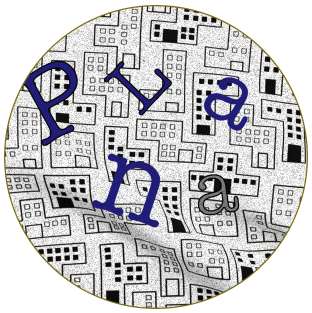
Bern University of Applied Sciences | BFH

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

January 19, 2021

Contents

Acknowledgments

Abstract

PLANA (Planning of Assignments for Lecturers) is a web application tool for effective teacher planning. With PLANA, teachers can participate in the planning process, make suggestions about the modules which they teach, and schedule hourly workloads for specific modules. In this thesis, we improve the planning of assignments for lecturers by implementing a web application using ASP.NET Core Blazor. We also describe the application's system architecture and system design and perform domain analysis. Additionally, we justify our choice of implementing PLANA as a web application by exploring the advantages of web applications over Microsoft Excel solutions and desktop applications.

1 Introduction

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

- The teachers themselves should be involved in the planning process
- The ability to create groups of modules and groups of teachers
- Increased planning flexibility

This thesis also employs an innovation technique called **product development.** The previous project (project2) [1] and this thesis include all phases of product creation.[1]

- An analysis of new trends in technology
- A study of the client's needs
- An idea of how to create the product based on the selected technologies and concepts
- Product implementation and testing

This thesis contains the following novelties.

- 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 - a product that includes great advantages over the existing product. These advantages are:
  - Possibility of collaborative work, i.e. all system actors can actively participate in the planning process.
  - Data consistency
  - A better overview of the entire system
  - Accessibility regardless of location, etc.

[1] Project 2 is a project module of the Bern University of Applied Sciences, in which students often deal with the topic of the Bachelor's start working.

The planning process involves the input of specific data for specific user-defined views for each user, as well as time limits set by the system.

All of these requirements need a more suitable system than Microsoft Excel. In the previous project, we compared a **Microsoft Excel** solution to a generic **Web Application** with respect to several criteria and 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 **data consistency**
- Also, the data is much **safer** in a database
- The web application gives the best **overview** of the entire system

Comparing a generic **Desktop Application** to a generic **Web Application** also results in the web application being the better option:

- **It is accessible from anywhere**
- **No updates required**
- **Costs less**

We decided to make a web application that will meet all the requirements and will be created using suitable modern technologies. ASP.NET Core Blazor Server with Entity Framework Core (EFCore) and MS SQL (Microsoft Structured Query Language) was the database technologies of choice. ASP.NET Core Blazor is a new framework that is gaining popularity. A major advantage of using ASP.NET Core Blazor is that it becomes possible to do the entire application in C# without using JavaScript.
This work is a continuation of project 2 [2], which was completed last semester, where we have prepared the necessary environment for this project by creating a prototype. In this thesis, the system will be 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.

[2]Project 2 is a project module of the Bern University of Applied Sciences, in which students often deal with the topic of the Bachelor's Start working.

  - Each teacher can make a group with other lecturers.

- **Grouping of the modules.** The group of lecturers can choose a group of modules and assign 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.

This thesis has the following structure. First, we will explain how the project was organized and how SCRUM was used to manage it. Then, we will carry out an additional analysis of the system pertaining to the expansion of system requirements. We will also make changes to the domain analysis and extend the System Architecture and System Design chapters. Finally, we present the Project Implementation and Testing.

1.1 Acronyms

Acronyms	Words
EF	Entity Framework
CSS	Cascading Style Sheet
KKK	345

Table 1: Caption2

1.2 Glossary

API Application Programming Interface.

ASP.NET Core API Is a free and open-source web framework and successor to ASP.NET, developed by Microsoft and the community. It is a modular framework that runs on both the full .NET Framework, on Windows, and the cross-platform .NET Core.

ASP.NET Core 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.

Azure Cosmos DB is a fully managed NoSQL database for modern app development.

Backend refers to any part of a website or software program that users do not see.

BPMN Business Process Model and Notation

CRUD Create, read, update and delete

C# is a general-purpose, multi-paradigm programming language encompassing static typing, strong typing, lexically scoped, imperative, declarative, functional, generic, object-oriented (class-based), and component-oriented programming disciplines.

Database Database is an organized collection of data, generally stored and accessed electronically from a computer system

DBContext DbContext is an important class in Entity Framework API. It is a bridge between domain or entity classes and the database. DbContext is the primary class that is responsible for interacting with the database

DTO Data Transfer Object

Entity Framework(EF)Core It is open-source, lightweight, extensible, and a cross-platform version of Entity Framework data access technology.

Frontend. The Frontend of a software program or website is everything with which the user interacts.

- **FURPS+**[2] is a system for classifying requirements.
  - Functionality
  - Usability
  - Reliability

  - Performance
  - Supportability

- **HTML** Hyper Text Markup language
- **MS SQL** Structured Query Language
- **JS** JavaScript
- **JSON**(JavaScript Object Notation)is an open standard file format, and data interchange format, that uses human-readable text to store and transmit data objects consisting of attribute-value pairs and array data types (or any other serializable value).
- **Microsoft Teams** is a proprietary business communication platform developed by Microsoft.
- **Microsoft SQL Server** Microsoft SQL Server is a relational database management system (RDBMS) that supports a wide variety of transaction processing, business intelligence and analytic applications in corporate IT environments
- **MySQL** is an open-source relational database management system.
- **Microsoft Visual Studio** is an integrated development environment (IDE) from Microsoft. It is used to develop computer programs, as well as websites, web apps, web services, and mobile apps.
- **NuGet** Is a free and open-source package manager designed for the Microsoft development platform.
- **PostgresSQL** is a free and open-source relational database.
- **REST** Representational state transfer (REST) is a de-facto standard for a software architecture for interactive applications that typically use multiple Web services. To be used in REST-based application, a Web Service needs to meet certain constraints; such a Web Service is called RESTful.[3]
- **Product Backlog** is a prioritized list of work for the development team that is derived from the roadmap and its requirements.
- **Relational Database** is a digital database based on the relational model of data.
- **SCRUM** is a process framework used to manage product development
- **SQLite** is a server-less database and is self-contained.
- **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.

- **Stakeholder** The term s̈takeholderr̈efers to the people or groups affected by a software development project. They may be end users, or they might simply be affected by the process.
- **UI** User Interface is the space where interactions between humans and machines occur.

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. To control the hours allotted to work on the project we used Microsoft Excel. At the end of the project, by the time the documents were submitted the total working time was 400 hours. Together with the subsequent preparation for the defense and presentation, the total time will be from 420 to 450 hours.

2.2 Scrum

The foundation of the project organization was Scrum [3]. 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 everyone 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 to 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.

i. Scrum Roles

The Scrum Roles have been defined as follows:

[3] Scrum is an agile framework for developing, delivering, and sustaining complex products.

- Product Owner: Mr. Pfahrer acts as the owner of the product. He is also responsible for feedback on the work and overseeing the work process.
- Development Team: Since there is only one developer, this role is assigned to Shiryagina Kristina.
- Scrum Master: Shiryagina Kristina takes on this role.

The Thesis expert Federico Flueckiger takes a part in the meetings thus has the role of stakeholder.

ii. 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.

iii. Scrum Artifacts

[4]

Gitlab[5] from the BFH school was used for the project, which makes it easy to figure out what has already been done and what else is in the process of work, as well as make plans for sprints. The figures below show that with the help of Gitlab the Product Backlog and Sprint backlog [6] was created. **Product Backlog** contains everything that the product needs. **Sprint Backlog** is a set of items selected from Product backlog for working on concrete sprint. For each task there is a label "TODO", "DOING" and "DONE", which facilitates the overall control of the work. We created document(PLANA_project_management) in Excel format, that can be found under the following link `https://gitlab.ti.bfh.ch/shirk1/planning-of-the-assignments-for-lecturers-plana/-/tree/master/3project_management`, which was used to organize this project. This document contains all data on the deadlines, goals of the sprints, duration of sprints, as well as tasks for a specific sprint, description, duration, and their prioritization.

[4] Scrum Artifacts provide key information that the Scrum Team and the stakeholders need to be aware of for understanding the product under development, the activities being planned, and the activities are done in the project.

[5] GitLab is an open-source code repository and collaborative software development platform for projects.

[6] A sprint backlog is the set of items that a cross-functional product team selects from its product backlog to work on during the upcoming sprint.

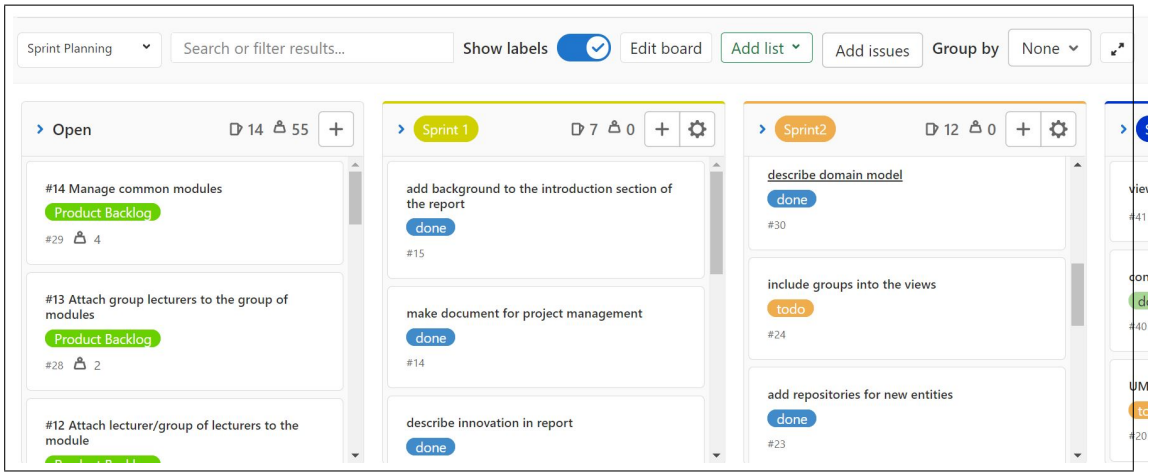


Figure 1: Gitlab boards with issues

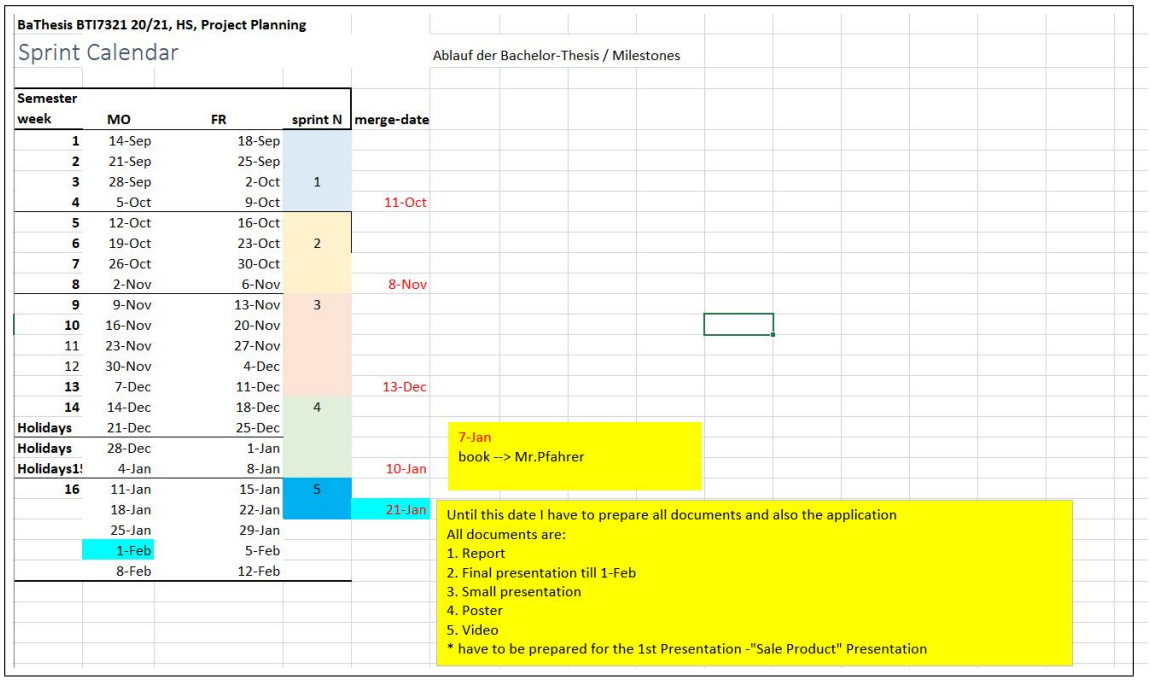
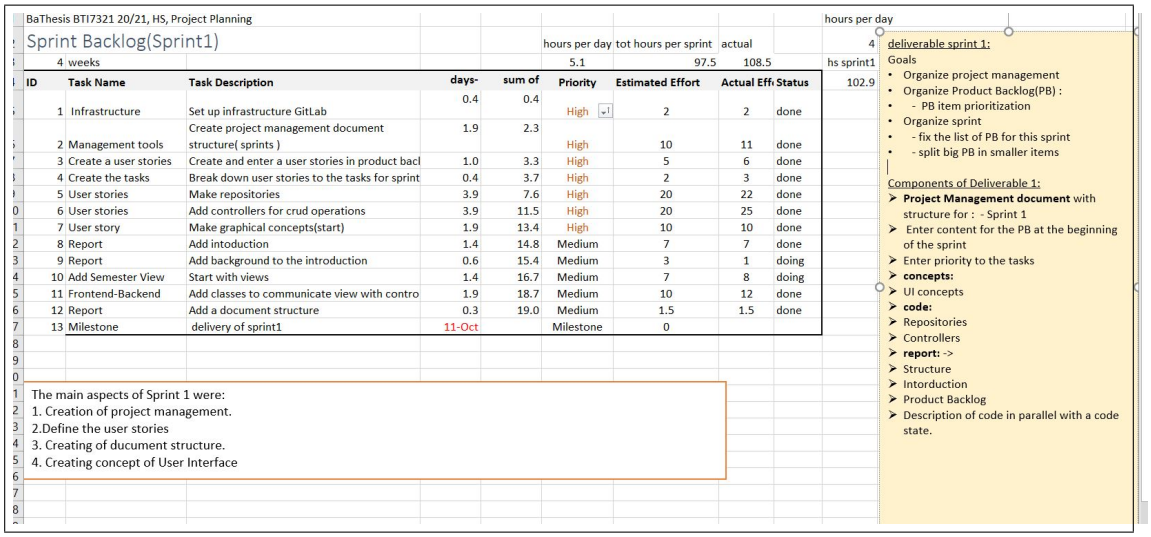


