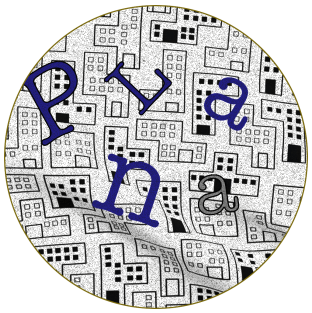
Bern University of Applied Sciences | BFH

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

December 2, 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
- the ability of create groups of modules and groups of teachers
- increased planning flexibility

An innovative type of this work is **product development.** Product development describes the process of creating product. Previous project and this work include all phases of creation: an analysis of new trends in technology, study of what the client wants, and idea how to create product based on selected technologies and concepts. As well as product implementation and testing. [1] The degree of novelty in this work includes:

- establishing successful ideas such as:
  - Involving teachers in the planning process, which reduces planning time and makes the process easier
  - Creation of groups of teachers and modules, making it easier to coordinate planning
- creation of a specific solution - product that includes great advantages over the existing product. These advantage are:
  - possibility of collaborative work. That is, all system actors can actively participate in the planning process.
  - consistency of the data
  - better overview of the entire system
  - accessibility anywhere, etc.

Below we will describe in details which model of the system can offer us such opportunities.

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 Microsoft 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+**[2] 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** Hyper Text 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 bi-weekly and sometimes 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

Table 2: Multi-column table

Multi-column	
X	X

Epic			
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.			
ID	**User Story Name**	**User Story**	**Acceptance Criteria**
7.0	list modules	As a Lecturer, I want to see the list of modules for a concrete semester, so that I can choose the modules I want to plan in my own plan.	User is able to: • navigate to his plan page • able to see the module list
5.0	add modules to my assignment plan	As a Lecturer, I want to be able to add some of the modules to my assignment plan I want to teach in a specific semester or remove it from my plan so that I can participate in the main planning by making suggestions or requests.	User is able to: • navigate to his plan page • able to select a module and set himself to it
08	manage my plan	As a Lecturer, I want to be able to manage my plan, so I can modify some data in my plan.	User is able to: • navigate to his plan page • able to select a module and set himself to it • able to remove himself from the module he has added himself • able to modify some data of his planning

Table 3: Product Backlog

Epic			
As a Study Director, I want to be able to make a group of lecturers and attach it to a specific module, and also make a 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			
ID	**User Story Name**	**User Story**	**Acceptance Criteria**
09	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 and make assignment plan.	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
10	Make 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.
11	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.
12	Attach lecturer/group of lecturers to the module or module group	As a Study Director, I want to be able to 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 a group of lecturer button • select lecturer or group of lecturer and save it

Table 4: Product Backlog

ID	User Story Name	User Story	Acceptance Criteria
13	Attach a group of lecturers to the group of modules	As a Study Director, I want to be able to 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 a group of lecturer button • select group of lecturer and save it

Table 5: Product Backlog

Epic			
As a Lecturer, who joins a group, I want to be able to independently manage the tasks related to the common module.			
ID	**User Story Name**	**User Story**	**Acceptance Criteria**
14	Manage common modules	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: • Open page with common modules • Manage his tasks related to him.

Table 6: Product Backlog

3.2 Functional Requirements

3.3 c. User Interface Requirements

Identifier	Priority Weight	Requirements
REQ-	1	UI must have a landing page (log in).
REQ-	10	UI must have study director start page.
REQ-	10	UI must have a page where the study director can make a new plan.
REQ-	10	UI must have a page to add a module.
REQ-	7	UI must have a page to load existing module.
REQ-	10	UI must have a page to create a new module .
REQ-	10	UI must have a page to add lecturers or group of lecturers to the module.
REQ-	7	UI must have a page with copied assignments from last year.
REQ-	10	UI must have a page to see the lecturer's assignments plan
REQ-	10	UI must have a page to add a module to the lecturer's assignment plan
REQ-	2	UI must have a lecturer's page to see the last year assignment plan
REQ-	5	UI must have a page to see the groups of lecturers to the corresponding module/module group
REQ-	10	UI must have a page to see and manage the plan after editing of lecturers

Table 7: Caption2

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.

4.1 a. Architecture Styles according business logic

Applications are designed to provide a variety of services to solve specific problems. And each problem has a list of rules (business rules). That is, the developer writes the code to implement the business rule. This code is business logic. There are various techniques and patterns for handling business logic. [3]

In this application we use a layered architecture. We use a pattern to make development of database access faster and to make code cleaner. The pattern we use here

based on the domain-driven design(DDD) pattern from Eric Evans. But in our case the business logic is not in the entity classes but in the Business Logic Service layer.

The following list shows the guidelines that build the business logic pattern of our application. **The guidelines that build the business logic**

- Low coupling between layers, high cohesion within them.
- Separation of concerns.
- Adaptability: be able to change.
- ???(still an deciding) Business logic layers contain no user interface and do not refer to user interface modules.
- No circular references between layers
- Lower layers should not depend on upper layers.

[4] The Figure below shows the structure of application's back-end.

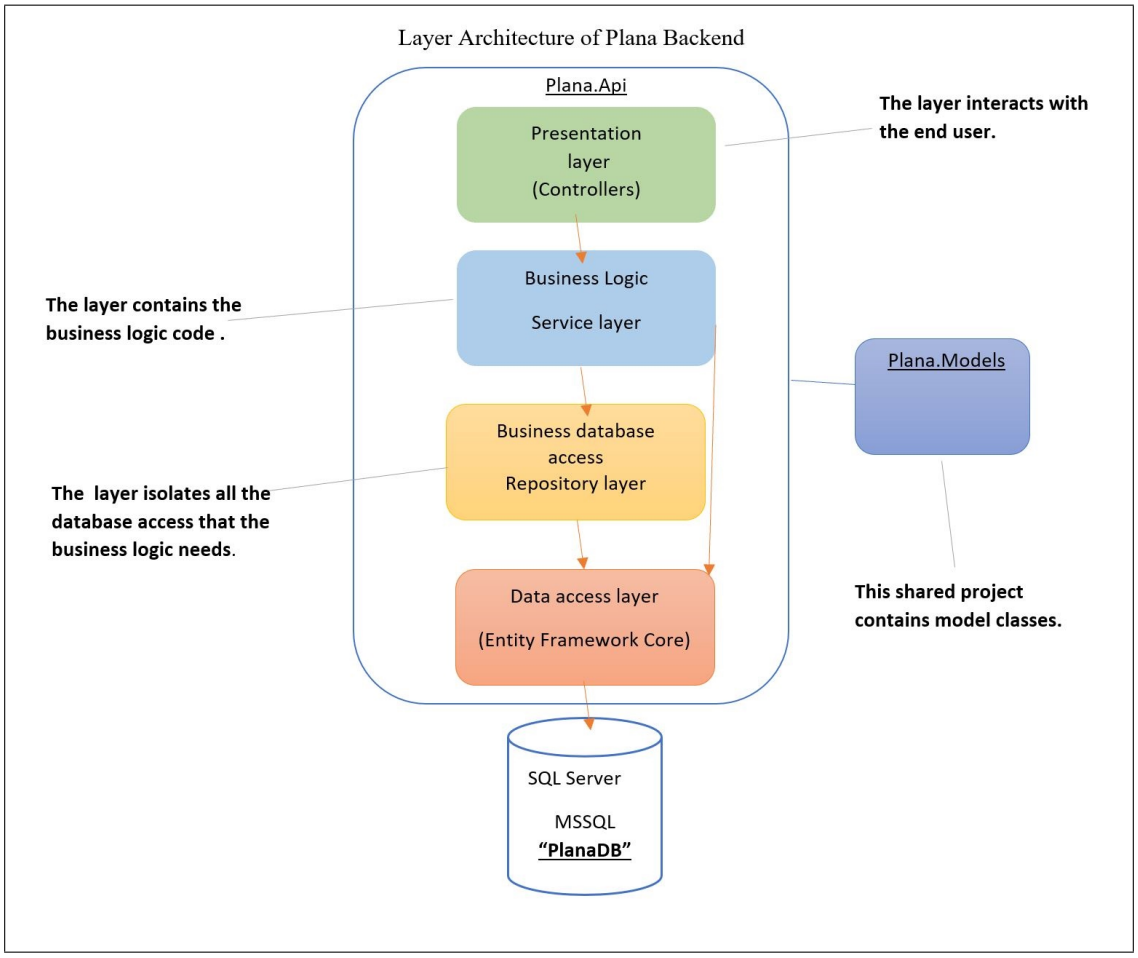


Figure 1: Layer Architecture of Plana Back-end

Layer Name	**Description**
Data access layer	The layer stores and retrieves the information from a database
Business database access layer	The layer performs database operation using EF Context object.
Business Logic Service layer	The layer makes logical decisions, processes commands
Presentation layer	The layer contains the code for the Web API endpoint logic. It contains a minimal business logic.[5]

