Lab Management System

Tomas Svejnoha
987451

Submitted to Swansea University in fulfilment
of the requirements for the Degree of Bachelor of Science

Department of Computer Science
Swansea University

April 26, 2023

Declaration

This work has not been previously accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

Signed	Tomas Svejnoha	(candidate)
Date	26/04/2023	

Statement 1

This thesis is the result of my own investigations, except where otherwise stated. Other sources are acknowledged by footnotes giving explicit references. A bibliography is appended.

Signed	Tomas Svejnoha	(candidate)
Date	26/04/2023	

Statement 2

I hereby give my consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed	Tomas Svejnoha	(candidate)
Date	26/04/2023	

This document is written in pursuit of a Software Engineering, BSc (Hons) degree at Swansea University.

Abstract

Swansea University employs teaching assistants to run lab sessions for modules throughout the academic years. This project develops a web application for their allocation to modules and labs based on their module preferences, time availability, as well as the requirements of the modules. This document discusses the background information of this project, requirements, software architectures, design patterns, implementation, testing and work needed for future improvements.

Acknowledgements

I am grateful to Dr Mike Edwards for his guidance during this project. I would also like to thank my family, Professor Markus Roggenbach, Petr Hoffmann, Harry Bryant, Victor Cai, Kira Pugh, Naveen Whaind and Anaya Syal, for their advice and moral support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The Computer Science department employs Teaching Assistants to run lab sessions throughout an academic year. There have been many line managers and systems for managing Teaching Assistants over the past years. Unfortunately, these systems were either manual, too complex or seised to exist during the handover of management responsibilities due to staff leaving the department.

The Computer Science department, therefore, requires a standardised system for managing and allocating Teaching Assistants to modules. This document describes an implementation of a web application attempting to deliver such a system.

1.1 Motivation

The motivations for delivering this project remain the same as in the Initial Document [1] produced in October. These are improvement, automation and standardisation of processes related to managing teaching assistants, reducing the number of responsibilities of line managers, and cost reductions for the Computer Science department.

Due to the nature of this project and the fact there were several attempts to implement such a system in the past. The project concluded the probability of the department using the product of this project is low. Therefore, the motivation for this project shifted from delivering a fully functional product to implementing the system with best practices in mind. This document will discuss this motivation later in this document.

1.2 Aims and Objectives

This project outlined a total of three aims. The first aim was to develop a web application to fulfil the requirements of the Computer Science department for managing Teaching Assistants. The second was to ensure the system allocates Teaching Assistants to modules based on the module requirements and their experience and preferences. The third aim was to set a precedent for software development within the department and motivate students to develop, maintain and improve such software.

The project promised to collect requirements and design, implement and test the software product to achieve these aims. It also aimed to produce user and technical reference documentation, which would help guide users when using the system.

1.3 Deliverables

The project aimed to produce four deliverables. The web application and tests, together with technical reference and user documentation. The project successfully delivers the web application, test suites and technical reference documentation. The project failed to produce the user documentation for reasons outlined later in this document.

Chapter 2

Background

This chapter provides information about the project's background, including a high-level overview of the management of teaching assistants and existing solutions on the market.

2.1 Management of Teaching Assistants

The teaching assistants get first recruited by the University's Human Resources (HR) Department, which does all interviews and right-to-work checks required by the law. Once the recruitment is over, each department receives a list of recruited teaching assistants to allocate them to modules and labs. The departments must collect time availability and module preferences from teaching assistants to do this allocation.

The departments then try to assign teaching assistants to modules based on the module requirements while trying to satisfy the module preferences and time availability of individual teaching assistants. If problems occur due to changed timetables, the department attempts to reallocate the affected teaching assistants to different modules. This process of allocation and reallocation is a manual, time-consuming task that not many people want to do, and therefore, software solutions are required to solve this problem.

