
            services.AddScoped<IModuleRunRepository, ModuleRunRepository>();
            services.AddControllers();
        }

        public void Configure(IApplicationBuilder app, AppDbContext context)
        {
            app.UseDeveloperExceptionPage();
            app.UseHttpsRedirection();
            app.UseRouting();
            app.UseAuthorization();
            app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllers();
            });
            SeedData.SeedDatabase(context);
        }
    }
}
```

Figure 8: Startup.cs in the Plana.Api project.Preparing Services and Middleware

7.13 Create a Controller

Complex Data Model

in this section I would like to highlight more complex features of coding and data model structure in asp .net core.

```
public async Task<IEnumerable<Lecturer>> GetLecturers()
        {
              return await appDbContext.Lecturers.ToListAsync();

        }
```

Figure 9: LecturerRepository.cs file in the Plana.Api project's folder


```
{"lecturerId":12,"photoPath":"images/michele.jpg","lecturersModules":null,"lecturersModuleRuns":null,"lecturersSemesters":null,"
additionalAssignments":null,"birthDate":"1986-05-11T00:00:00","gender":0,"workingRate":0.0,"isActive":false,"activeTill":"0001-
01-
01T00:00:00","isDeleted":false,"firstName":"Michele","lastName":"Orsi","token":null,"email":"mo@gmx.ch","password":null,"role":0
}]
```


Figure 10

```
public async Task<IEnumerable<Lecturer>> GetLecturersModules()
        {
            return await appDbContext.Lecturers
                .Include(m => m.LecturersModules)
                .ThenInclude(mo => mo.Module).ToListAsync();
        }
```

Figure 11: ...

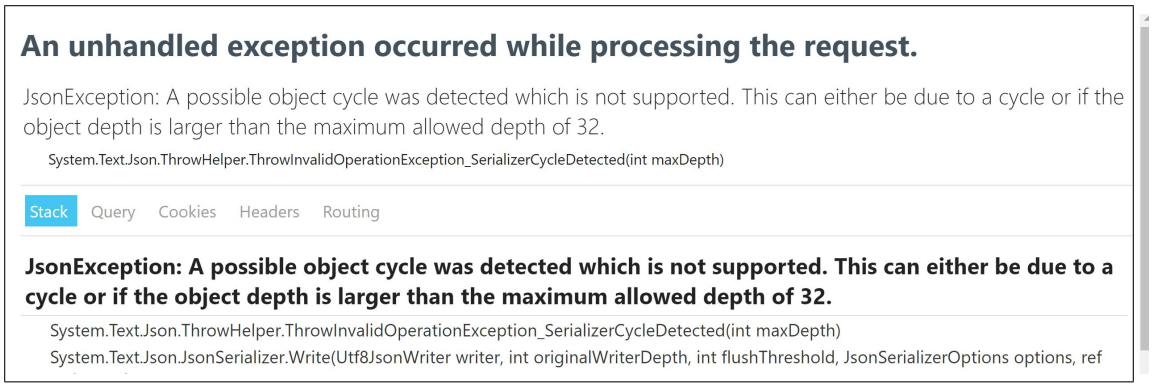

Figure 12

```
Package Manager Console

PM> Install-Package Microsoft.AspNetCore.Mvc.NewtonsoftJson -Version 3.0.0
```

Figure 13

```
public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }
        public IConfiguration Configuration { get; set; }

        scr
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddDbContext<AppDbContext>(options =>
            options.UseSqlServer(Configuration.GetConnectionString("DbConnection")));

            services.AddScoped<ILecturerRepository, LecturerRepository>();
            services.AddScoped<IModuleRepository, ModuleRepository>();
            services.AddScoped<IModuleRunRepository, ModuleRunRepository>();
            services.AddControllers();
            services.AddControllers().AddNewtonsoftJson(options =>
            options.SerializerSettings.ReferenceLoopHandling =
Newtonsoft.Json.ReferenceLoopHandling.Ignore
);
        }
```

Figure 14: ...


```
le":0},{"lecturerId":12,"photoPath":"images/michele.jpg","lecturersModules":[{"lecturerId":12,"moduleId":14,"module":
{"ects":0,"moduleId":14,"title":"Computer Science Basics","code":"BTI1021","lectPerWeek":4,"totalHours":200.0,"lecturers":
[],"moduleRuns":null,"studyBranch":null}}],"lecturersModuleRuns":null,"lecturersSemesters":null,"additionalAssignments":null,"bi
rthDate":"1986-05-11T00:00:00","gender":0,"workingRate":0.0,"isActive":false,"activeTill":"0001-01-
01T00:00:00","isDeleted":false,"firstName":"Michele","lastName":"Orsi","token":null,"email":"mo@gmx.ch","password":null,"role":0
}]
```


Figure 15

Blazor Server

Configuring ASP.NET Core for Blazor Server

Call the API from Asp.net Core Blazor

.. add picture with a blazor page

...write about imports

... write about registration of http client services