Table 8: Layers Description

Layers shall to be loosely coupled and design dependency in only one direction. [6] [3]

4.2 The Plana application Architecture

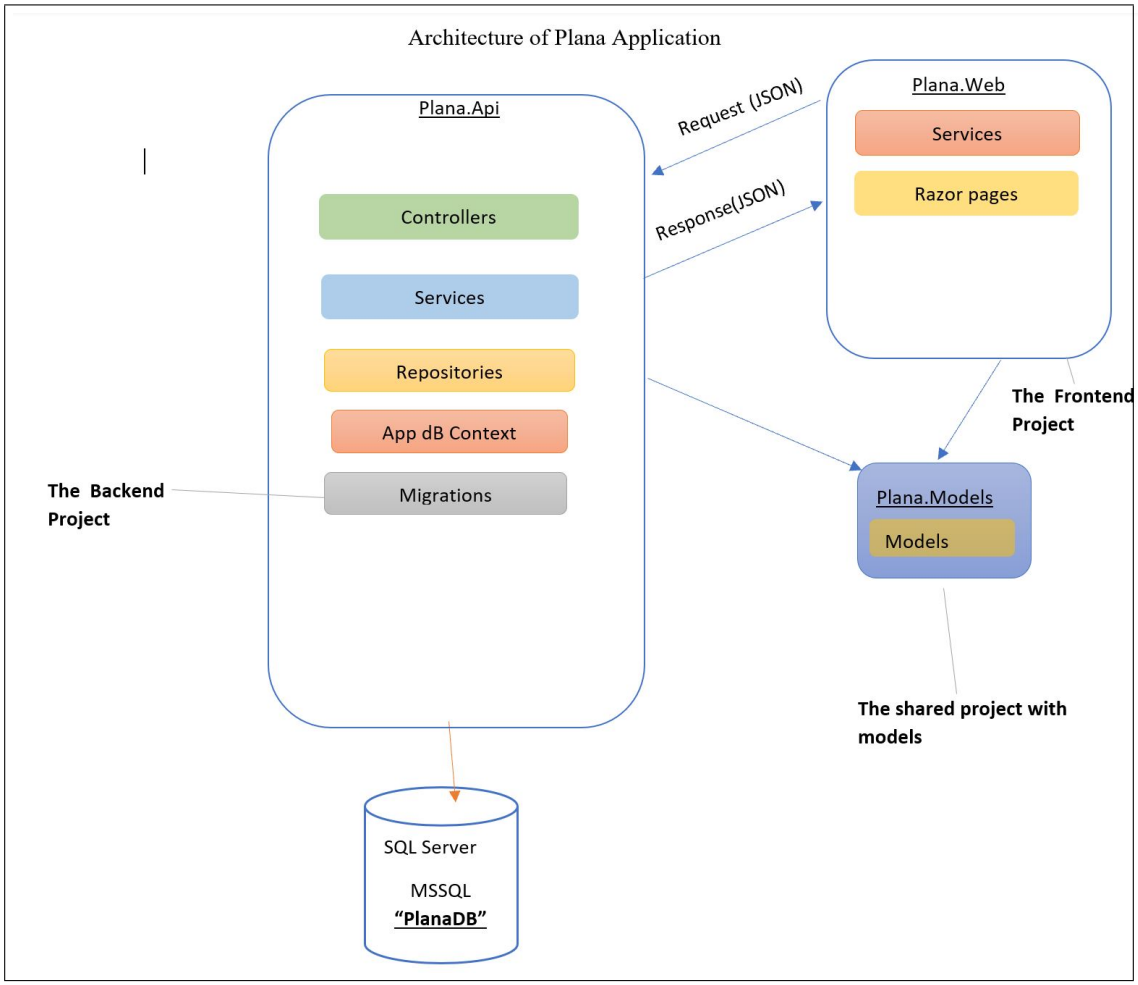


Figure 2: Layer Architecture of Plana Application

Components	**Description**
Plana.Api Project	
Controllers	..
Services	...
Repositories	...
App dB Context	...
Migrations	...
Plan.Web Project	
Services	..
Razor Pages	...
Plana.Models Project	
models	...
App dB Context	...
Migrations	...

Table 9: Plana Projects Components

5 User Interface Specification

*Note: only selected Interface designs are presented below because they are the main functions of our application.

5.1 a. Preliminary Design

i. Login
ii.Study director's dashboard
Once the study director successfully logged in, the personal dashboard page should be shown directly

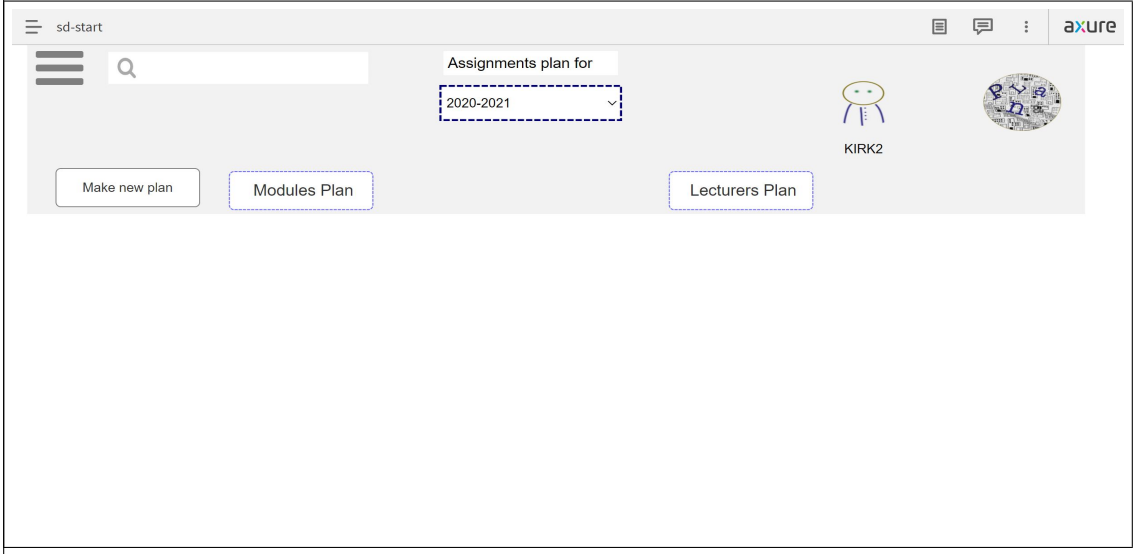


Figure 3: A boat.

iii.Create a plan

Once the personal dashboard is displayed, user could click on the ”Make a new plan” button, and add new module page will be shown where study director can make a new assignments plan.

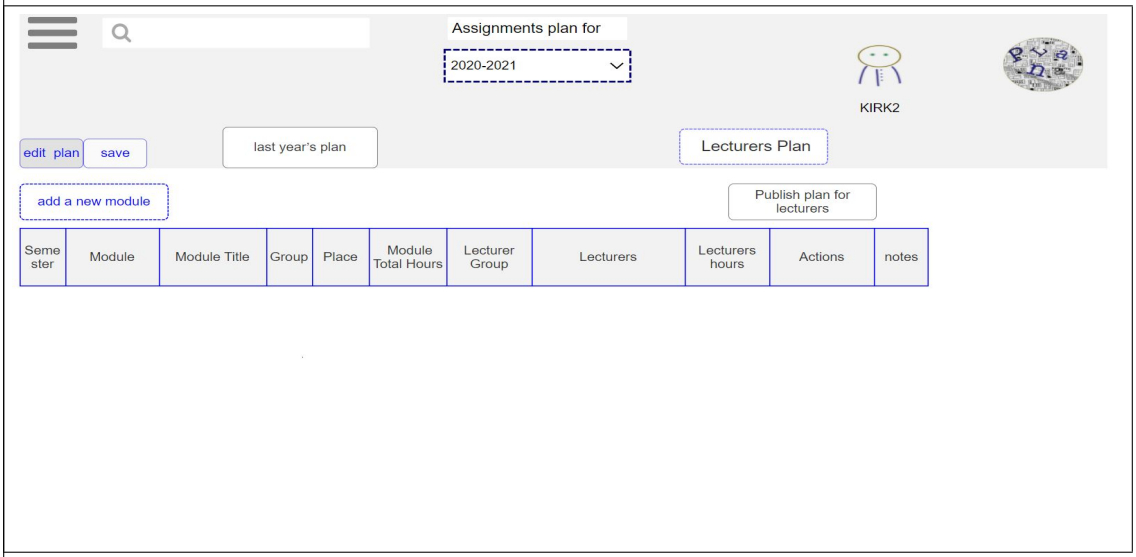


Figure 4: A boat.

iv. Add a new module

Study director could click on the ”add new module” button, and a window with buttons of adding or creating a module will be shown.