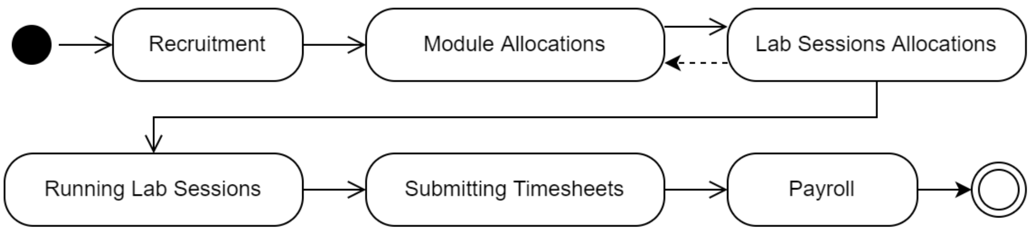


Figure 2.1: UML Activity Diagram of the Teaching Management Process.

2.2 Existing Software Solutions

There are existing software solutions on the market to solve the same problem as this project. The Initial Document [1] discussed these solutions in detail, and this chapter only provides a brief overview of these solutions.

The first of these solutions is a Student Demonstrator Manager [2] developed by Petr Hoffmann, a Swansea University graduate, as his final project. The Computer Science department never deployed this project for bureaucratic reasons, despite several attempts by Petr to facilitate this.

Another software the department could use is UniTime [3], a web application developed by UniTime, s.r.o. UniTime covers more scheduling problems Universities face and is far better suited for use than projects students made as part of their final year. It is an open-source project which also provides limited free support over email and additional commercial services for a fee.

Chapter 3

Requirements

This chapter describes the requirements of this project. The project collected the requirements from various stakeholders throughout the years through discussions and emails. For the purposes of this document, the requirements can have one of the following statuses:

1. *New* - newly added to the project,
2. *Outstanding* - acceptance criteria have not been met,
3. *Cancelled* - no longer considered for delivery,
4. *Completed* - all acceptance criteria have been met.

The following tables contain the requirements defined in the Initial Document [1] their current statuses. The first section outlines the overall requirements for the web application and follows with tables outlining requirements for the different user roles ordered by their privileges.

3.1 Web Application Requirements

The following table describes the general requirements for the web application with their current statuses.

Table 3.1: A table with generic web application requirements.

Requirement ID	Type	Description	Status
Req-Gen-M-01	Mandatory	Must have the following user roles: Administrator and User	Completed
Req-Gen-M-02	Mandatory	Must record Modules, Labs, Lab Schedules, Users, Users' Time Availability, and Module Preferences	Completed
Req-Gen-M-03	Mandatory	Must have the following module roles: Module Coordinator, Lab Coordinator, and Teaching Assistant	Completed
Req-Gen-M-04	Mandatory	Must have a naive algorithm for allocating teaching assistants to modules and lab sessions	Completed
Req-Gen-M-05	Mandatory	Must support limits on the number of undergraduate and postgraduate teaching assistants for each lab session	Outstanding
Req-Gen-O-01	Optional	Should support single sign-on (SSO) with Azure Active Directory	Completed
Req-Gen-O-02	Optional	Should be responsive	Completed
Req-Gen-O-03	Optional	Should display the same in all major browsers	Completed
Req-Gen-O-04	Optional	Should be able to run in Docker	Outstanding
Req-Gen-O-05	Optional	Should have a more advanced algorithm for allocating teaching assistants to lab sessions	Outstanding

3.2 Administrator Requirements

The following table contains the requirements set for the *Administrator* role with their current statuses.

Table 3.2: A table with administrator requirements.