Figure 2: Sprints Calendar

iv. Sprints

Conclusion Project Management

If to look closely at each sprint in the document `https://gitlab.ti.bfh.ch/shirk1/planning-of-the-assignments-for-lecturers-plana/-/tree/master/3project_management"`, it is easy to see that some tasks have been moved from one sprint to the next. This



BaThesis BTI7321 20/21, HS, Project Planning

Sprint Backlog(Sprint1)

	4	weeks				hours per day	5.1	tot hours per sprint	97.5	actual 108.5
ID		Task Name	Task Description	days-	sum of	Priority	Estimated Effort	Actual Eff	Status	
	1	Infrastructure	Set up infrastructure GitLab	0.4	0.4	High	2	2	done	
	2	Management tools	Create project management document structure(sprints)	1.9	2.3	High	10	11	done	
	3	Create a user stories	Create and enter a user stories in product bacl	1.0	3.3	High	5	6	done	
	4	Create the tasks	Break down user stories to the tasks for sprint	0.4	3.7	High	2	3	done	
	5	User stories	Make repositories	3.9	7.6	High	20	22	done	
	6	User stories	Add controllers for crud operations	3.9	11.5	High	20	25	done	
	7	User story	Make graphical concepts(start)	1.9	13.4	High	10	10	done	
	8	Report	Add intoduction	1.4	14.8	Medium	7	7	done	
	9	Report	Add background to the introduction	0.6	15.4	Medium	3	1	doing	
	10	Add Semester View	Start with views	1.4	16.7	Medium	7	8	doing	
	11	Frontend-Backend	Add classes to communicate view with contro	1.9	18.7	Medium	10	12	done	
	12	Report	Add a document structure	0.3	19.0	Medium	1.5	1.5	done	
	13	Milestone	delivery of sprint1	11-Oct		Milestone	0			

The main aspects of Sprint 1 were:
1. Creation of project management.
2.Define the user stories
3. Creating of ducument structure.
4. Creating concept of User Interface

hours per day 4
hs sprint1 102.9

deliverable sprint 1:
Goals
- Organize project management
- Organize Product Backlog(PB) :
  - PB item prioritization
- Organize sprint
  - fix the list of PB for this sprint
  - split big PB in smaller items

Components of Deliverable 1:
- **Project Management document** with structure for : - Sprint 1
- Enter content for the PB at the beginning of the sprint
- Enter priority to the tasks
- **concepts:**
- UI concepts
- **code:**
- Repositories
- Controllers
- **report:** ->
- Structure
- Intorduction
- Product Backlog
- Description of code in parallel with a code state.

Figure 3: Organization of Sprint one

suggests that not all tasks were completed on time, that is, first of all, attention was paid to tasks with high importance. This is since some tasks may vary significantly over time. therefore, the prioritization system is in demand in this case. There was a noticeable delay during the implementation of the code for the program, which I could not solve for several days, in this case, I turned to a colleague for help, who explained to me what I needed to change in the configuration. And I think that school projects are always associated with a risk that they will not meet the deadline, since there is a constant study of new technologies, and working alone is always inferior to teamwork. On the other hand, dividing work into sprints allows you to have more clear control over the overall situation.

3 System requirements

3.1 Product Backlog and User Stories

In project 2, the main user stories were identified, in this work we supplemented this list in accordance with the new requirements.

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 modify some data of his planning

Table 2: 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	**Story Name**	**User Story**	**Acceptance Criteria**
09	See the requests for the modules	As a Study Director, I want to see proposals for the selected modules so that it will be easier to make an assignment plan.	User is able to: • open a list of modules • see the lecturers requests
10	See the requests for the groups	As a Study Director, I want to see teachers' suggestions for the group work so that it will be easier to approve specific groups and make an assignment plan.	User is able to: • open a list of lecturers group • see the changes made by lecturers
11	See the working hours request for the concrete modules	As a Study Director, I want to see teachers' proposals for working hours for the concrete module so it will be easier to set working hours for each lecturer.	User is able to: • open a plan • see the hours requests
12	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.
13	Set groups of lecturers to the modules	As a Study Director, I want to be able to set a group of lecturers to the assignment plan.	User is able to: • open the main planning page • select the group of lecturers and attach it to a specific module.
14	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.

Table 3: Product Backlog

ID	User Story Name	User Story	Acceptance Criteria
15	Attach Lecturers to the module	As a Study Director, I want to be able to attach a lecturer or group of lecturers to the specific module in the assignment plan.	User is able to: • open the main planning page • select lecturer • click add lecturer button and save it to the module
16	Attach Lecturers to the module group	As a Study Director, I want to be able to attach a group of lecturers to the specific module in the assignment plan.	User is able to: • open the main planning page • select and add lecturer to the group of modules and save it
17	Attach Lecturers Group to the Module Group	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 and add lecturers group to the group of modules and save it

Table 4: 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 5: Product Backlog

Figure 4 below shows all identified user stories with their prioritization. There is an identifier in priority weight from 1 to 10. A higher priority weight indicates that the corresponding user story is more important for the project's success and for meeting customer needs.

Product Backlog

ID	Actor	Task Name	Priority
1	Study Director	Create a plan of aasignments	10
2	Study Director	Manage plan of assignments	10
3	Study Director	Add new module to the plan	10
4	Study Director	Add lecturer to the module	10
5	Study Director	Publish plan for lecturers	10
6	Study Director	Officially publicize a plan	10
7	Study Director	Set lecturer's hours to for module	10
8	Study Director	See the assignment plan for a specific year	10
9	Study Director	See the requests for the modules	10
10	Study Director	See the requests for the groups	10
11	Study Director	See the working hours requests for the concrete module	10
12	Study Director	Make a groups of lecturers	10
13	Study Director	Set the group of lecturers to the module	10
14	Study Director	Make groups of modules	10
15	Study Director	Attach the lecturers to the module	10
16	Study Director	Attach the lecturers to the module group	10
17	Study Director	Attach the lecturers group to the modules group	5
18	Study Director	See last year plan	3
19	Study Director	Make a copy from last year plan	5
20	Study Director	See hours conflicts	7
21	Lecturer	See the plan of my assignments for concrete year	10
22	Lecturer	Set my hours for concrete module	10
23	Lecturer	Add new modules to my plan	10
24	Lecturer	Add new additional assignments to my plan	10
25	Lecturer	See last year plan	3
26	Lecturer	Publish my plan	10

Figure 4: Product Backlog

4 Functional Requirements Specifications

4.1 Stakeholder Summary

The stakeholders of our system are:

- **Study director**
  The first major stakeholder is a study director. The study director can be both the person who makes an assignment plan and also a lecturer. He participates in planning and is the person who takes control and main responsibility for planning.

- **Lecturer**
  The lecturer is another major stakeholder of our system. He participates in assignment planning.

4.2 Actors and Goals

The roles of people that will directly interact with the system and their goals.

- **actor** -Study director
  - Initial Actor
  - goal Uses system to make a plan of assignments for lecturers.
- **actor** -Lecturer
  - Initiating Actor
  - **goal** Uses the system to participate in the planning of his assignments .
  - **goal** Uses system to see his assignment plan.
  - **goal** Uses system to make a request for the modules he/she wants to teach.
  - **goal** Uses system to make a request for the teaching hours for a concrete module.

4.3 c. User Interface Requirements

In project2 we did the main part of the Functional Requirements. In this work, we pay special attention to the development of the user interface concepts. There is an identifier in the form of REQ-x[7] and a priority weight from 1 to 10. A higher priority weight indicates that the corresponding requirement is more important for the project's success and for meeting customer needs.

[7]REQ - is an abbreviation of Requirements

Identifier	Priority Weight	Requirements
REQ-1	1	UI must have a landing page (log in).
REQ-2	10	UI must have study director start page.
REQ-3	10	UI must have a page where the study director can make a new plan.
REQ-4	10	UI must have a page to add a module.
REQ-5	7	UI must have a page to load the existing module.
REQ-6	10	UI must have a page to create a new module .
REQ-7	10	UI must have a page to add lecturers or groups of lecturers to the module.
REQ-8	7	UI must have a page with copied assignments from last year.
REQ-10	10	UI must have a page to see the lecturer's assignments plan
REQ-11	10	UI must have a page to add a module to the lecturer's assignment plan
REQ-12	2	UI must have a lecturer's page to see the last year assignment plan
REQ-13	5	UI must have a page to see the groups of lecturers to the corresponding module/module group
REQ-14	10	UI must have a page to see and manage the plan after editing of lecturers

Table 6: User Interface Concepts

5 System Architecture and System Design

In project 2 describes the System architecture and design. In this work, this chapter is supplemented with new diagrams and descriptions.

5.1 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. [4]

This application uses a layered architecture. The pattern is used to make the development of database access faster and to make the code cleaner. And this pattern based on the domain-driven design(DDD) pattern from Eric Evans. But in this 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 the Plana application.

The guidelines that build the business logic [5]

- Low coupling between layers, high cohesion within them
- Separation of concerns
- Adaptability: be able to change
- No circular references between layers
- Lower layers should not depend on upper layers

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

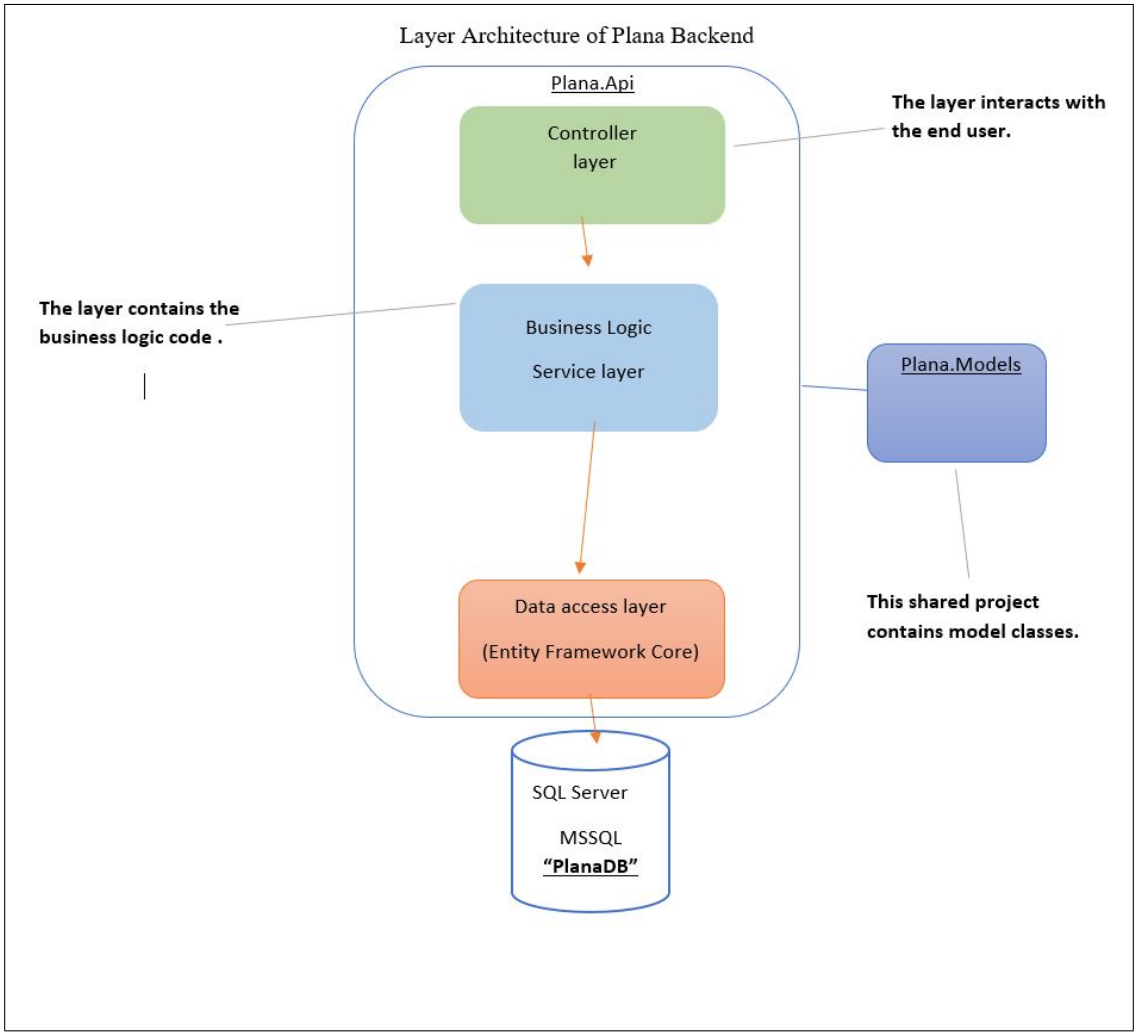


Figure 5: Layer Architecture of Plana Back-end

Layer Name	**Description**
Data access layer	The layer stores and retrieves the information from a database
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 minimal business logic.[6]

Table 7: Layers Description

Layers shall be loosely coupled and design dependency in only one direction. [7] [4]