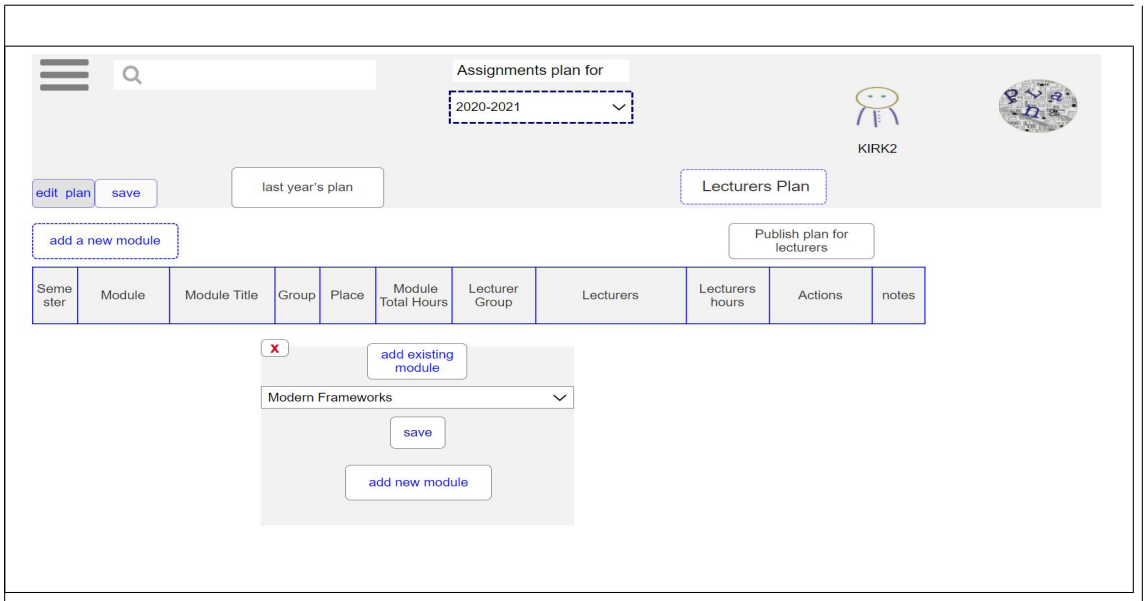


Figure 5: A boat.

v. Possibility of adding a module or group of modules.
Study director could click on the ”add an existing module” button, and a drop down menu with possible modules will be displayed, then he could choose one of them and click button ”save” and the chosen module will be displayed in the table. Or study director can choose ”add new module button” and the row with an editable fields will be displayed.

Figure 7: A boat.

vi. Possibility of adding a lecturer or group of lecturers.
Study director could click on the ”+” button that represents an add button and the windows with different buttons will be displayed. There are ”add existing group” button

to add an existing group of lecturers, ”add new group” button to create a new group of lecturers and it to the module/group of modules and the ”add lecturer” button to add lecturer.

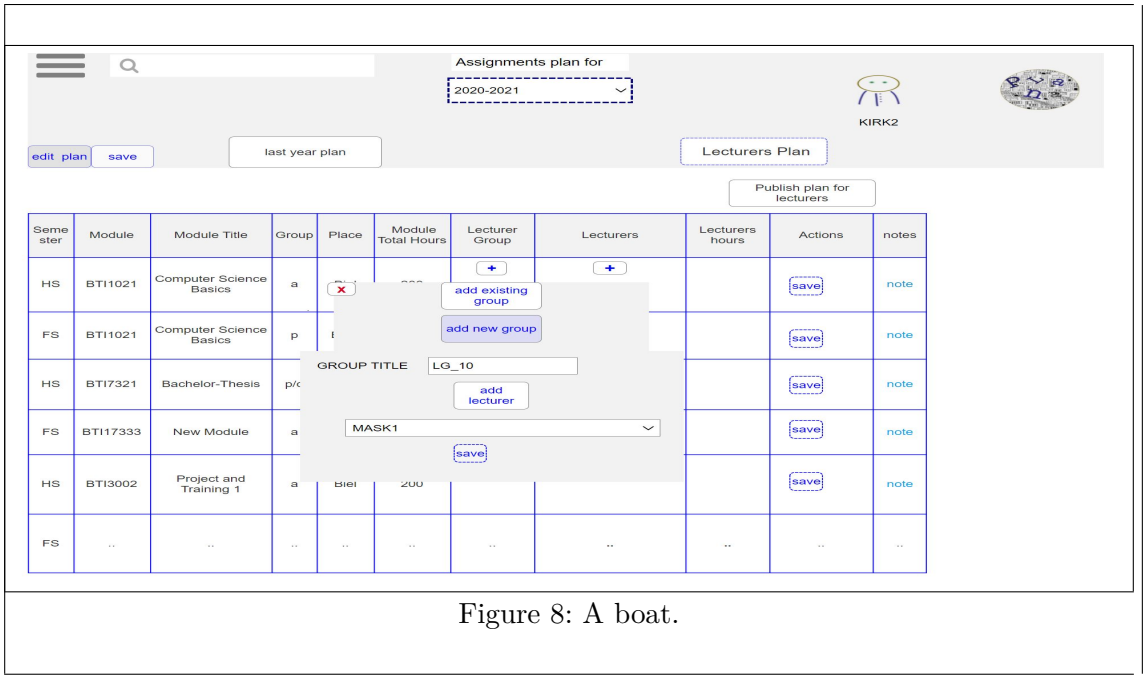


Figure 8: A boat.

vii. See and copy last year plan.

The study director could click on the button ”last year plan” to see the last year’s plan. Then he could copy the whole plan clicking the button ”copy whole plan to the actual plan” or he could choose the modules he likes and copy it to the actual plan clicking the button ”copy to the actual plan”.

Figure 10: A boat.

Study director can use this page to make a plan and could publish it for the lecturers by clicking the button ”publish on 18 July”.

viii. Lecturer's dashboard

Once the lecturer successfully logged in the dash board page is shown. And if the study director has already published the plan, the lecturer's assignment plan will be displayed. The dash board contains also buttons with different functions. Here the lecturer can manage his assignments.