Requirement ID	Type	Description	Status
Req-Admin-M-01	Mandatory	Can create, read, update, and delete modules	Completed
Req-Admin-M-02	Mandatory	Can create, read, update, and delete labs	Completed
Req-Admin-M-03	Mandatory	Can create, read, update, and delete lab schedules	Completed
Req-Admin-M-04	Mandatory	Can create, read, update, and delete users	Completed
Req-Admin-M-05	Mandatory	Can assign and remove module roles	Completed
Req-Admin-M-06	Mandatory	Can add and remove users from labs	Completed
Req-Admin-M-07	Mandatory	Can approve and deny module preference requests	Completed
Req-Admin-M-08	Mandatory	Can run allocation algorithm	Completed
Req-Admin-M-09	Mandatory	Can send questionnaires to all users	Completed
Req-Admin-M-10	Mandatory	Can delete all data in the system	Completed

3.3 Module Coordinator Requirements

The following table contains the requirements set for the *Module Coordinator* role with their current statuses. The table assumes that a user has a *Module Coordinator* role for a module.

Table 3.3: A table with module coordinator requirements.

Requirement ID	Type	Description	Status
Req-ModCo-M-01	Mandatory	Can read and update modules	Completed
Req-ModCo-M-02	Mandatory	Can create, read, update, and delete labs	Completed
Req-ModCo-M-03	Mandatory	Can create, read, update, and delete lab schedules	Completed
Req-ModCo-M-04	Mandatory	Can see staff in modules	Completed
Req-ModCo-M-05	Mandatory	Can approve and decline module preference requests	Completed

3.4 Lab Coordinator Requirements

The following table contains the requirements set for the *Lab Coordinator* role with their current statuses. The table assumes that a user has a *Lab Coordinator* role for a module.

Table 3.4: A table with lab coordinator requirements.

Requirement ID	Type	Description	Status
Req-LabCo-M-01	Mandatory	Can read modules	Completed
Req-LabCo-M-02	Mandatory	Can read and update labs	Completed
Req-LabCo-M-03	Mandatory	Can create, read, update, and delete lab schedules	Completed
Req-LabCo-M-04	Mandatory	Can see staff in modules	Completed

3.5 Teaching Assistant Requirements

The following table contains the requirements set for the *Teaching Assistant* role with their current statuses. The table assumes that a user has a *Teaching Assistant* role for a module.

Table 3.5: A table with teaching assistant requirements.

Requirement ID	Type	Description	Status
Req-TA-M-01	Mandatory	Can read modules	Completed
Req-TA-M-02	Mandatory	Can read labs	Completed
Req-TA-M-03	Mandatory	Can read lab schedules	Completed

Chapter 4

Design

This chapter describes the software architectures and design decisions made during the implementation of the web application.

4.1 Software Architecture

The web application combines different architectural styles. More specifically, Monolithic, N-tier (Multi-layered), and Event-driven architectures.

4.1.1 Monolithic Architecture

Monolithic architecture [4] is probably the most known architectural style. It packages all application code into one binary, which makes it easy to deploy. On the other hand, it has disadvantages which make monolithic architecture unsuitable for modern applications. The main two disadvantages are complexity and scalability. The complexity comes from a large code base where changes can take considerable work. Scalability also comes with challenges, as monolithic application needs to scale as one. This problem makes scaling ineffective when only a part of the application experiences high demand, but due to the nature of monolithic architecture, the whole application needs to scale.

4.1.2 N-tier Architecture

The N-tier architecture [5] is a traditional enterprise architecture which divides code into separate layers. Each layer is responsible for different functionalities, such as presentation, business logic and data access. Each layer can only call lower layers but not the other way around, which makes the applications' code loosely coupled and makes it easier to replace higher levels without refactoring the lower ones.

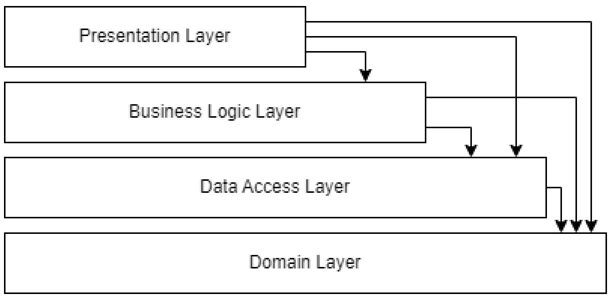


Figure 4.1: N-Tier Architecture diagram.