5.2 The Plana application Architecture

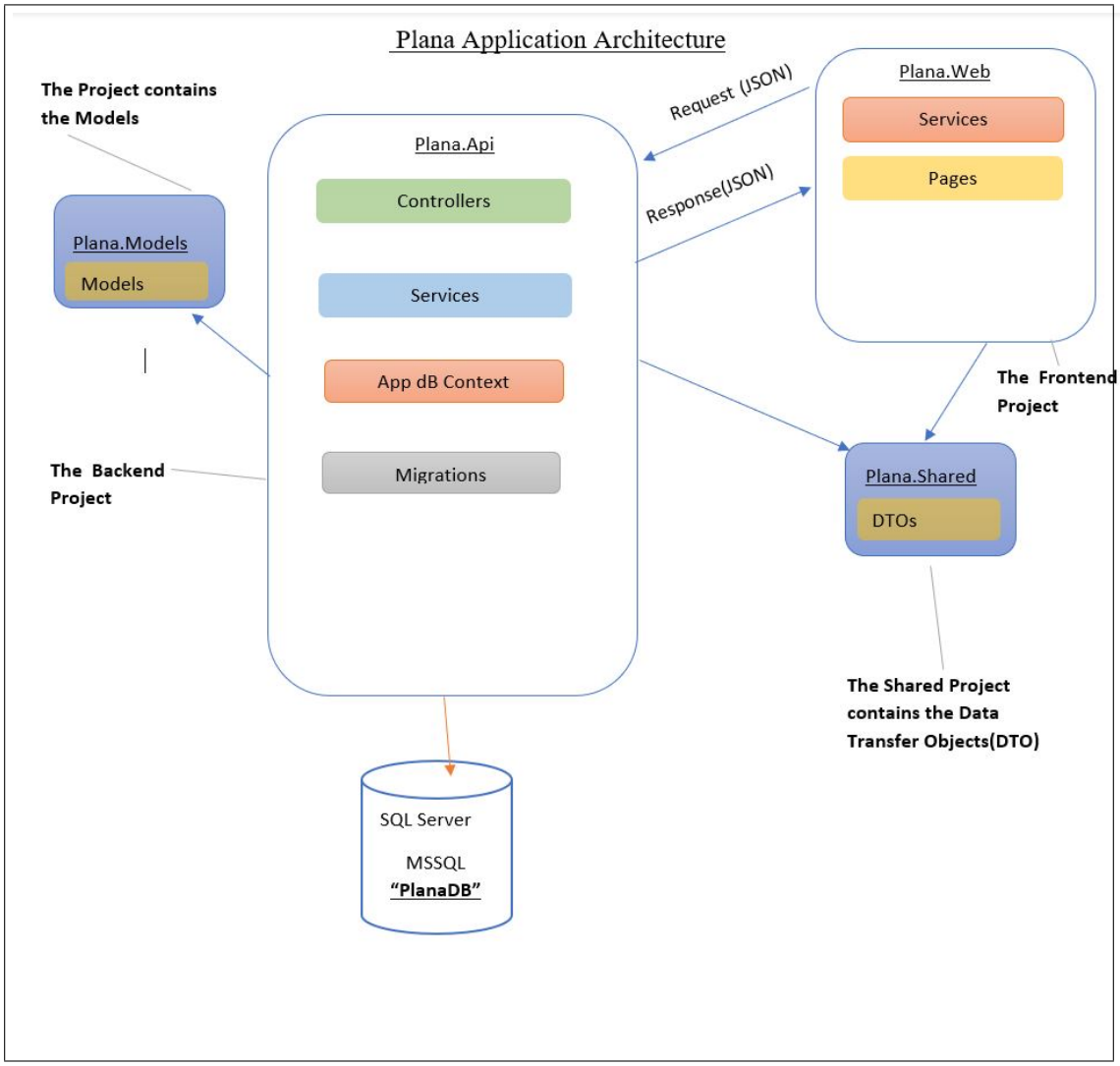


Figure 6: Layer Architecture of Plana Application

Components	Description
Plana.Api Project	
Controllers	Controllers - define the end point into the web api from client application via http requests. [8]
Services	This folder contains classes that are communicate with the data layer and controllers. These classes contain the main business logic
App dB Context	An instance of DbContext represents a session with the database which can be used to query and save instances of your entities to a database. DbContext is a combination of the Unit Of Work and Repository patterns.[9]
Migrations	This folder includes Entity Framework Core generated migration classes
Plan.Web Project	
Pages and Shared	The Pages and the Shared folders contain .razor files, which are the Blazor components [10]
Services	Service class uses HttpClient[8] to call the REST API service[11]
Plana.Models Project	
Models	This folder holds the application domain objects, the Code First classes. [12]
Plana.Shared Project	
DTO models	In this application DTO has a role of the view model, which is used to transfer data between the domain model and the view, it contains User Interface logic

Table 8: Plana Projects Components

6 User Interface Specification

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

6.1 a. Preliminary Design

We made a user interface prototype using a **Axure RP** tool. This prototype can be found under the following link `https://5ztn9l.axshare.com`, the password to get access to the data is **oblako**. On the login page of the prototype, the user name and

password are not required, it is necessary to click the button login.
The following are print screens of the prototype we have created.
i. Login
ii.Study director's dashboard
Once the study director successfully logged in, the personal dashboard page should be shown directly

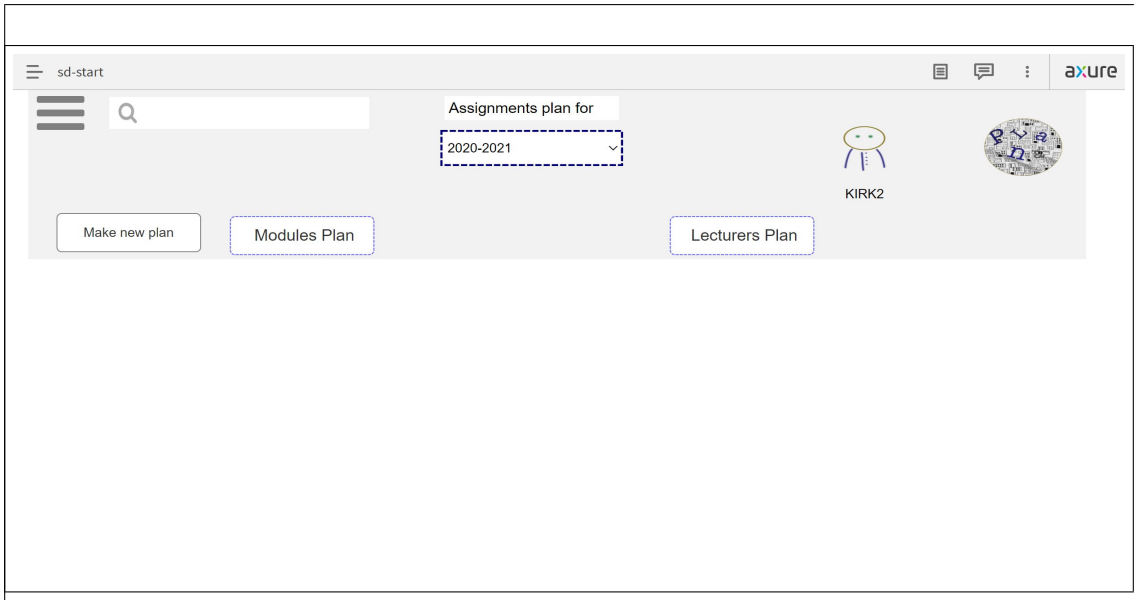


Figure 7: Study director's dashboard view

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.

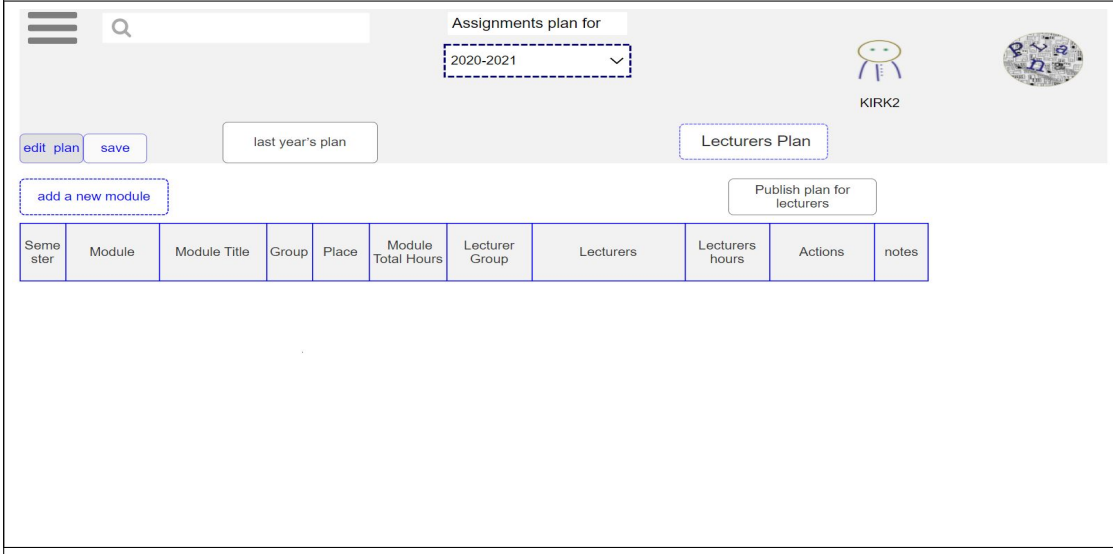


Figure 8: Create plan view

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.

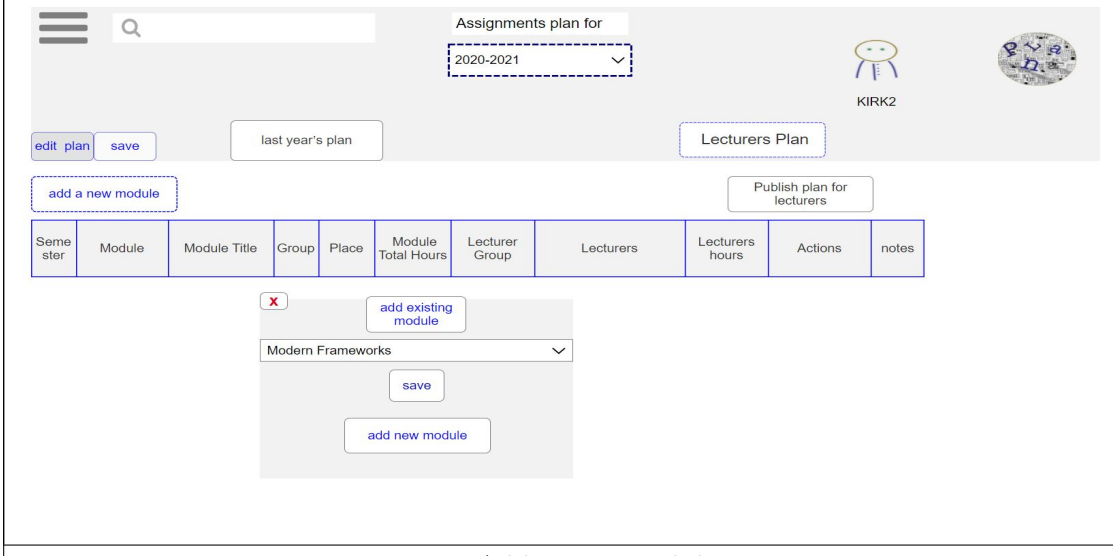


Figure 9: Add a new module view

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 the 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 11: Add module/ modules group view

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.

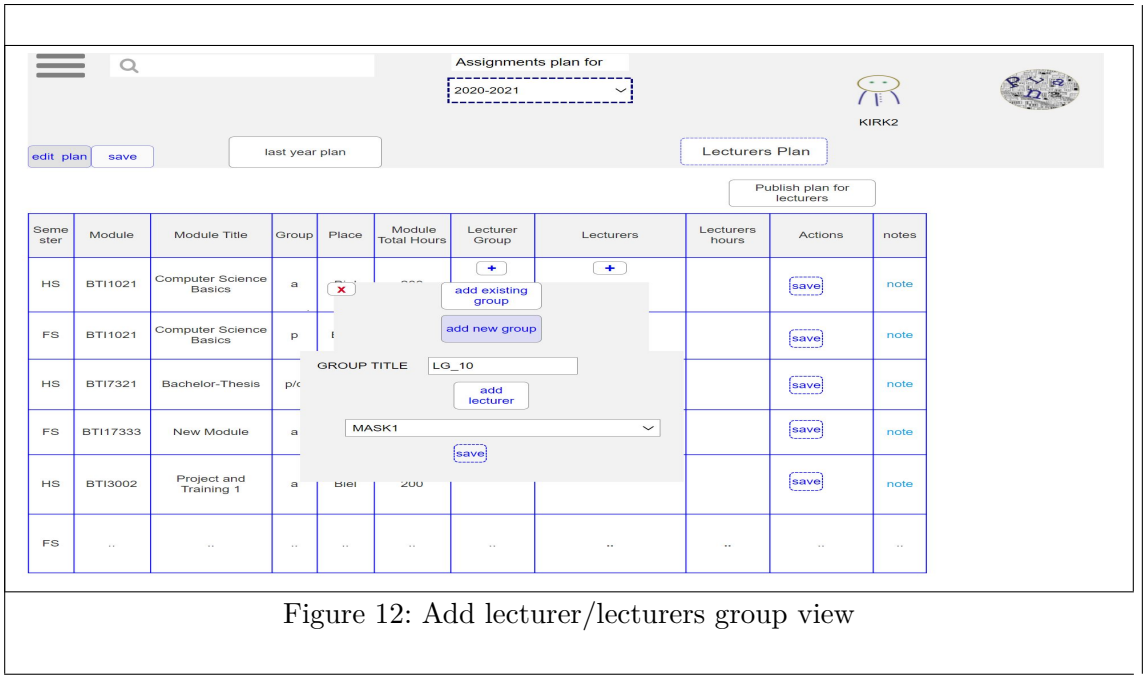


Figure 12: Add lecturer/lecturers group view

vii. See and copy last year's 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 by 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 by clicking the button "copy to the actual plan".

Figure 14: Copy last year plan.