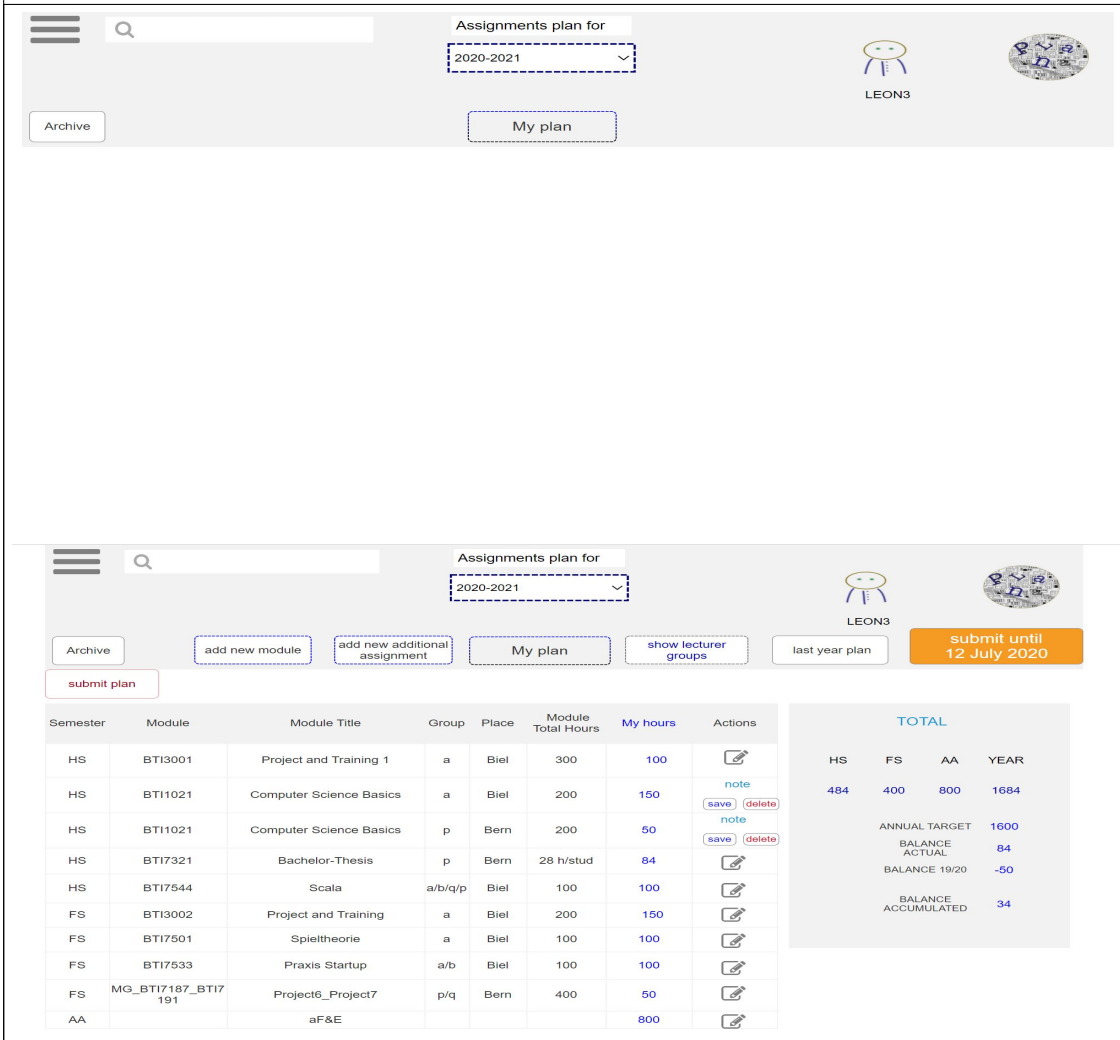


Figure 12: A boat.

ix. Add new module

Lecturer could click on the ”add new module” button and UI would show the window with possible modules. The lecturer could add him to the module and set desired hours for the module. The lecturer could click the button ”add new additional assignment” and the window with possible additional assignment will be displayed. And like with modules he can add desired assignment to his plan and set the hours.

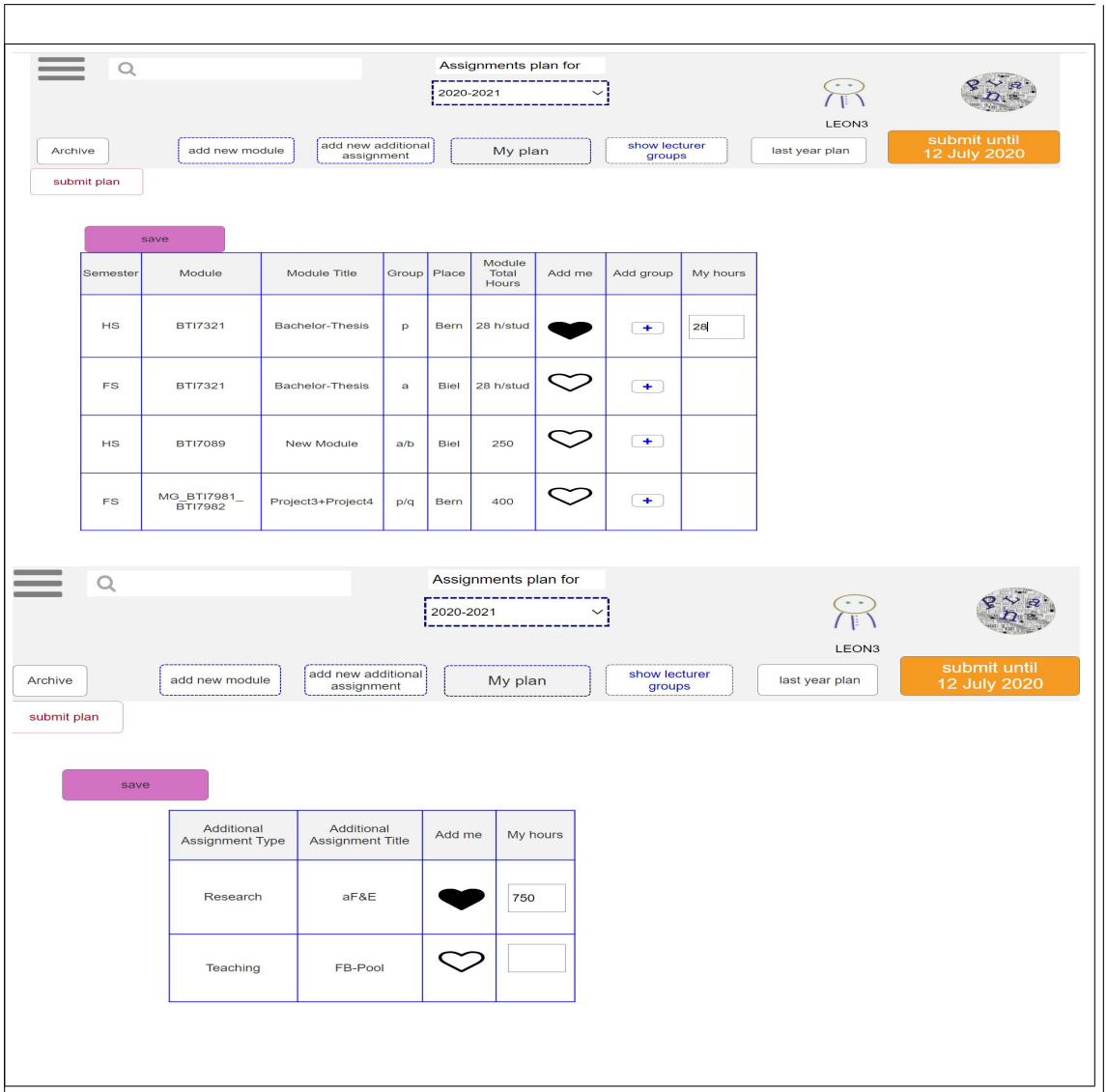
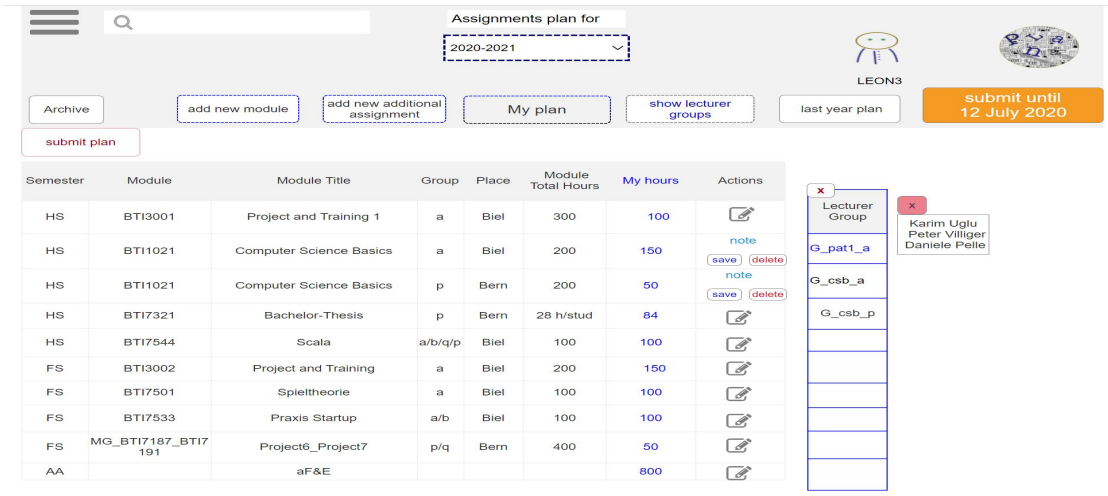


Figure 14: A boat.

x. Lecturers groups

Lecturer could click on the 'show lecturer groups' and the window with lecturers corresponding to the module will be displayed".



xi. Lecturer could click the link "note" and the note windows will be displayed.