4.1.3 Event-driven Architecture

Event-driven architecture [6] is a paradigm which defines producers and consumers of events. An event is a significant change of state that needs to propagate throughout an application. This paradigm decouples producers and consumers of events. In other words, publishers do not know which consumers are listening to the events. Consumers are also independent and can handle events in isolation. This loose coupling of producers and consumers allows for their replacement without affecting the rest of the implementation as long as the event definition stays the same. Furthermore, adding additional consumers to an application does not require changes to the existing implementations, which makes an application extensible.

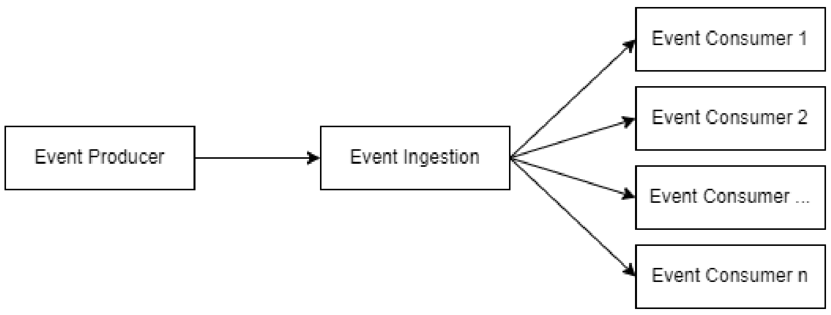


Figure 4.2: Event-driven Architecture diagram.

4.2 UML Class Diagram

Based on the requirements specified in Chapter 3, the project produced the following UML class diagram.