```
public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }


        public void ConfigureServices(IServiceCollection services)
        {
            services.AddRazorPages();
            services.AddServerSideBlazor();
            services.AddAutoMapper(typeof(LecturerProfile));

            services.AddHttpClient<ILecturerService,LecturerService>(client =>
                {
                    client.BaseAddress = new Uri("https://localhost:44399/");
                });

             services.AddHttpClient<ILecturersModulesService,
LecturersModulesService>(client =>
            {
                client.BaseAddress = new Uri("https://localhost:44399/");
            });
        }
```

Figure 16: Registration of Http Client Services in Startup File in Plana.Web Project folder

```
using Microsoft.Extensions.Hosting;
using Plana.Web.Models;
using Plana.Web.Services;

namespace Plana.Web
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        public void ConfigureServices(IServiceCollection services)
        {
            services.AddRazorPages();
            services.AddServerSideBlazor();
            services.AddAutoMapper(typeof(LecturerProfile));

            services.AddHttpClient<ILecturerService,LecturerService>(client =>
                {
                    client.BaseAddress = new Uri("https://localhost:44399/");
                });
        }


        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            else
            {
                app.UseExceptionHandler("/Error");
                app.UseHsts();
            }

            app.UseHttpsRedirection();
            app.UseStaticFiles();

            app.UseRouting();

            app.UseEndpoints(endpoints =>
            {
                endpoints.MapBlazorHub();
                endpoints.MapFallbackToPage("/_Host");
            });
        }
    }
}
```

Figure 17: Adding Services and Middleware in the Startup.cs File in the Plana.Web project folder

When we need to use some data from other folders in a blazor files like Razor files(.razor) all necessary imports we include in the partial class **_Imports.razor** using the **@using** directives.

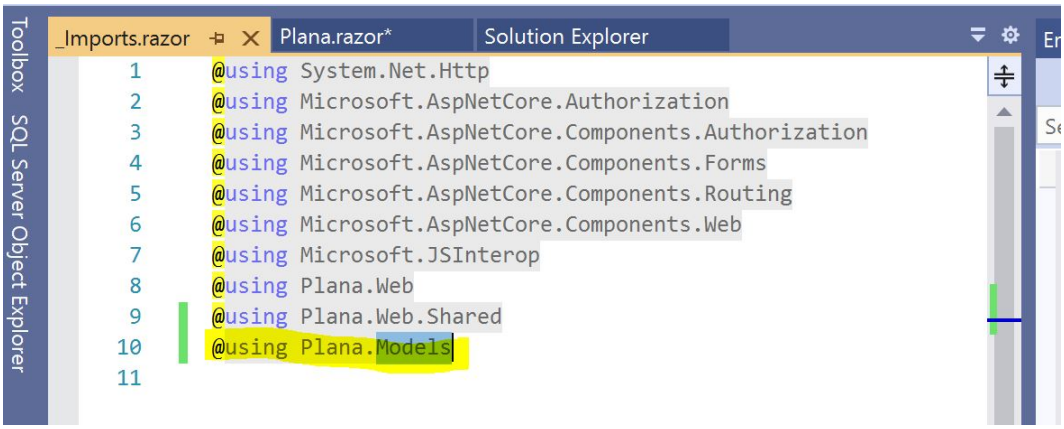

Figure 18: Adding required namespaces to the _Imports.razor file in Plana.Web project folder

7.14 Setup for Blazor

To use the Blazor framework it is necessary to install :

- **.NET Core SDK 3.1 or later** from `http://dotnet.microsoft.com/download`
- **Visual Studio 2019** from `https://visualstudio.microsoft.com/downloads/`

The sprints covered a one three-four weeks period. At the end of each sprint, there was a discussion with the supervisor.

8 Testing

9 Summary

9.1 Conclusions

9.2 Future Work

9.3 Lessons Learned

10 List of illustrations

11 Contents of the table

12 Appendix

13 Declaration of Authorship

I hereby certify that I composed this work completely unaided, and without the use of any other sources or resources other than those specified in the bibliography. All text sections not of my authorship are cited as quotations, and accompanied by an exact reference to their origin.

Place, date:

Signature:

14 Protocol

Frequency: (biweekly)
Meeting length: (60 minutes)

Agenda

- Demo and Discuss Deliverable(Demo)
- Planning next Goals(Plan)
- Lessons learned (Lessons)
- Date, time of the next meeting(next meeting)

Report from 24.09.20
Plan
Future goals are:

-
-

Lessons learned

Next Meeting: 08.09.20
***
Report from 08.10.20
Plan
Future goals are:

-
-

Lessons learned

Next Meeting: 21.10.20

***
Report from 21.10.20
Plan
Future goals are:

-
-

Lessons learned

Next Meeting: 05.11.20

***