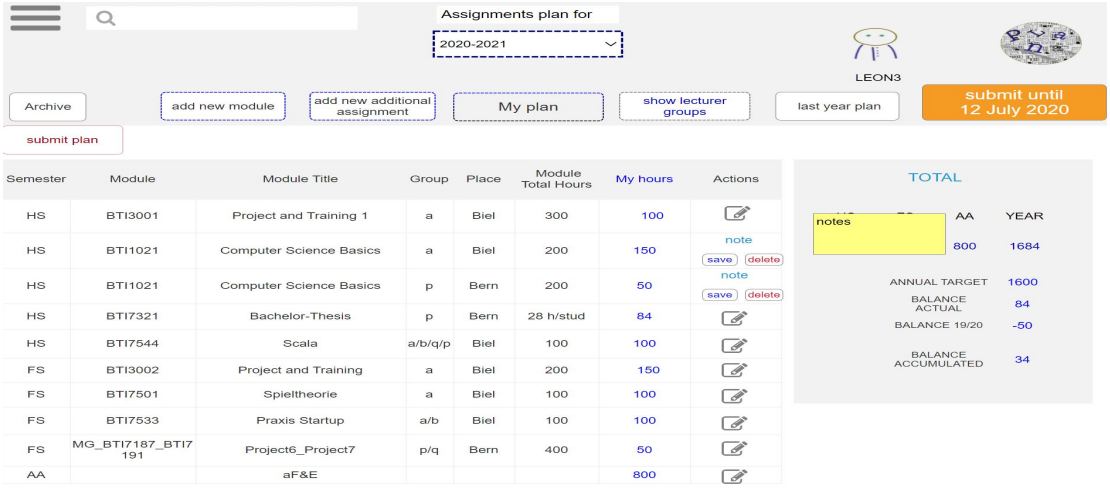


Figure 16: A boat.

xii. Lecturer's last year plan

Lecturer could click on the button "last year plan" and the window with the last year plan will be displayed.

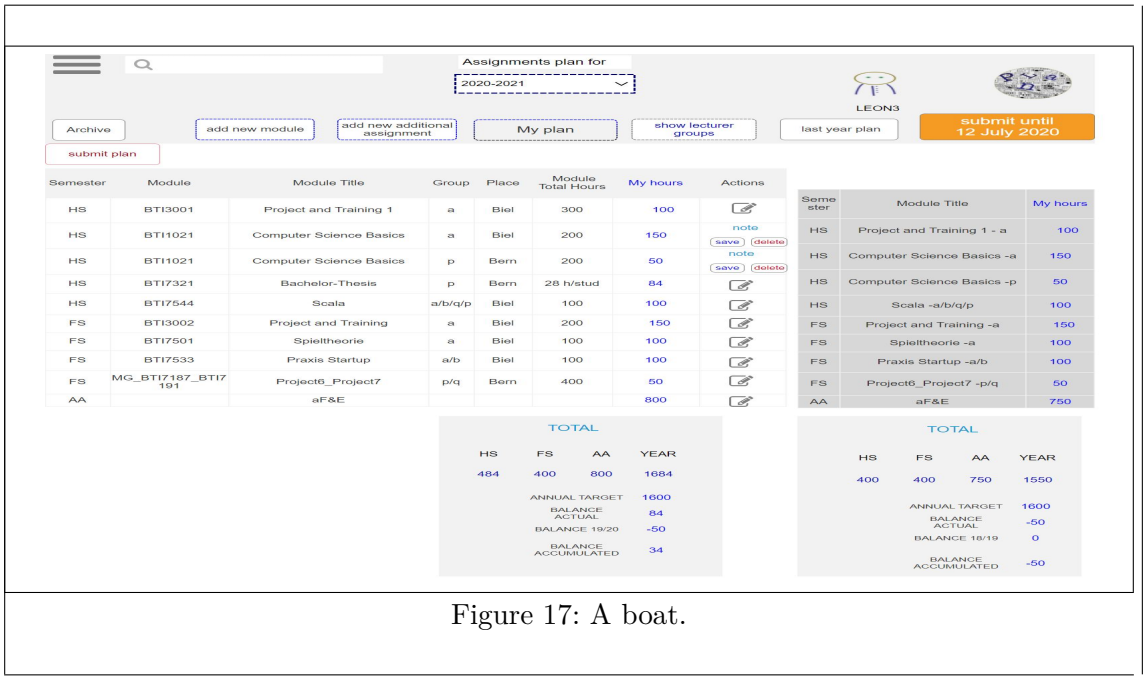


Figure 17: A boat.

xiii. Seeing the data after lecturers input.

The lecturer could submit plan by clicking the ”submit plan” button. After that the study director could see the page with the plan after lecturers editing by clicking ”Modules Plan” button. In this page he could manage the plan.

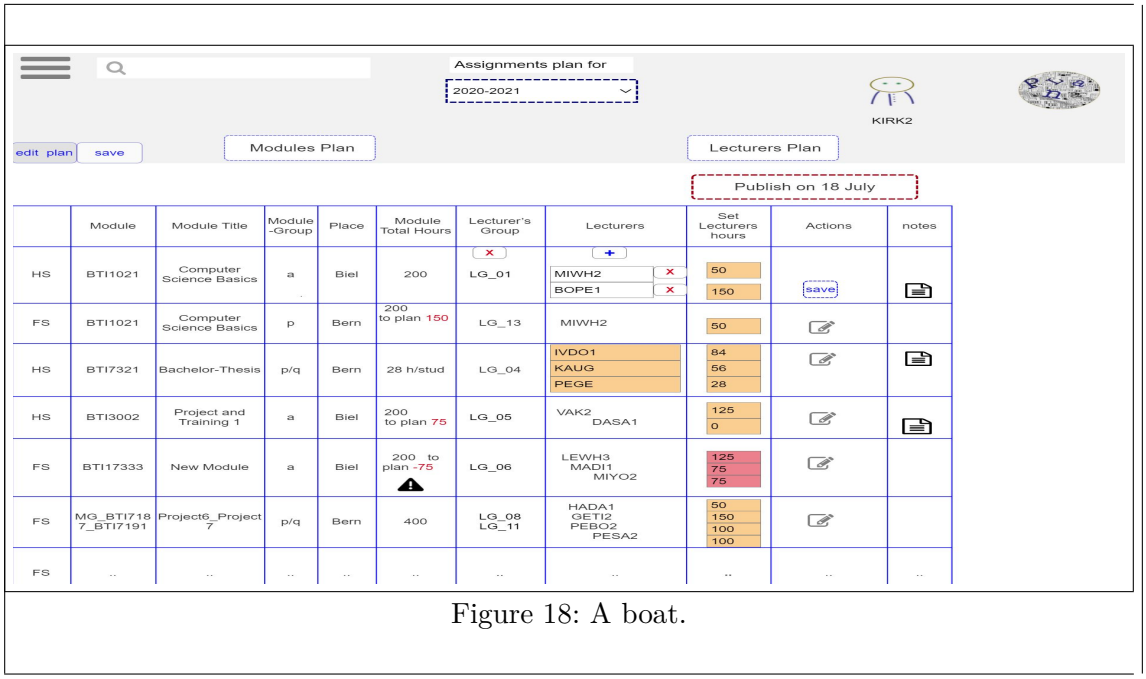


Figure 18: A boat.

xiv. Make a note

The study director could make a note clicking note link corresponding to the certain module.

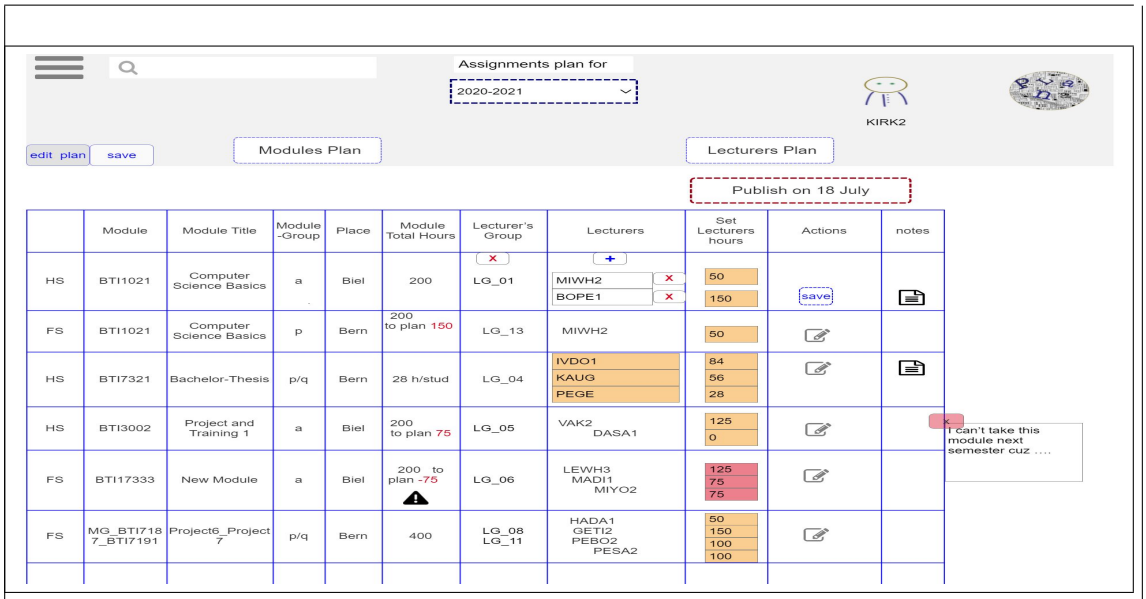


Figure 19: A boat.