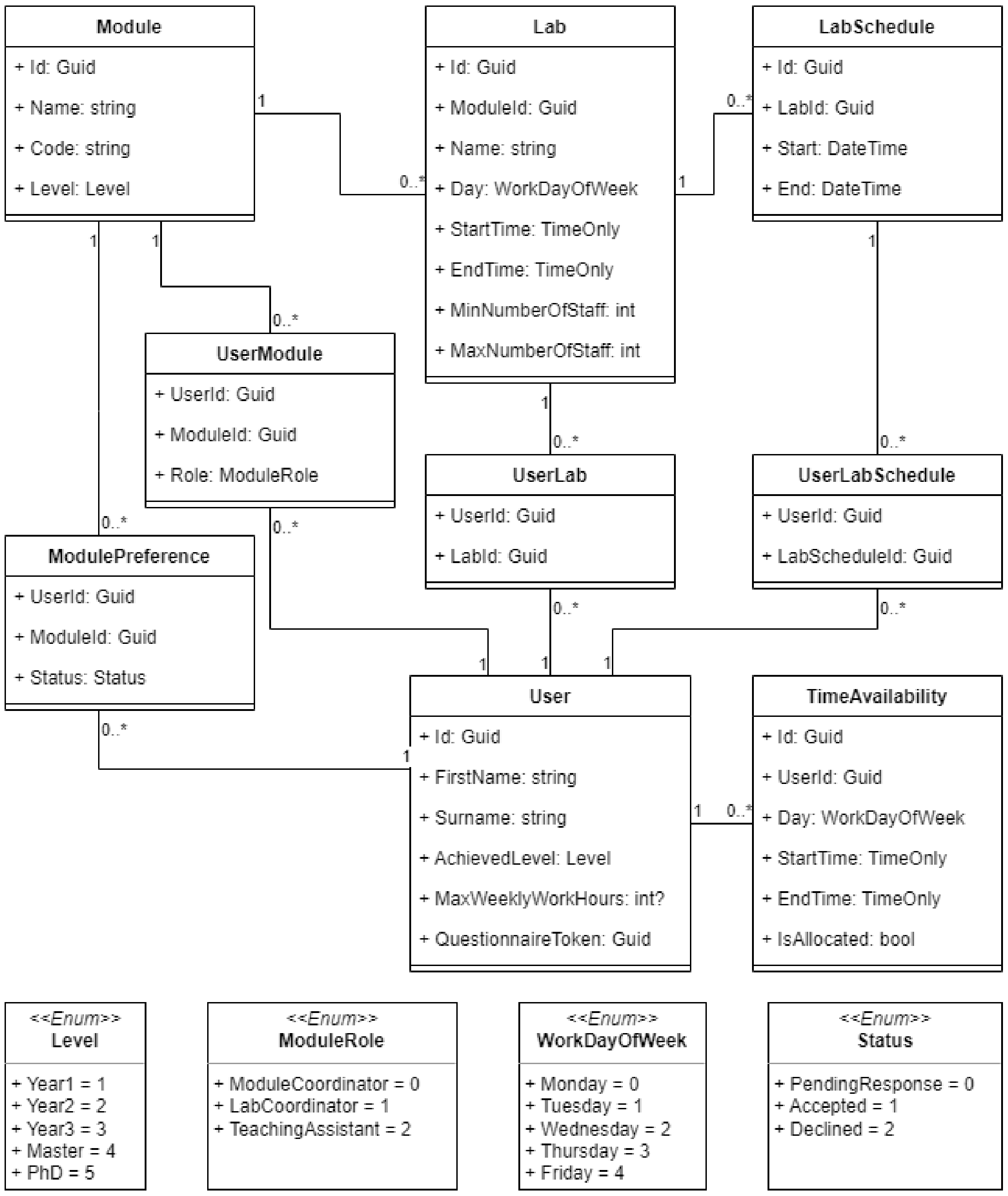


Figure 4.3: UML Class Diagram.

4.3 Database Schema

Based on the UML Class Diagram, the project produced the following database design.