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

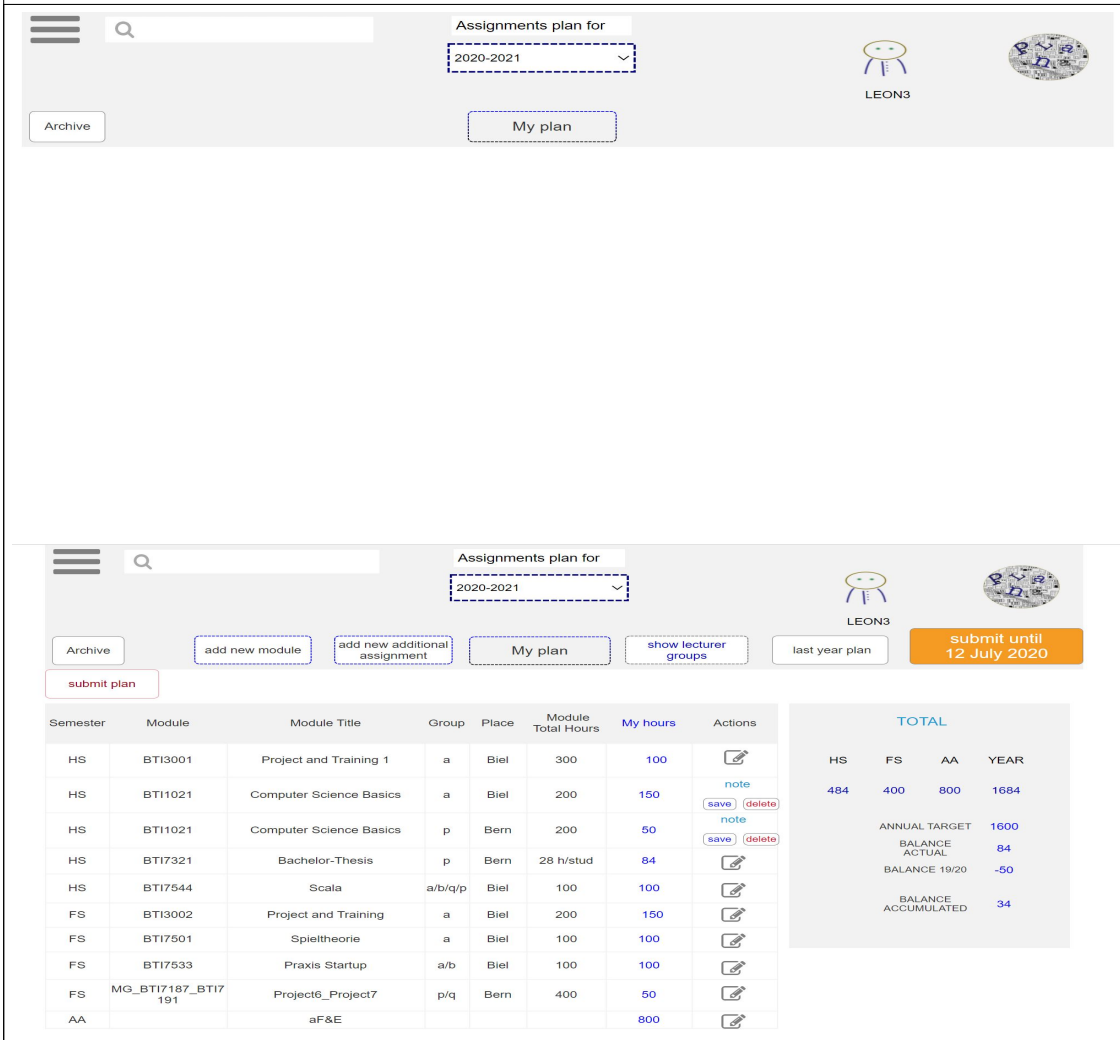


Figure 16: Lecturer's dash board view with opened plan

ix. Add new module

The 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 the desired assignment to his plan and set the hours.

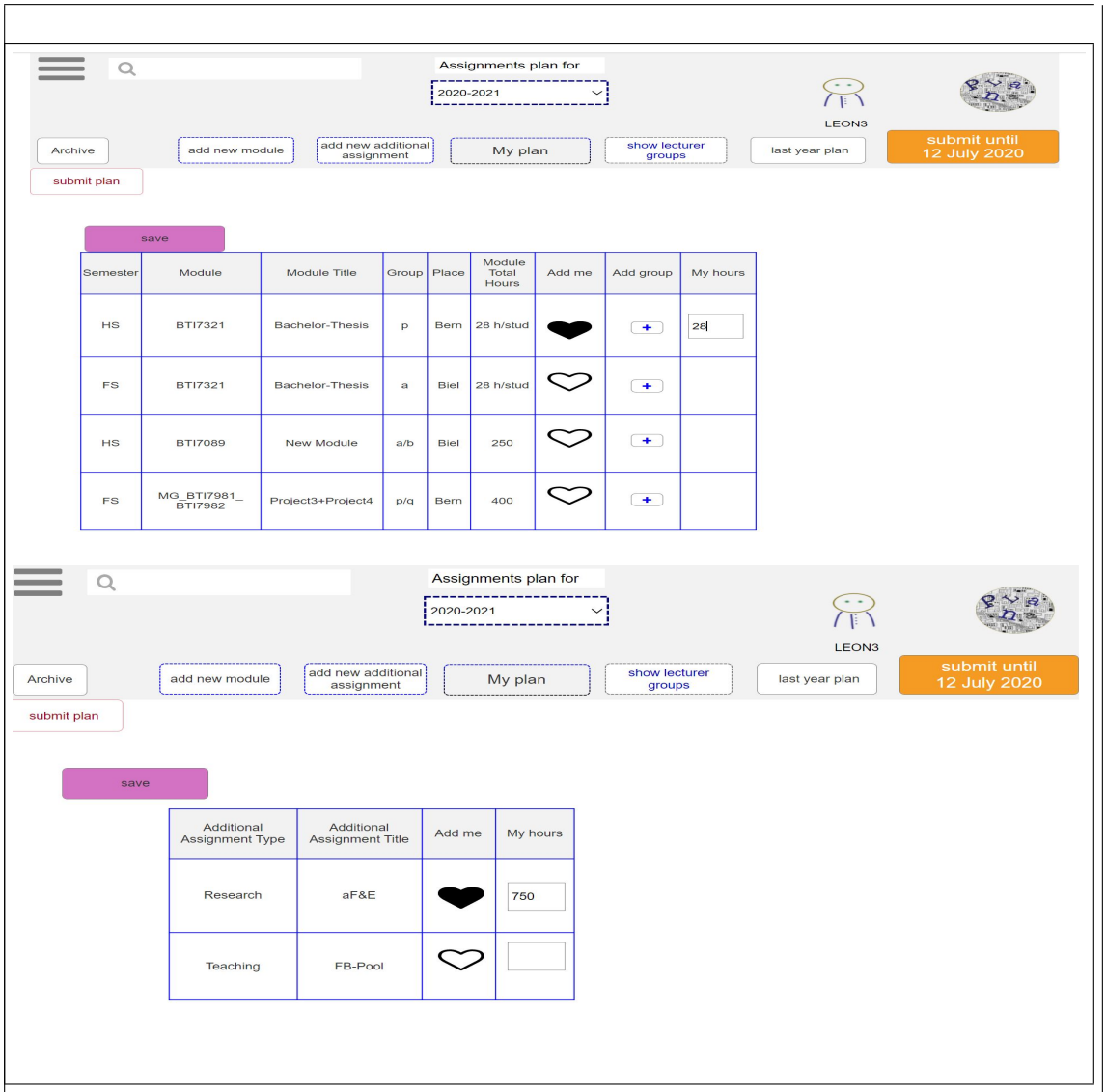
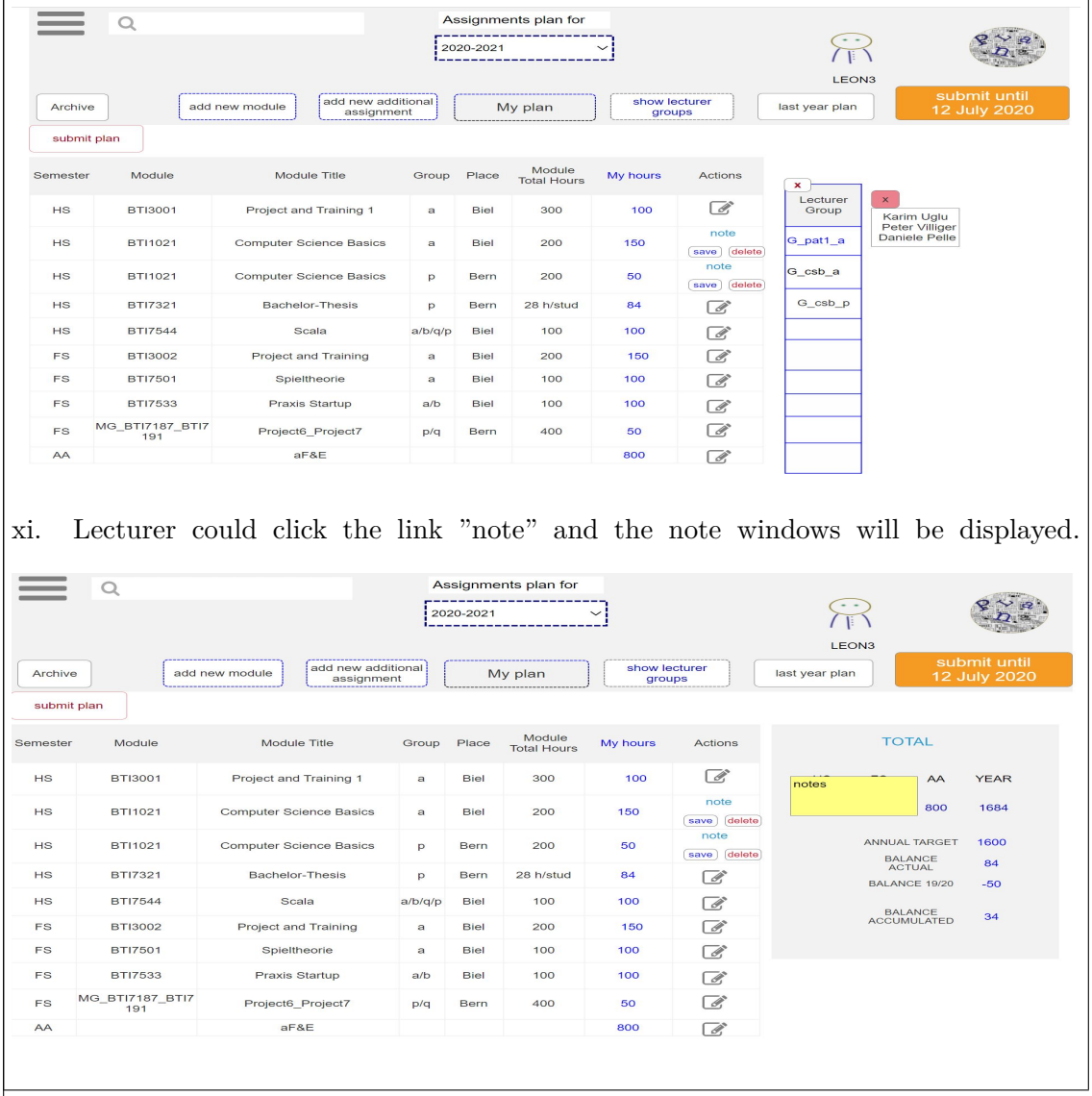


Figure 18: Add a new additional assignment.

x. Lecturers groups

The 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 20: See the notes view.

xii. Lecturer's last year plan

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

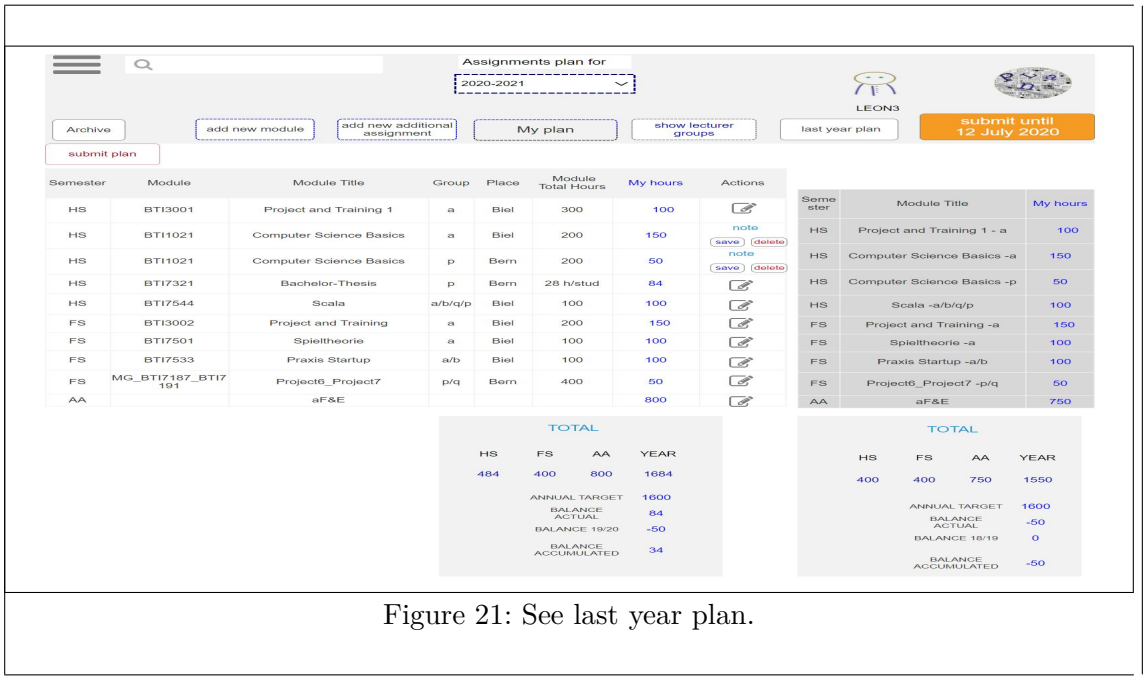


Figure 21: See last year plan.

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.