6 Domain Analysis

6.1 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.

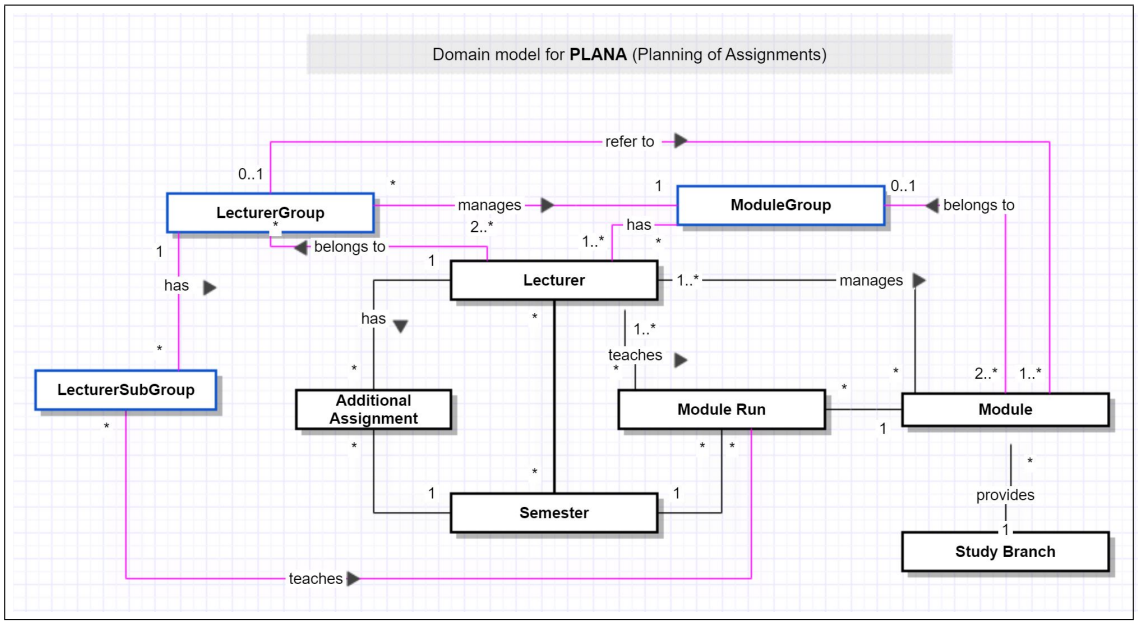


Figure 20: Domain Model for PLANA

In the domain model (Figure 2) we added small changes. When analyzing graphic concepts and the domain model, we found that it is necessary to add associations between the teacher and the teacher subgroup, since they are directly related.

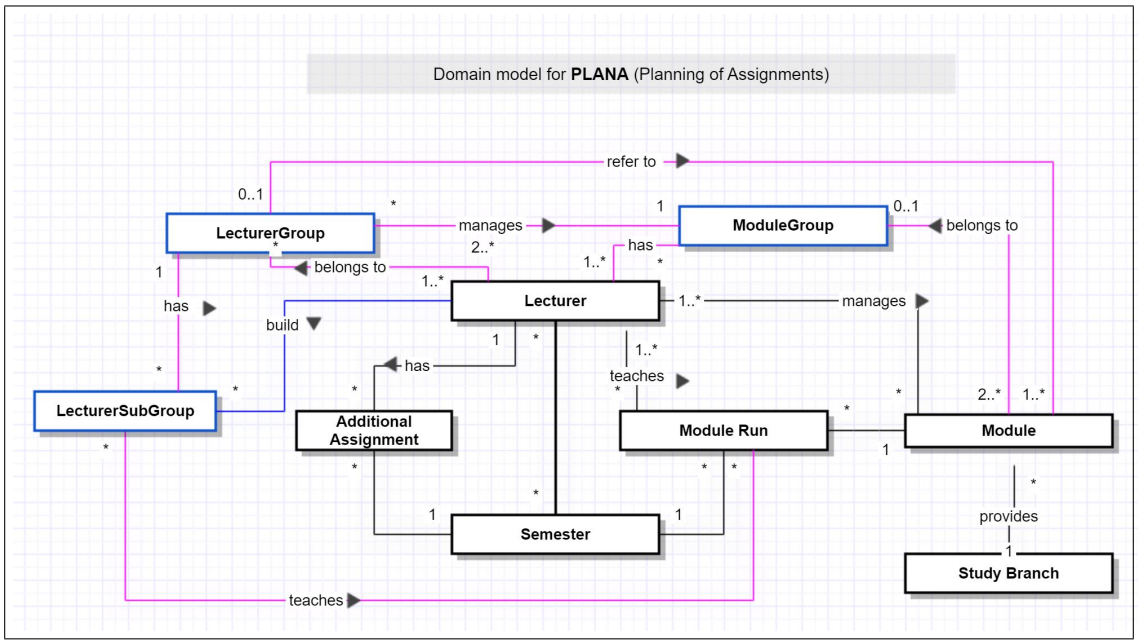


Figure 21: Domain Model for PLANA

The domain model (Figure 3) is the post-analysis Domain model. We have removed

the LecturerSubGroup since we can simplify the application. The LecturerGroup can consist of other LecturerGroups, that is, we create a relationship between them. Also we removed the links between the Module and Module Group and added it between the Module Run and the Module Group, since it is intended to create a groups of the Module Runs.