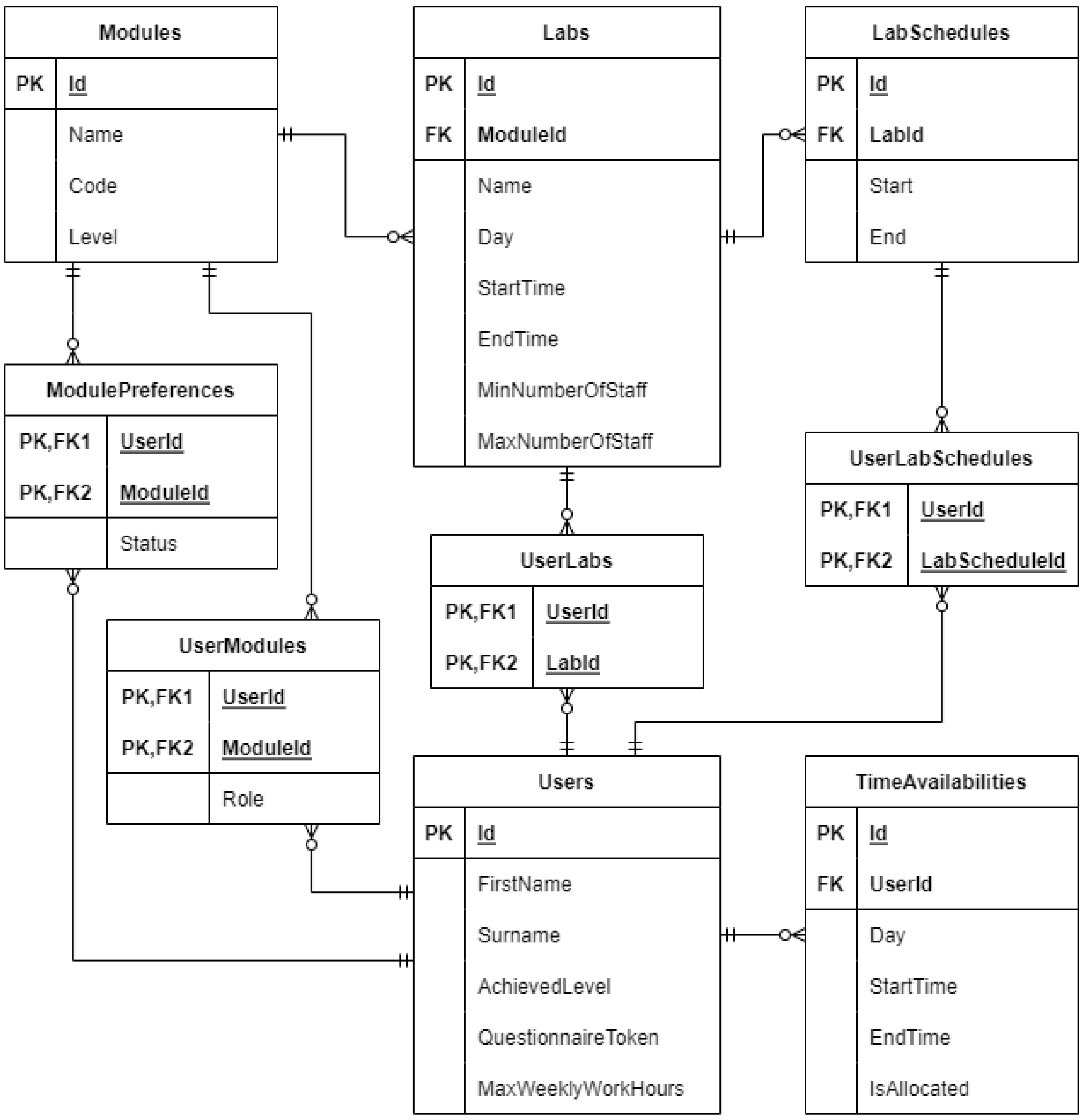


Figure 4.4: Entity–relationship database model.

Chapter 5

Implementation

This chapter provides information about frameworks, libraries and design patterns used in the implementation, as well as the allocation algorithm, which are the main parts of this project.

5.1 Framework

ASP.NET Core [7] is an open-source web application framework for developing web applications in C#. It is for developing both APIs and web applications and supports everything from Model-View-Controller (MVC) to WebAssembly (WASM).

One of the frameworks ASP.NET Core supports is Blazor [8]. Blazor allows developers to write code for both the front-end and back-end in C# by replacing JavaScript on the client. Eliminating JavaScript reduces the knowledge a developer needs to build a web application and speeds up the development process by allowing code reuse.

Blazor has two hosting modes, Blazor WebAssembly and Blazor Server. Each of these is useful in different scenarios, and their comparison is available in the ASP.NET Core documentation [9].

This project uses the Blazor Server hosting mode, which runs the app on a server and updates the client using a SignalR [10] and WebSocket [11] protocol. That differs from the Blazor WebAssembly hosting mode, which downloads and runs applications on the client in a browser and calls servers when needed, acting as an application installed on the underlying operating system.

5.2 Authentication & Authorization

The web application supports Single Sign-On (SSO) with Azure Active Directory (AAD), which allows users to authenticate and authorise with one account and one Identity Provider (IP) across multiple services. That improves security and prevents reusing passwords within a given blast radius. These security claims only hold if the IP system is well fortified. Furthermore, when a breach occurs, the user only has to change one password with the IP, which enhances the overall user experience.

The authentication works over the OAuth 2.1 protocol [12], while the authorisation works over the OpenID Connect protocol [13], an extension of the OAuth protocol. The web application uses these protocols to enforce security policies in the Presentation and Business Logic Layers, ensuring the user can only perform authorised actions.