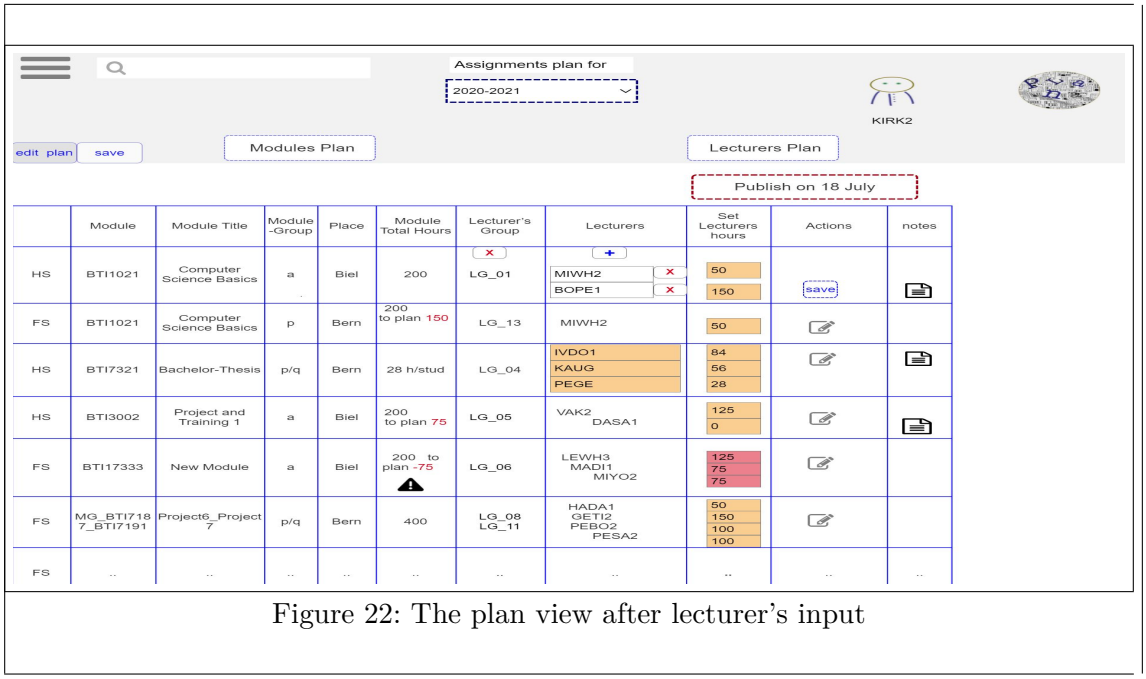


Figure 22: The plan view after lecturer’s input

xiv. Make a note

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

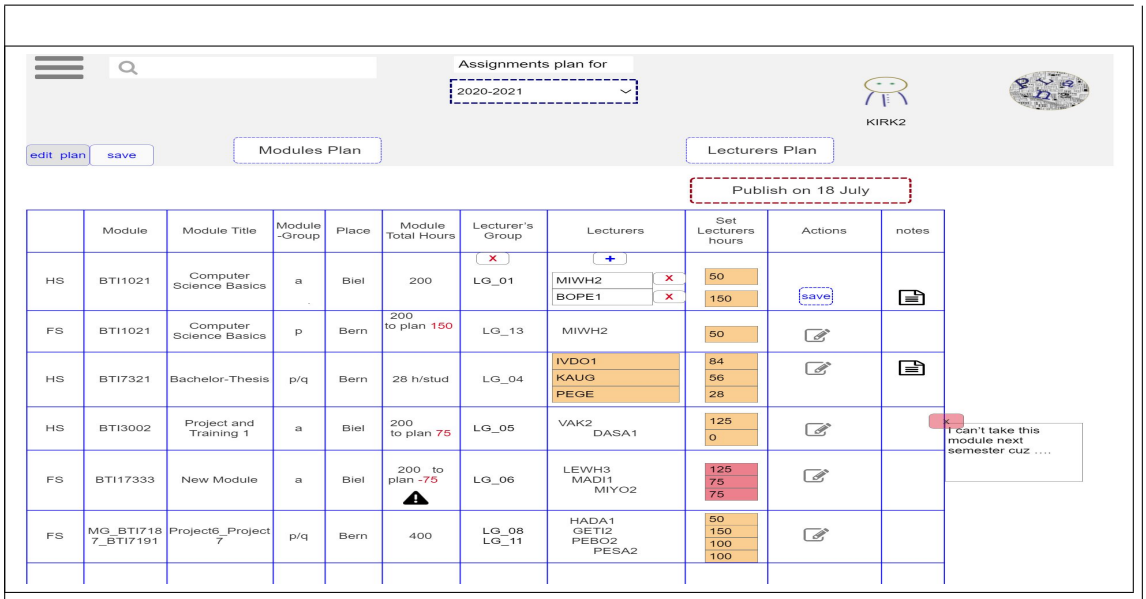


Figure 23: Make a note view.

7 Domain Analysis

7.1 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.

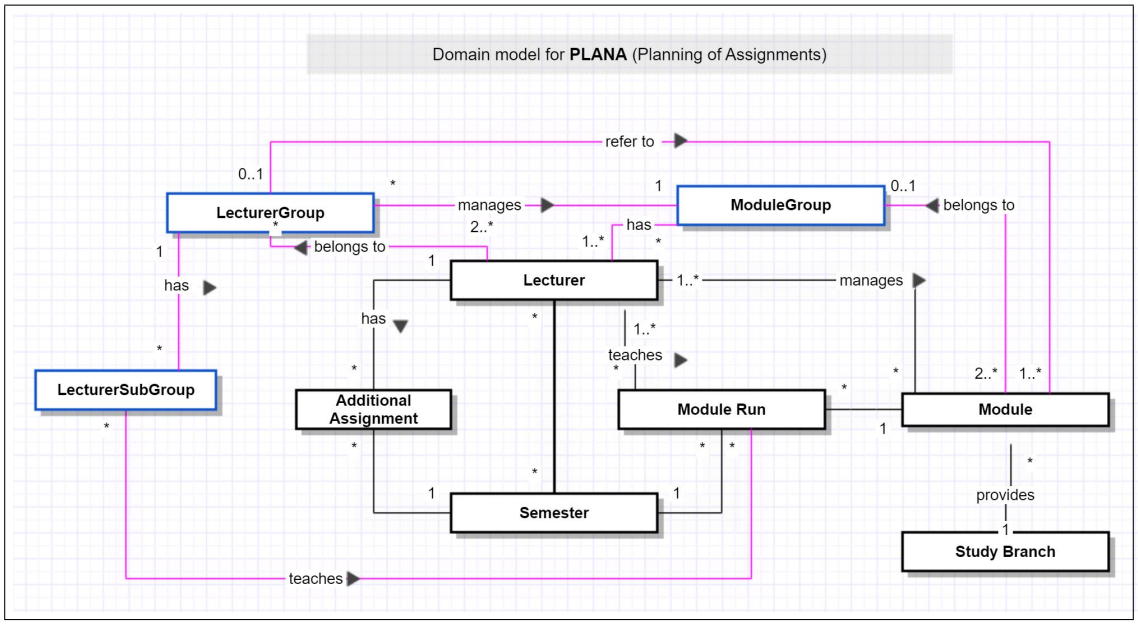


Figure 24: Domain Model for PLANA

In the domain model (Figure 24) 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.

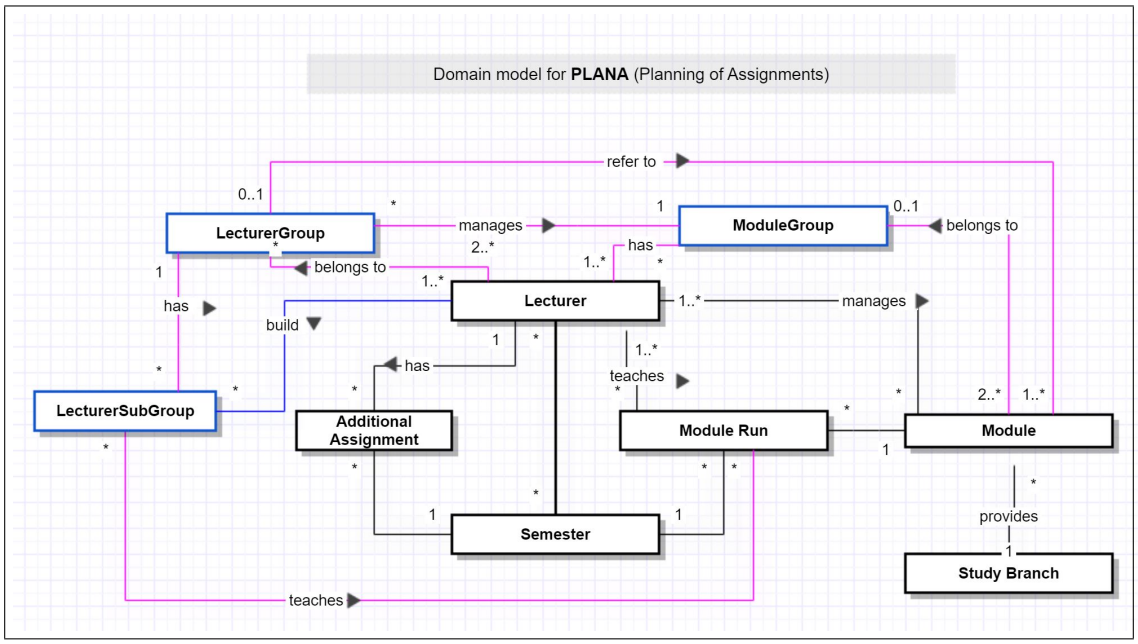


Figure 25: Domain Model for PLANA

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 groups of the Module Runs.

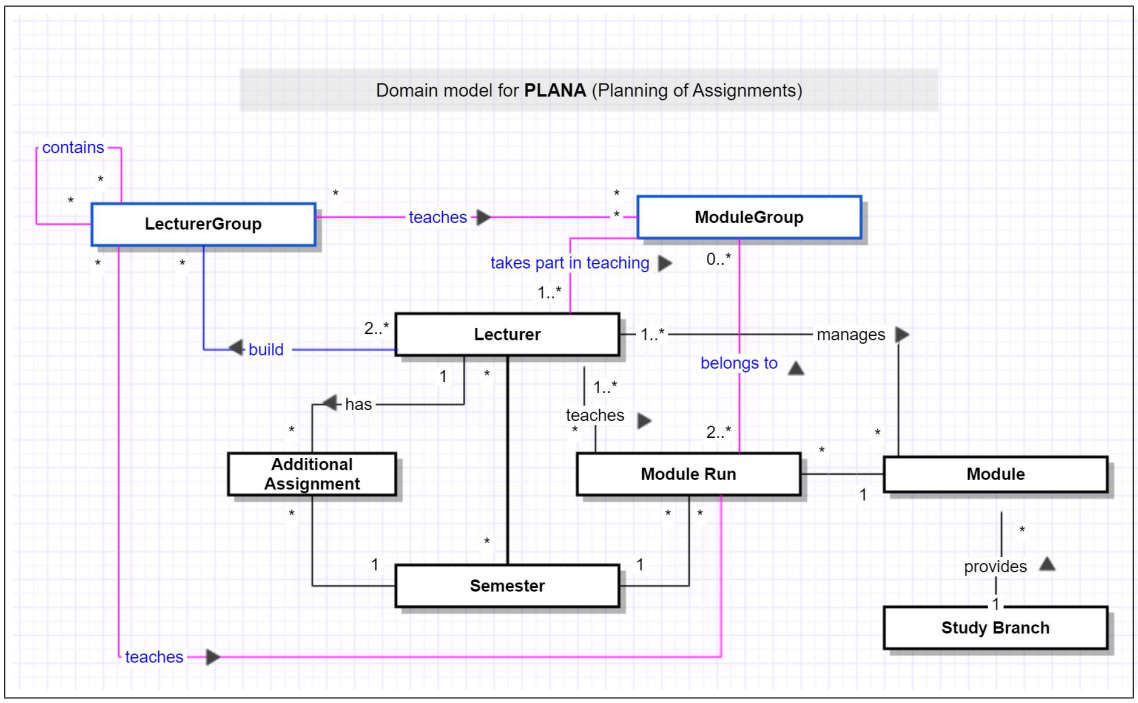


Figure 26: Domain Model for PLANA

The Figure 27 show the last variant of the Domain Model, where we added the concept class Planänd association between Semester and Lecturer.This gives us possibility to make the code easier.

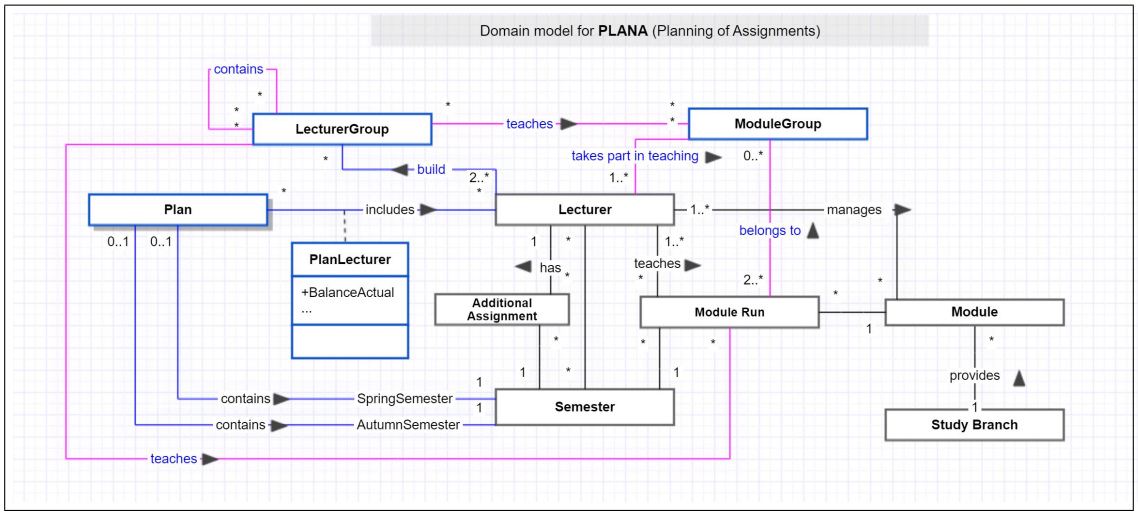


Figure 27: Domain Model for PLANA

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 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.
- Concept class **Plan** models the Plan for concrete year.