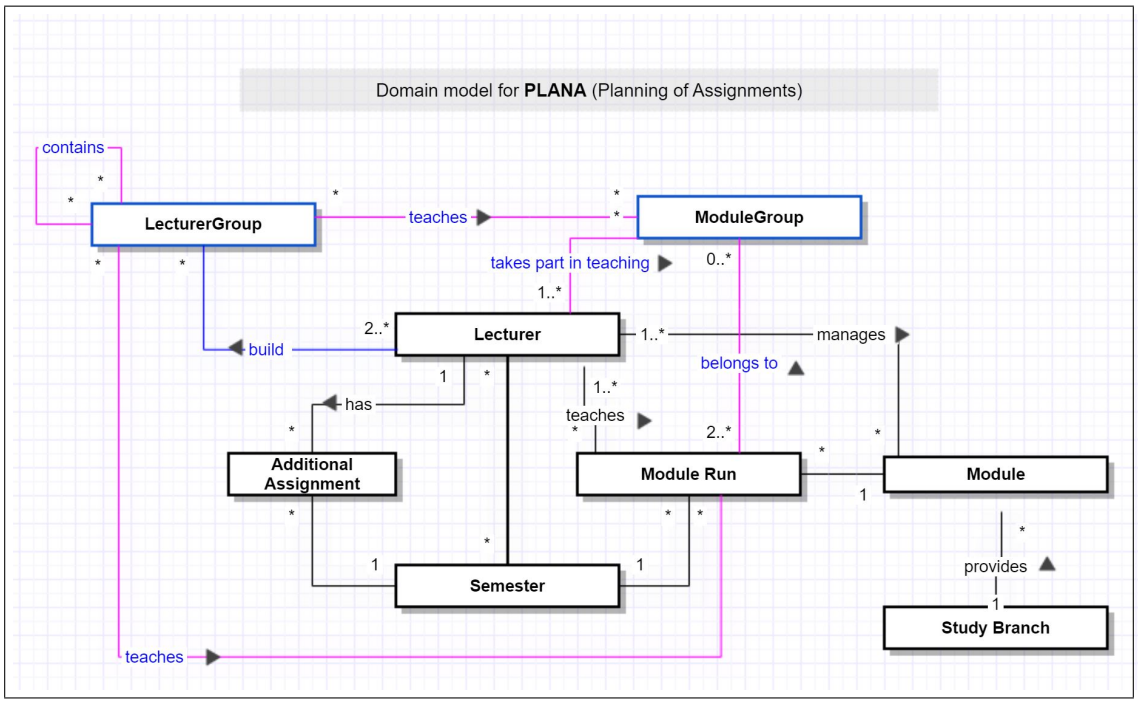


Figure 22: Domain Model for PLANA

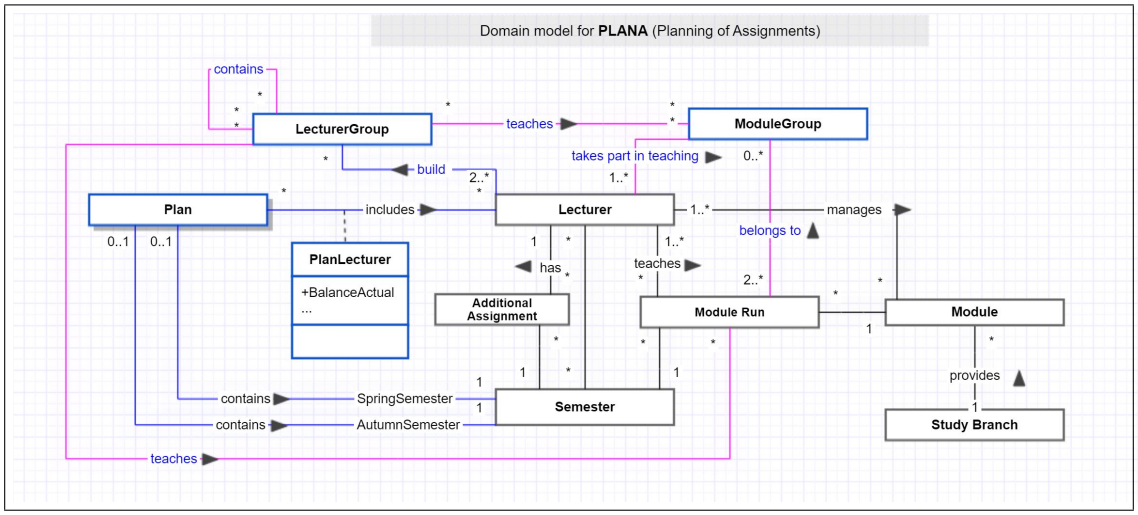


Figure 23: Domain Model for PLANA

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

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 program.
- 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. There are several types of the Additional Assignments. For example, teaching, research. Each teacher can plan his Additional Assignment and manage it.
- 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 teaching one or more modules.
- Concept class **ModuleGroup** models several modules collected in groups for further effective use.

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
Lecturer -> LecturerGroup	Each Lecturer build zero or multiple groups. Each LecturerGroup must consist of two or more lecturers.	build
LecturerGroup -> ModuleRun	Each LecturerGroup can have from zero to many ModuleRuns. Each ModuleRun can be teached by zero or multiple LecturerGroup	teaches
ModuleRun -> ModuleGroup	A ModuleRun can belong to zero or one ModuleGroup. ModuleGroup can have from two to many ModuleRuns.	belongs to
LecturerGroup -> ModuleGroup	A LecturerGroup can teach zero or multiple ModuleGroups. A ModuleGroup can have zero to many LecturerGroup. .	teaches
Lecturer -> ModuleGroup	Each Lecturer can take a part in teaching zero or many ModuleGroups. ModuleGroup can be teached by one or many Lecturers .	takes part in teaching
LecturerGroup -> LecturerGroup	Each LecturerGroup can consist of zero or multiple LecturerGroup and each LecturerGroup can be included in zero or multiple LecturerGroup .	consists

Table 10: Association Definition

7 Technologies

8 Creating the Projects

8.1 Structure of Projects and Folders

8.2 Data Models

8.3 Entity Framework Core Packages

8.4 Connection String

8.5 Creating the Database Context Class

8.6 Entity Framework Core Configuration

8.7 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 [7] [8]

8.8 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

8.9 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.[7]

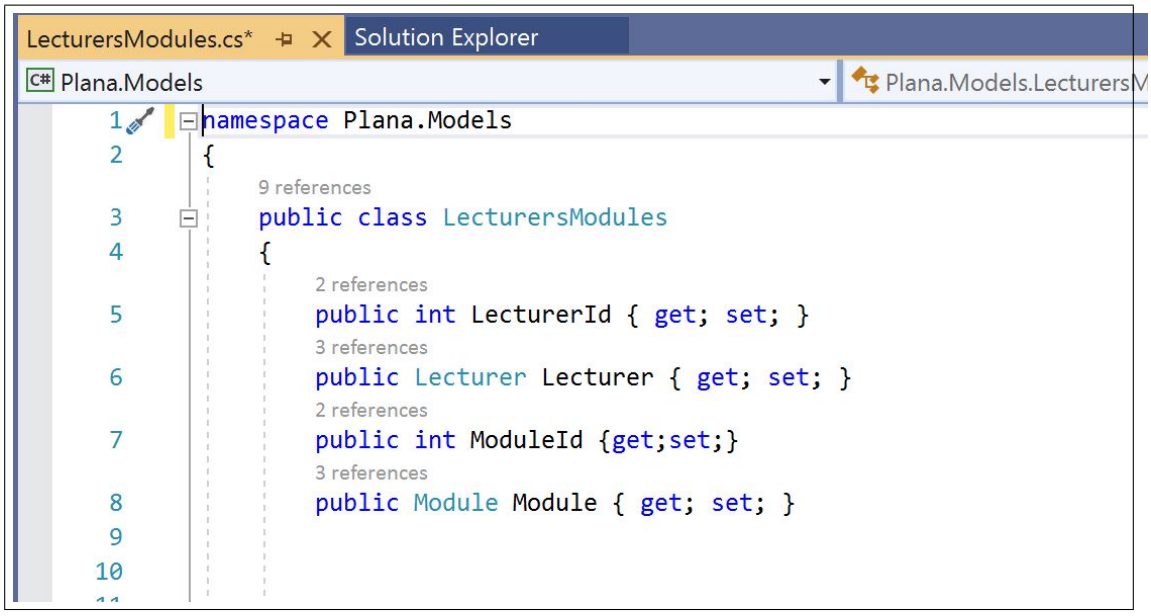


Figure 24: The LecturersModules entity class

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**;