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	Each Semester has many teachers, and these teachers are teaching in more than one Semester	includes
Lecturer -> LecturerGroup	Each Lecturer builds 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

Table 9: Association Definition

Concept Pair	Association Definition	Association Name
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 contain a zero or multiple LecturerGroup and each LecturerGroup can be included in zero or multiple LecturerGroup .	contains
Plan-> Semester	Each Plan contains exactly one Autumn Semester and exactly one Spring Semester and each Semester can be included in zero or one Plan .	contains
Plan-> Lecturer	Each Plan includes from zero to many Lecturers and each Lecturer can be included in zero or many Plans .	includes

Table 10: Association Definition

8 Technologies

This chapter is described in project 2 (report-roject2.pdf in Chapter 7), which can be found at the following link `https://gitlab.ti.bfh.ch/shirk1/planning-of-the-assignments-for-le -/tree/master/help%20documents`

9 Creating the Projects

In project 2 we started the implementation of the Plana application. We created the initial structure of the project, added domain classes, and implemented the crud(Create, Update, Delete) operation for the teacher.

9.1 Structure of Projects and Folders

The structure of Projects as well as the Data Models Project, Plana.API project, Plana.Web project, as well as Plana Database, were described in Project 2 in the document Visual Review In Initial Phase of Project.pdf at the following link `https://gitlab.ti.bfh.ch/shirk1/project2/-/tree/master/report-docs` In this work, we added one more sub-Project Plana.Shared, which contains the Data transfer object(DTO), the objects that are used to encapsulate data, and send it from one subsystem of an application to another.[13]

9.2 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 [14] [15]

9.3 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 an 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

9.4 Modeling Types of Database Relationships

Many-to-many Relationship

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

In EF Core database doesn't directly implement this kind of relationship. First, we have to create a 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 composite key.[14]

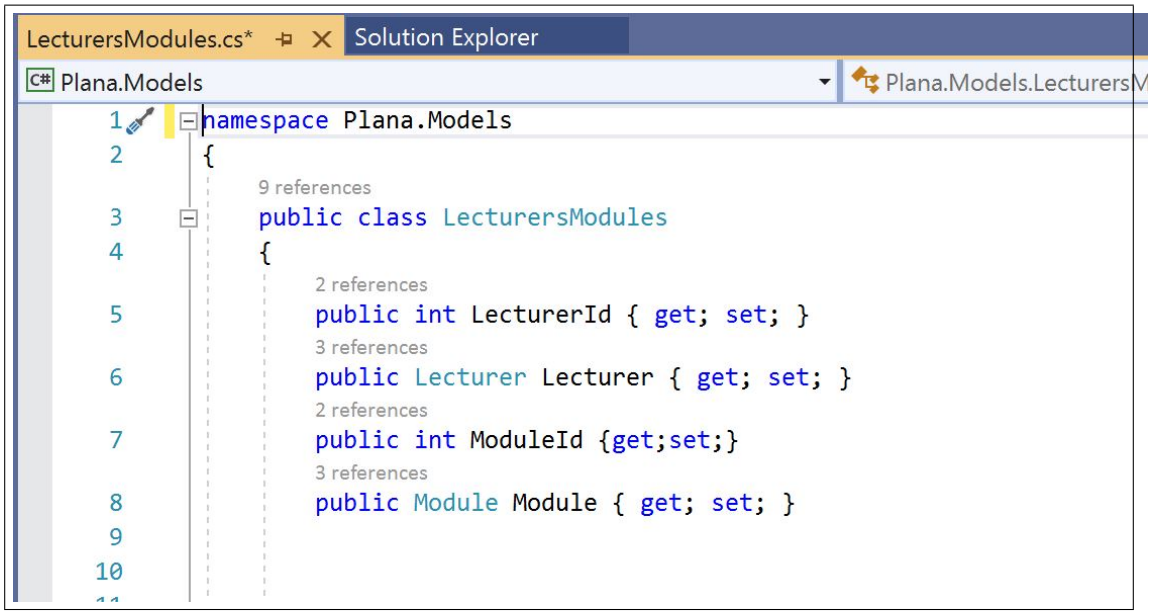


Figure 28: The LecturersModules entity class

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**;

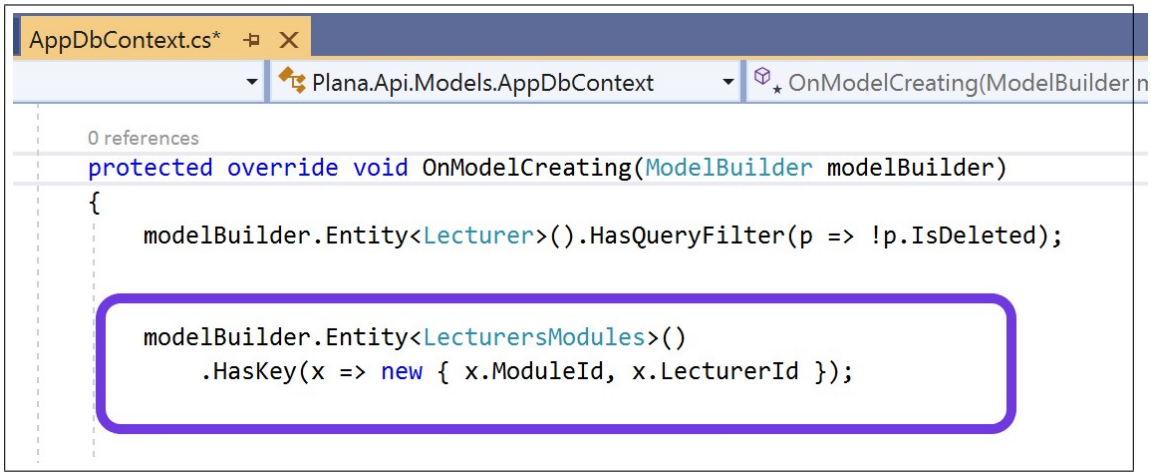


Figure 29: Adding Entity LecturersModules to the AppDbContext class

We need write just this and Entity Framework Core will do the correct implementation that we can see then in the migration files. [16]

```
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 30: Initial Migration File

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

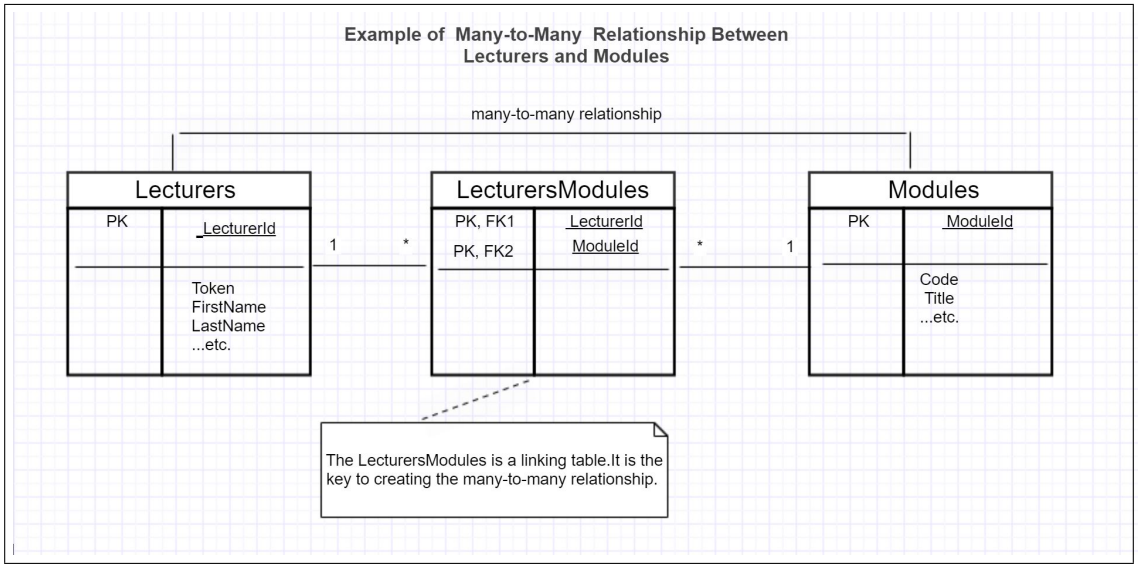


Figure 31: Creating Many-To-Many Relationship

9.5 Creating a Repository

todo: photo of the 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 32: 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 33: Creating Services in Startup.cs File

[17]

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** [17]

9.6 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 [17]. By default, Entity Framework Core uses cascade deletes depending on relationships with non-nullable foreign keys. [14]
..add a 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 [14]. 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 34: Startup.cs in the Plana.Api project.Preparing Services and Middleware

9.7 Create a Controller

Complex Data Model

In this section, we would like to highlight more complex features of coding and data model structure in ASP.NET Core.

To see a data from related entities it is necessary use methods **Include**, **ThenInclude**.

Without the data in related entities shows the null values.

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

        }
```

Figure 35: 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 36: JSON Response of GetLecturers method

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

Figure 37: GetLecturersModules method with added methods to get related data.

After we have changed the structure of the method we have got the exception. We solve this problem by installing the package (Microsoft.AspNetCore.Mvc.NewtonsoftJson) and by changing the configuration in the Startup class.

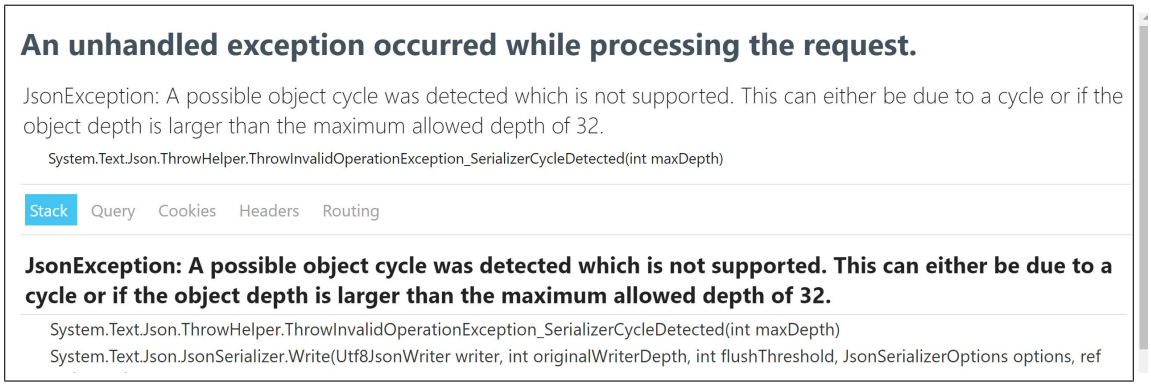


Figure 38: Excetption message

```
Package Manager Console

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

Figure 39: The Microsoft.AspNetCore.Mvc.NewtonsoftJson package