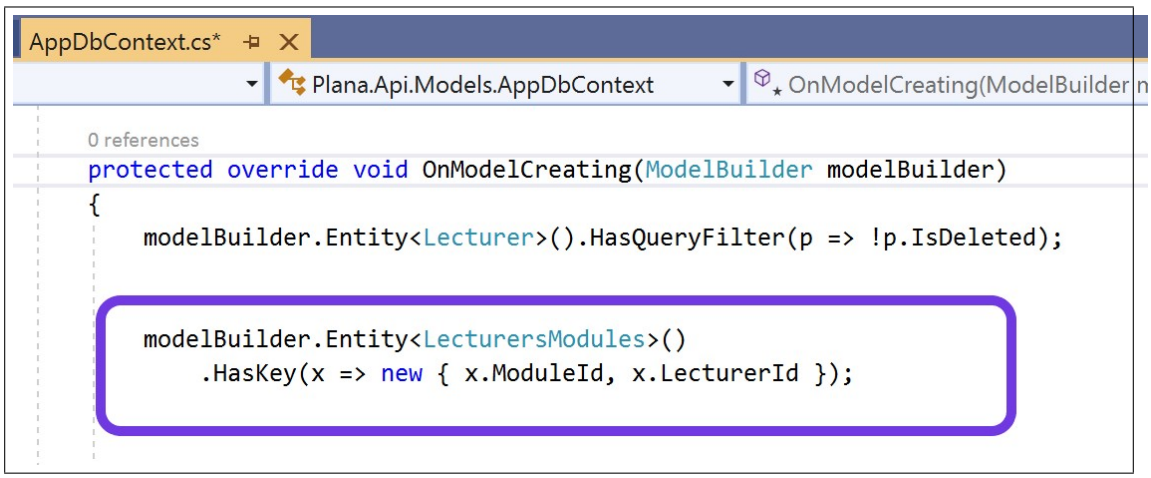


Figure 25: Adding Entity LecturersModules to the AppDbContext class

We need write just this and Entity Framework Core will do the correct implementation that we can see then in the migration files. [9]

```
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 26: 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.

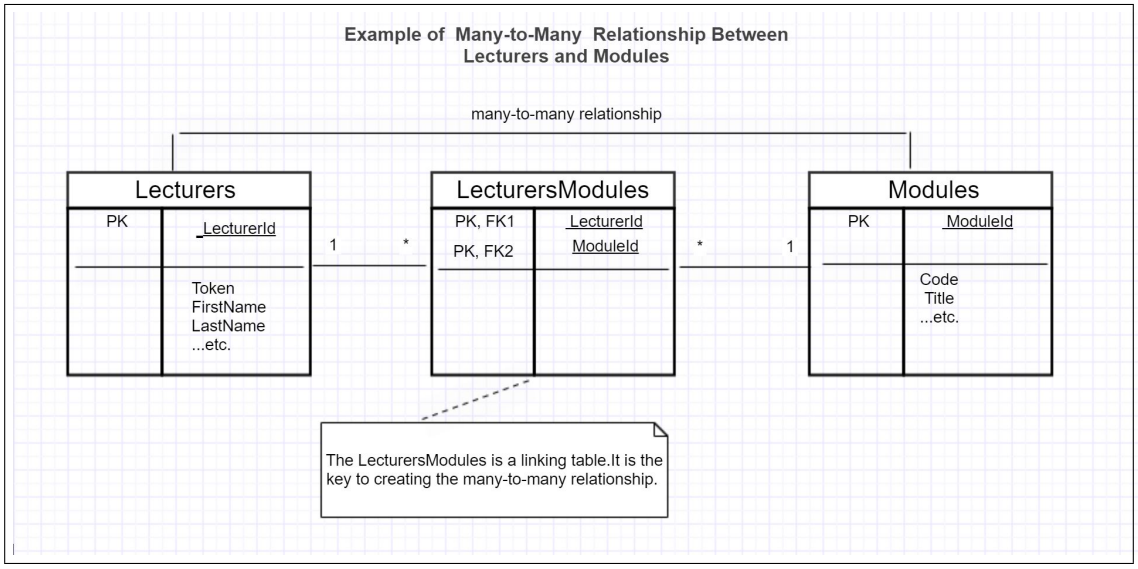


Figure 27: Creating Many-To-Many Relationship

8.10 Creating Database

8.11 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 28: 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 29: Creating Services in Startup.cs File

[10]

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** [10]

8.12 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 [10]. By default, Entity Framework Core uses cascade deletes for depend relationships with non-nullable foreign keys. [7]
..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 [7]. 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 30: Startup.cs in the Plana.Api project.Preparing Services and Middleware

8.13 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 31: 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 32

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

Figure 33: ...

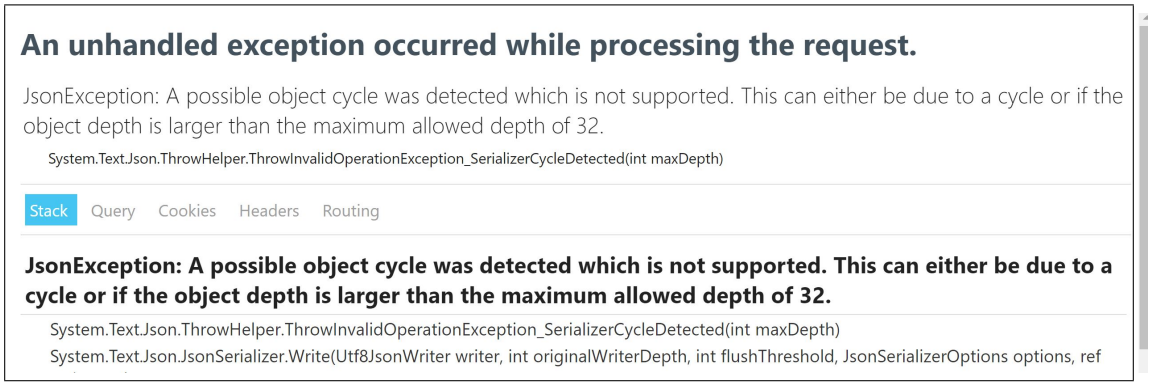


Figure 34

```
Package Manager Console

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

Figure 35

```
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 36: ...


```
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 37

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 38: 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 39: 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.

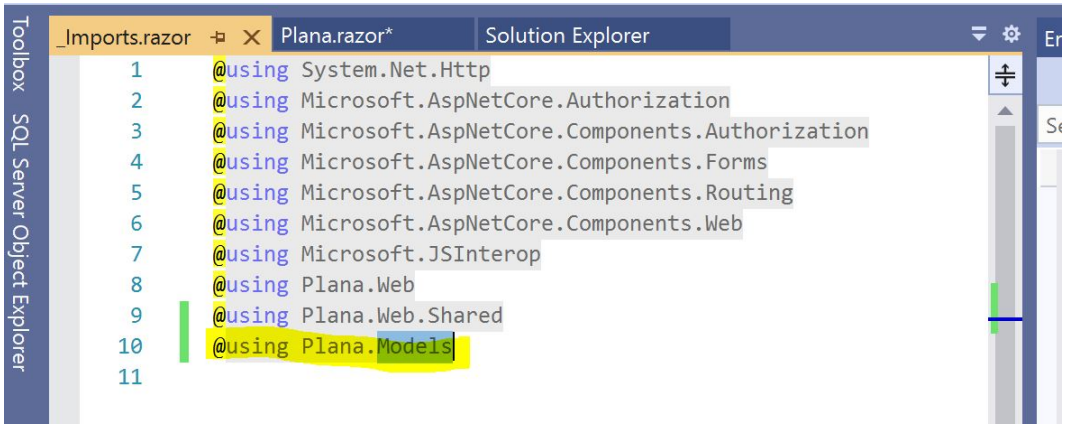


Figure 40: Adding required namespaces to the _Imports.razor file in Plana.Web project folder

8.14 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.

9 Testing

10 Summary

10.1 Conclusions

10.2 Future Work

10.3 Lessons Learned

11 List of illustrations

12 Contents of the table

13 Appendix

14 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:

References

[1] D. J.-U. Meyer, *Product development.* [Online]. Available: https://innolytics-innovation.com/product-development/ (visited on 11/23/2020).

[2] P. Eeles, "Capturing architectural requirements," *IBM Rational developer works*, 2005.

[3] J. P. Smith, *Entity Framework core in action.* Manning Publications Co., 2018.

[4] M. Flower, *Layeringprinciples.* [Online]. Available: https://martinfowler.com/bliki/LayeringPrinciples.html (visited on 01/07/2015).

[5] C. Woodruff, *Advanced architecture for asp.net core web api.* [Online]. Available: https://www.infoq.com/articles/advanced-architecture-aspnet-core/ (visited on 06/01/2018).

[6] E. Evans, *Domain-Driven Design: Tackling Complexity in the Heart of Software.* Addison-Wesley, 2004.

[7] J. P. Smith, *Entity Framework in Action.* 2018, ISBN: 9781617294563.

[8] Microsoft.com, *Entity framework core.* [Online]. Available: https://docs.microsoft.com/en-us/ef/.

[9] P. God, *Many-to-many relationship with entity framework core.* [Online]. Available: https://dev.to/_patrickgod/many-to-many-relationship-with-entity-framework-core-4059 (visited on 11/23/2020).

[10] A. Freeman, *Pro ASP.NET Core 3.* 2020, ISBN: 978-1-4842-5439-4.

15 Protocol

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

Agenda

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

Report from 24.09.20
Plan
Future goals are:

-
-

Retrospective

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

-
-

Retrospective

Next Meeting: 21.10.20

***
Report from 21.10.20(13:30)
Plan
Future goals are:

- Implementation. Add new concept classes into the application.
- Implementation. Start implementation of graphic concepts we've made.
- Make Study director plan view concept with less elements.

- Put UI concepts to the report.

Retrospective

- In graphical concept i have choice between semester view and year view, but for planning is more important year view.
- Actual state of planing view for study director looks little bit heavy because of many elements in it, would be better to minimize some staffs. In implementation we can manipulate view hiding some part of them.
- Additional Assignments have a category, description and number of hours.

Next Meeting: 04.11.20 (13:30)

***

Report from 04.11.20

Plan

Future goals are:

- **report** Describe the Additional Assignments. There are several Arts of Additional Assignments. The Lecturer and also Study Director can plan and manage it.
- **report** Describe what is innovative in this work. For example that Lecturer, Study Director and Institute Manager can active collaborate with each other. That is idea of making groups is also innovative. In general, we make a specific solution for problem that we face.
- Put tasks which have to be done until **21-January 2021** into the sprint backlog. These are
  1. Report
  2. Final Presentation
  3. Book
  4. Video
- **Application** Implement the views that we made with Axure tools.

Retrospective

- The graphical concepts looking better now.
- Conflict between number of hours and lecturers that can teach specific module have place in application.

Next Meeting: 18.11.20 (13:30)

***

Report from 18.11.20

Plan

Future goals are:

- Add business logic into separated project.
- Test it.
- Add layer diagram

Retrospective

- It is important to add the date to the bib file in case i use the article from the internet.

Next Meeting: 02.12.20)

***

Report from

Plan

Future goals are:

-

Retrospective

-

Next Meeting:)