```
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 40: The Startup class

Figure 41 shows a result that now contains the data we wanted.


```
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 41: Response of getLecturersModules method

Blazor Server

Configuring ASP.NET Core for Blazor Server

Call the API from Asp.net Core Blazor

.. add a 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 42: 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 43: 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.

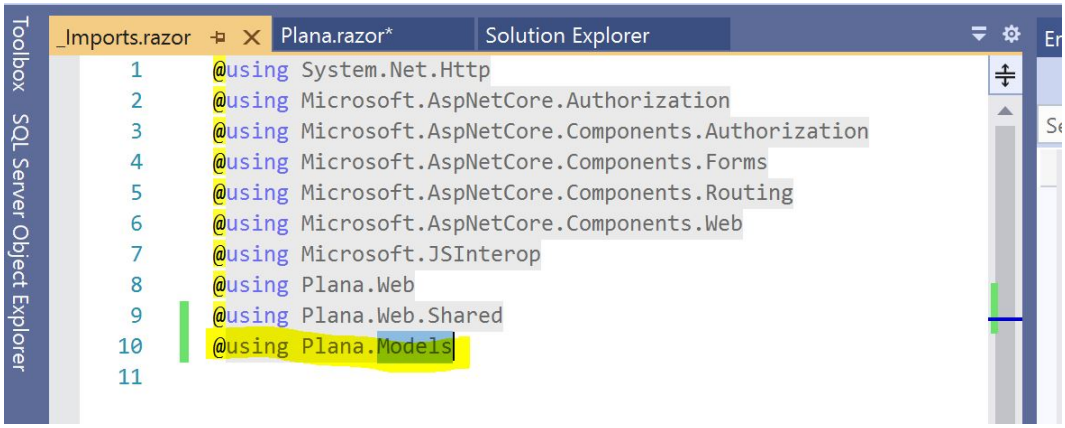


Figure 44: Adding required namespaces to the _Imports.razor file in Plana.Web project folder

9.8 Changing Architecture

After studying the literature about the Framework entity, we concluded that the Layer repositories do not really matter in the application, since Repository just duplicates what the DbContext class of Entity Framework already gives us. We deleted the repository layer, so the service class directly contacts with the data access layer.

9.9 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/`

10 Testing

10.1 Testing API

The API layer of our application is the most important part. It is the channel that connects the client to the server, contains business logic, in our case in the service classes, and provides the controllers which give value to users.[18] The API that is provided to the client, in case it breaks, will put at risk not only one application but the whole chain of business processes built around it. That is why it is necessary to test this part of the program.

API Test Actions

Each test consists of test actions. For each API request, the following steps are taken.

- Verify correct HTTP status code
  To Test the API the Postman application was chosen. There are several HTTP status codes. The status code it is when the server sends back an HTTP code to signal the status of the request.[19]
  Common HTTP Status Codes

Status Code	**Description**
200	OK Successful responses
201	CREATED responses
400	Bad_REQUEST
403	FORBIDDEN responses
404	NOT_FOND
500	SERVER_ERROR

Table 11: HTTP status code

- **Verify response payload.** Check valid JSON body and correct field names, types, and values
- **Verify response headers**
- **Verify correct application state.** Applicable if the check of the user interface is available [18]

Test Cases

We have tested all methods in Controller classes. Below are examples of testing certain methods and Response Statuses.

Testing the ModuleRun Controller

Update ModuleRun

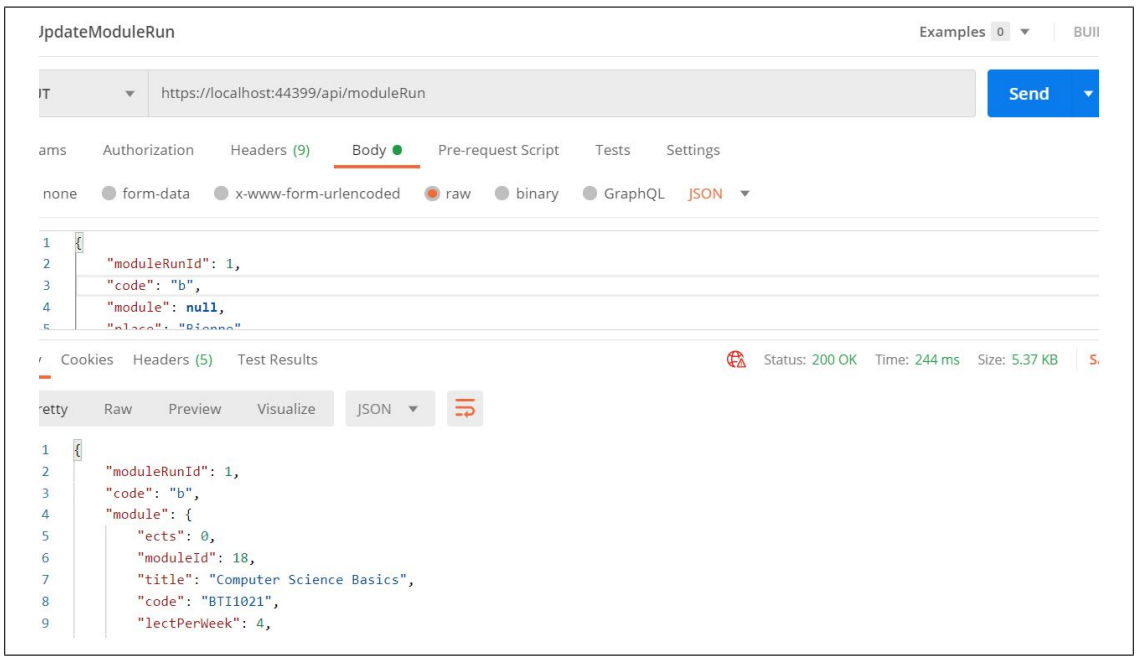


Figure 45: Test for update method of ModuleRun

Get ModuleRun

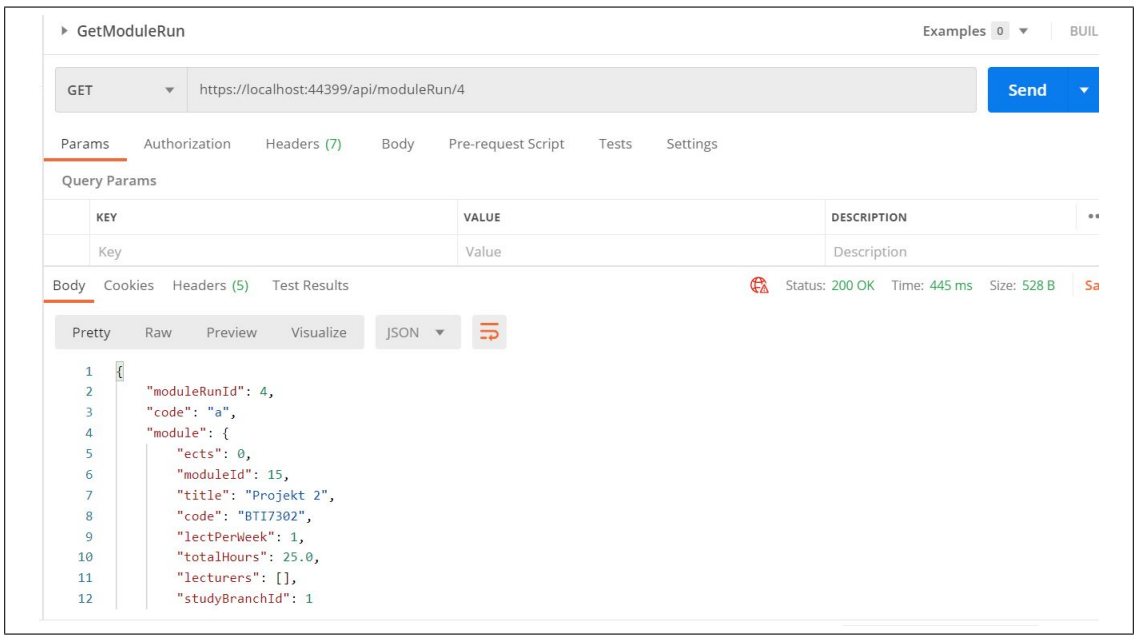


Figure 46: Test for get ModuleRun method

Testing the LecturerModuleRun Controller

Get LecturerModuleRun

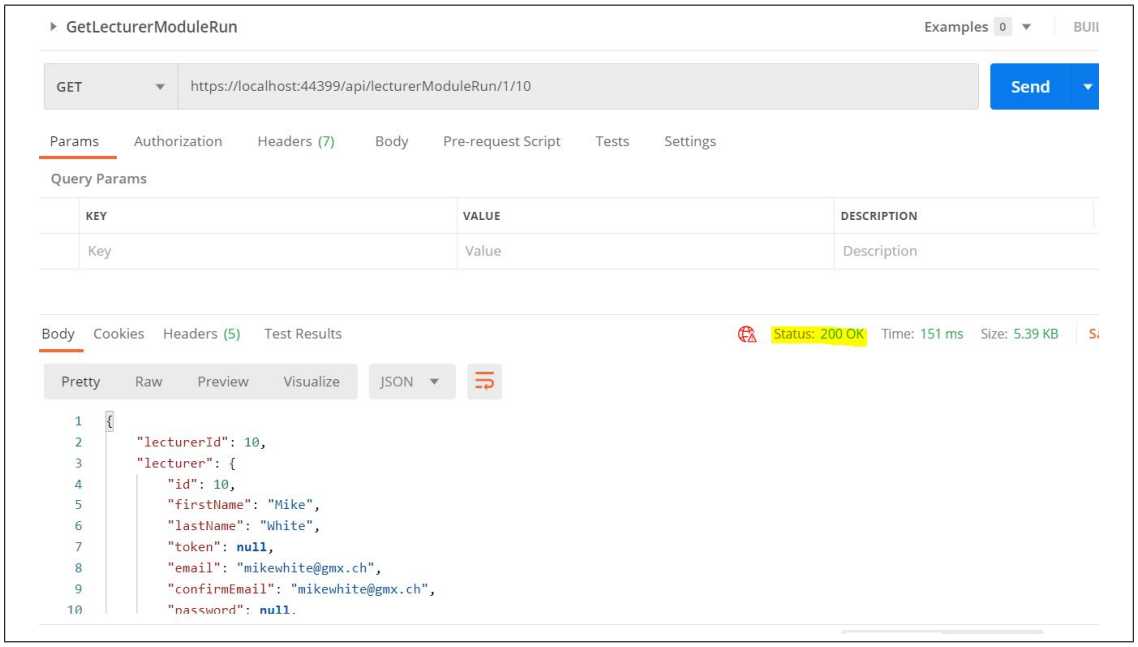


Figure 47: Test for get LecturerModuleRun method

Testing the Modules Controller

Get All Modules

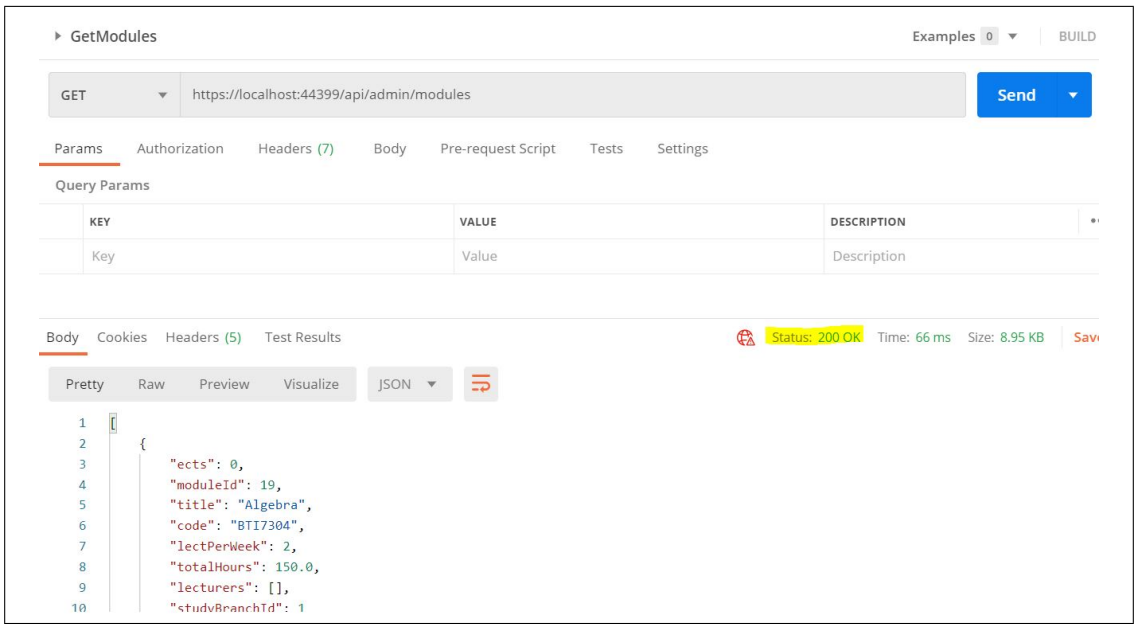


Figure 48: Test for get all modules

Get module

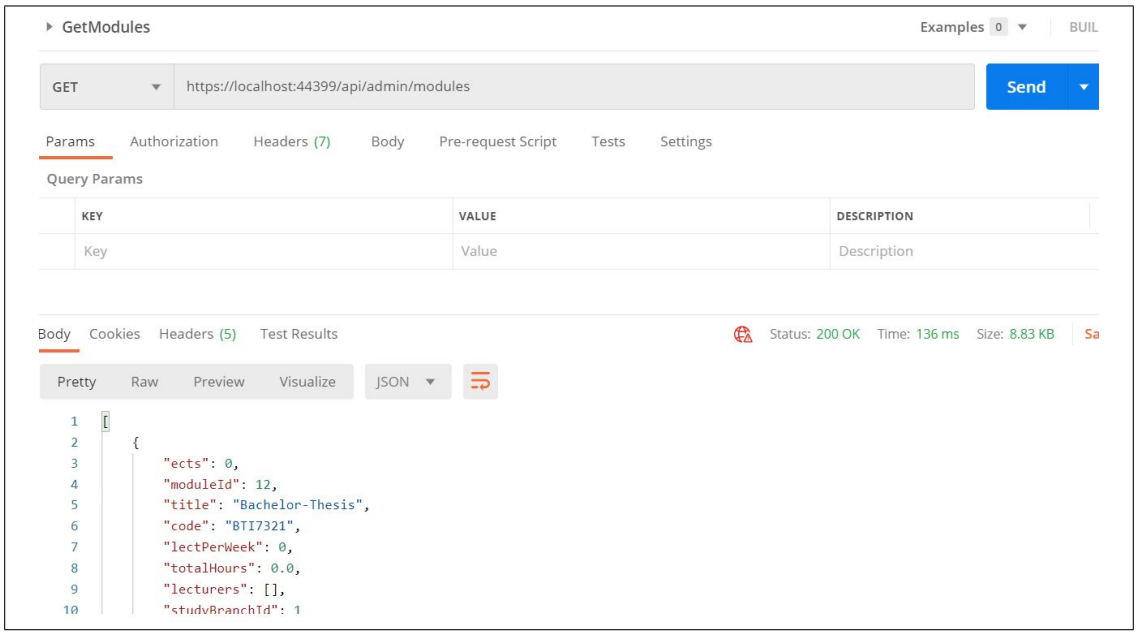


Figure 49: Test for get a module method

Create module

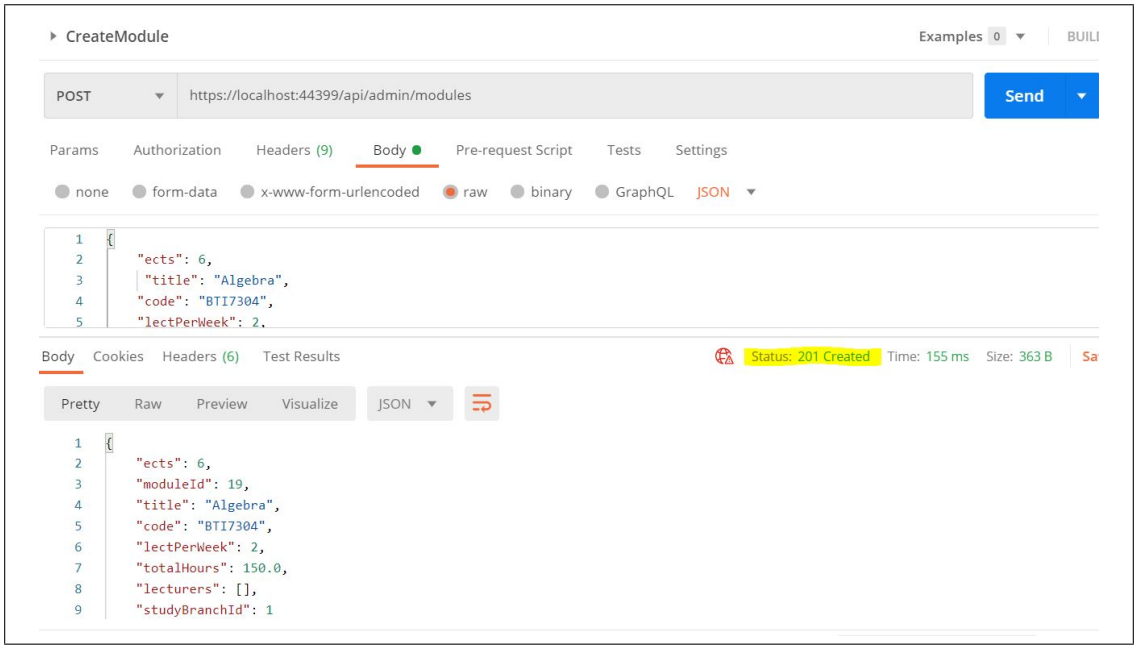


Figure 50: Test for create a new module method

Testing the Plan Controller

Create a plan

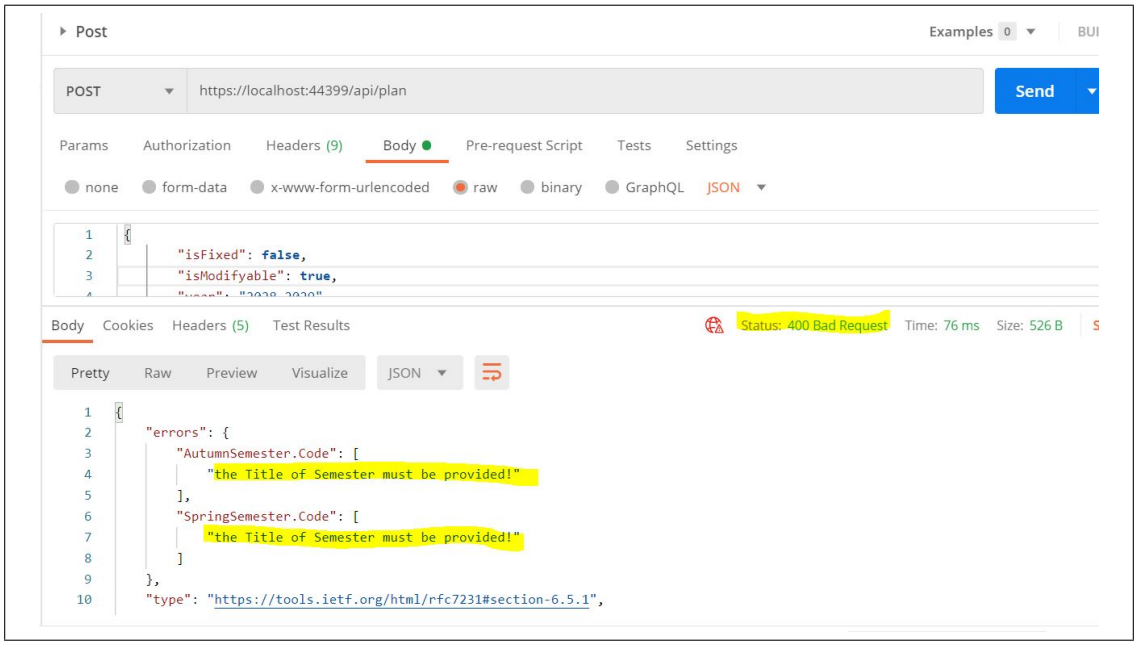


Figure 51: Test for create a new plan method

Testing the Plan Lecturer Controller

Get module runs of certain planLecturer

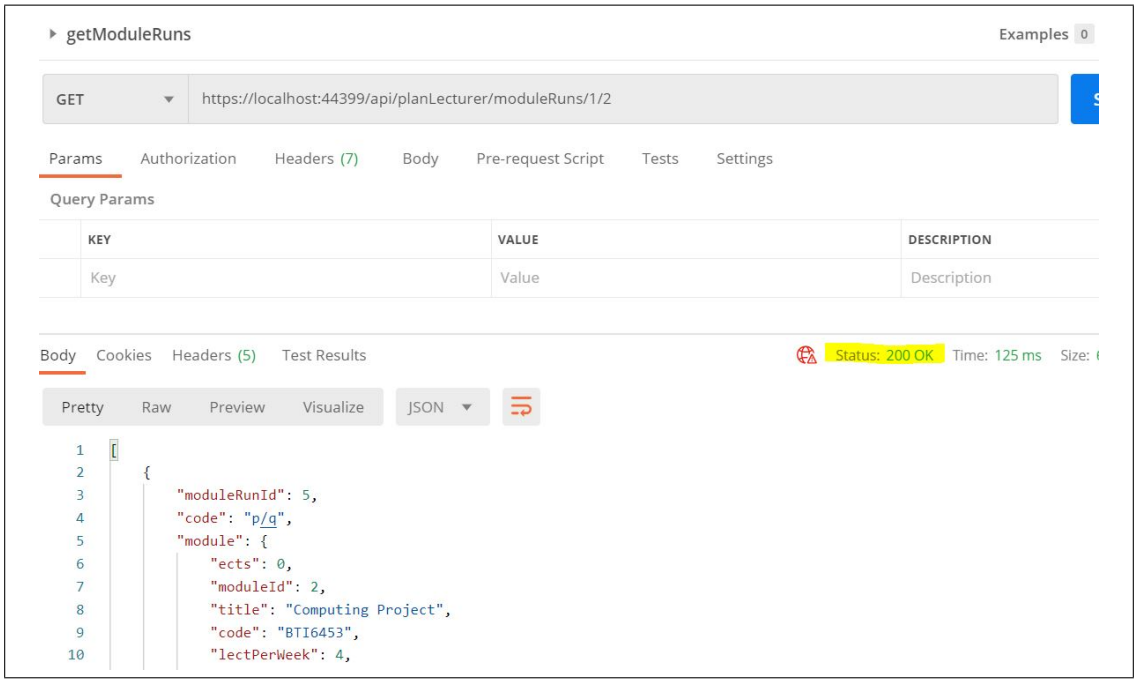


Figure 52: Test for get all module runs for the concrete Lecturer in concrete Plan method

Testing the Semester Controller

Delete Semester

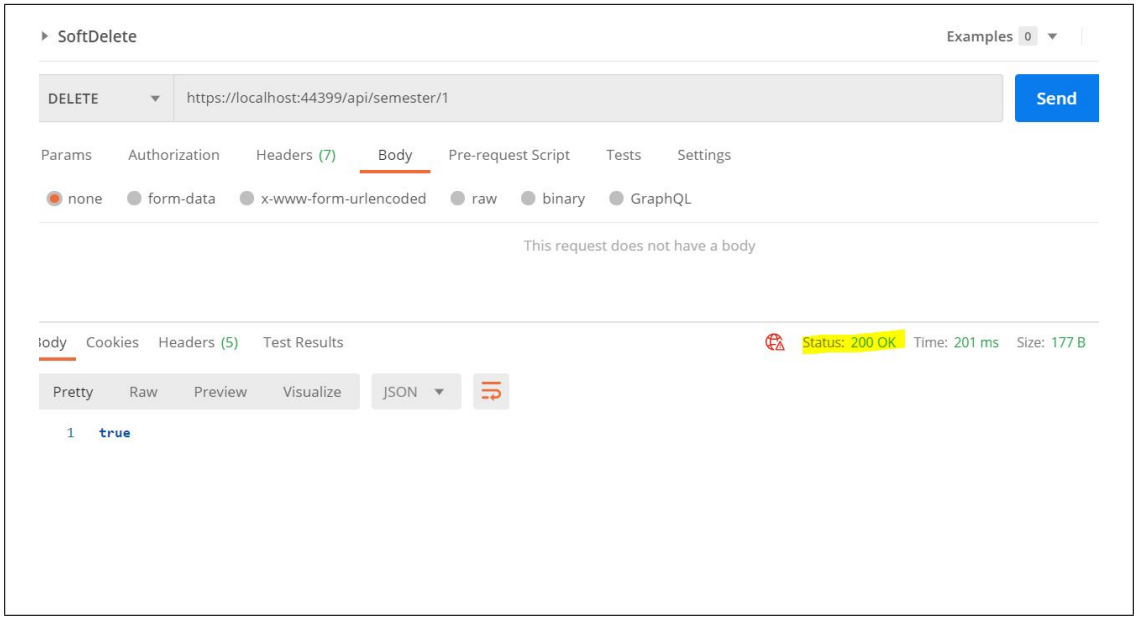


Figure 53: Test for delete a semester method

10.2 Testing Frontend

The frontend is the visual client-side of a program. To test a Presentation layer of our system we have asked two people to help with it, so they were our testers.

- **Usability Test** The first test task was to test the system without any knowledge about the system requirements. We tested the application for its intuitiveness, ease of use, and process clarity. Other front end tests need knowledge about requirements. The next tests were
- **Error-free test** Check each element for errors
- **Acceptance test** Evaluation the compliance of a system with business requirements[20]

Before starting with testing We have made a testing plan. The necessary questions to the testers were recorded. A checklist with the functions of the system was made.

Summing up this testing, I can say that it took a lot of time to test the API, since it contains the business logic of the system, respectively, the main operations. But such testing is very effective, since it allows to find errors at the development stage of the back end and, accordingly, eliminate them in time, which prevents further difficulties with the implementation of the code. Speaking of testing the Front End, bugs were revealed that were not obvious before.

11 Conclusion

First, we will discuss the efficacy of Blazor. Blazor has become a really good tool for writing web applications. We were not constrained in any way by using Blazor, mainly due to being able to use Blazor with Java Script, which offers a wide range of possibilities. Speaking specifically about the main contribution of this work, the creation of a new product, we summarize the goals that have been achieved.

- New use cases of the system(PLANA) have been determined with respect to task specification and the specification of all the necessary conditions that the system should satisfy
- Adjustments have been made to the domain model, related to the increasing of assignments and the improvement of domain logic
- The architecture of the system has been improved by adding a project with a DTO (Data Transfer Object)
- Graphic concepts have been developed
- The product has been implemented and tested

The figure below shows the Product Backlog. It contains all the identified user stories for this application. Completed tasks are marked with the word /“done/”, tasks that are partially completed with the word /“doing/”, and tasks that can be completed in the future with the word /“todo/”.

A possible continuation of this work may be the development of a system for scheduling of teachers’ employment.

Product Backlog

ID	Actor	Task Name	Priority	Status
1	Study Director	Create a plan of aasignments	10	done
2	Study Director	Manage plan of assignments	10	done
3	Study Director	Add new module to the plan	10	done
4	Study Director	Add lecturer to the module	10	done
5	Study Director	Publish plan for lecturers	10	doing
6	Study Director	Officially publicize a plan	10	doing
7	Study Director	Set lecturer's hours to for module	10	done
8	Study Director	See the assignment plan for a specific year	10	done
9	Study Director	See the requests for the modules	10	done
10	Study Director	See the requests for the groups	10	doing
11	Study Director	See the working hours requests for the concrete module	10	done
12	Study Director	Make a groups of lecturers	10	doing
13	Study Director	Set the group of lecturers to the module	10	doing
14	Study Director	Make groups of modules	10	doing
15	Study Director	Attach the lecturers to the module	10	doing
16	Study Director	Attach the lecturers to the module group	10	doing
17	Study Director	Attach the lecturers group to the modules group	5	doing
18	Study Director	See last year plan	3	todo
19	Study Director	Make a copy from last year plan	5	todo
20	Study Director	See hours conflicts	7	done
21	Lecturer	See the plan of my assignments for concrete year	10	done
22	Lecturer	Set my hours for concrete module	10	done
23	Lecturer	Add new modules to my plan	10	done
24	Lecturer	Add new additional assignments to my plan	10	done
25	Lecturer	See last year plan	3	todo
26	Lecturer	Publish my plan	10	doing

Figure 54: Product Backlog

List of Figures

12 Appendix

Gitlab-Repositories (Bachelor-Thesis):
`https://gitlab.ti.bfh.ch/shirk1/planning-of-the-assignments-for-lecturers-plana`
Gitlab-Repositories (Project2):
`https://gitlab.ti.bfh.ch/shirk1/project2`

Other sources used:
Blazor Tutorials: `https://www.pragimtech.com/courses/blazor-tutorial-for-beginners/`
Guide to a Writing a Thesis: `https://www.ece.rutgers.edu/~marsic/thesis-guide.html`

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:
Solothurn, 21.01.2021

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] Wikipedia, *Representational state transfer.* [Online]. Available: https://en.wikipedia.org/wiki/Representational_state_transfer (visited on 2021).

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

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

[6] 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).

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

[8] J. Watmore's, *Asp.net core 2.2 - role based authorization.* [Online]. Available: https://jasonwatmore.com/post/2019/01/08/aspnet-core-22-role-based-authorization-tutorial-with-example-api/ (visited on 01/08/2010).

[9] E. F. Tutorial, *Entity framework core: Dbcontext.* [Online]. Available: https://www.entityframeworktutorial.net/efcore/entity-framework-core-dbcontext.aspx (visited on 2020).

[10] M. Pfahrer, *E-business and web – web technologies, blazor, 2019/20.*

[11] P. technologies, *What are restful apis.* [Online]. Available: https://www.pragimtech.com/blog/blazor/what-are-restful-apis/ (visited on 2020).

[12] S. Singh, *Onion architecture in asp.net core mvc.* [Online]. Available: https://www.c-sharpcorner.com/article/onion-architecture-in-asp-net-core-mvc/ (visited on 06/06/2020).

[13] stackoverflow, *What is the data transfer object.* [Online]. Available: https://stackoverflow.com/questions/1051182/what-is-data-transfer-object/ (visited on 2020).

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

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

[16] 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).

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

[18] Sisense, *Rest api testing*. [Online]. Available: https://www.sisense.com/blog/rest-api-testing-strategy-what-exactly-should-you-test/ (visited on 09/23/2019).

[19] postman, *Learn http status codes with http cats*. [Online]. Available: https://blog.postman.com/http-cats-learn-http-status-codes/ (visited on 08/10/2020).

[20] Testim.io, *Front end testing*. [Online]. Available: https://www.testim.io/blog/front-end-testing-complete-overview/ (visited on 2021).

14 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:

- Project management document
- Add new items to Product Backlog

Retrospective
discussion about Project 2 Next Meeting: 08.09.20
***
Report from 08.10.20
Plan
Future goals are:

- Add an Introduction
- Start with a new views

Retrospective
Discussion about project management 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 fewer elements.
- Put UI concepts to the report.

Retrospective

- In graphical concept we have a choice between semester view and year view, but for planning is more important year view.
- Actual state of the planning view for the study director looks a little bit heavy because of many elements in it, would be better to minimize some staff. In the implementation, we can manipulate views 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, the Lecturer, Study Director, and Institute Manager can active collaborate with each other. That is the idea of making groups is also innovative. In general, we make a specific solution to the 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 a number of hours and lecturers that can teach specific modules have a place in the 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 we use the article from the internet.

Next Meeting: 02.12.20)

***

Report from 02.12.20

Plan
Future goals are:

- Check functionality of most important use cases

Retrospective

- Discussion about the Views

Next Meeting: 20.01.21)