ASP.NET Core abstracts away the technical details of the OAuth 2.0 and OpenID Connect protocols and makes it easy to use AAD with a few lines of code.

```
internal static IServiceCollection AddPresentation(
  this IServiceCollection services, IConfiguration configuration)
{
  services.AddAuthentication(OpenIdConnectDefaults.AuthenticationScheme)
    .AddMicrosoftIdentityWebApp(configuration.GetSection("AzureAd"));
  services.AddControllersWithViews()
    .AddMicrosoftIdentityUI();

  return services;
}
```

Listing 5.1: Injecting authentication services into a Dependency Injection container.

The code above shows how the project adds AAD Authentication to a Dependency Injection container for later use in the web application. The code below shows the web application using the services injected with the code above in a middleware pipeline.

```
var app = builder.Build();

app.UseAuthentication();
app.UseAuthorization();

app.Run();
```

Listing 5.2: Using authentication middleware in a request pipeline.

5.3 Design Patterns

This project uses several design patterns in every layer of the application's N-tier architecture to achieve high maintainability and testability of the web application. These patterns include Unit of Work and Repository patterns in the Data Access Layer, Mediator, Domain Event, Command and Query Responsibility Segregation and Specification patterns in the Business Logic layer. This section will describe these patterns in more detail.

5.3.1 Unit of Work and Repository Patterns

The Unit of Work pattern [14] aggregates all insert, update and delete operations into a single transaction. This aggregation of database operations improves database performance by avoiding a transaction per operation which requires significantly more resources.

The Repository pattern [15] provides a repository object per table and handles all the database operations on that table. This pattern significantly improves the testability of the application logic and allows developers to replace the repositories with mock repositories during testing.

The following figure shows the usage of the Repository pattern in implementation and unit tests compared to direct access to a database.

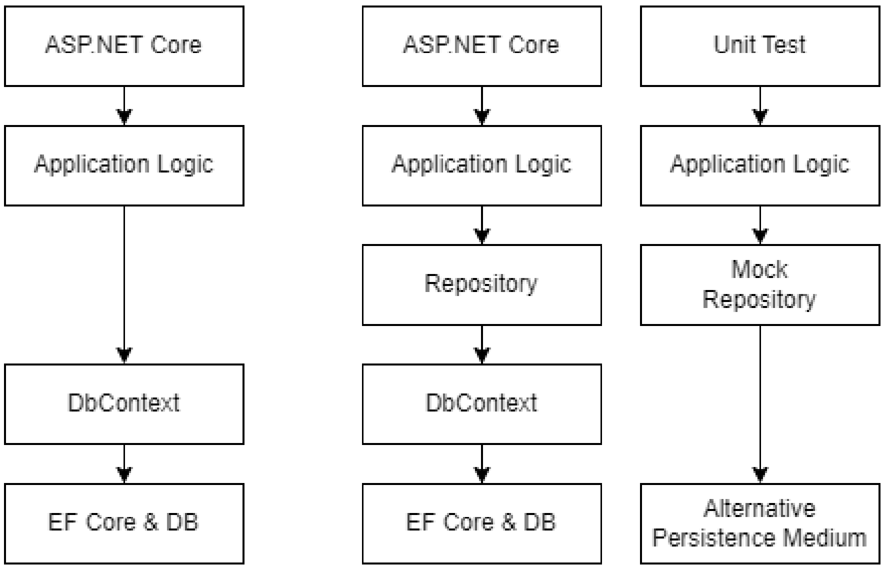


Figure 5.1: Repository Pattern.

5.3.2 Specification Pattern

The Specification pattern [16] encapsulates business logic into a single object, promoting reusability, and follows the Single Responsibility Principle, the first of the SOLID principles. Dividing business logic into Specifications improves the testability and maintainability of the overall system. This project uses the Specification with the Repository pattern to query data from a database. Without this pattern, the project would have a method per query in each Repository, which would grow the codebase and impact the maintainability and testability.

5.3.3 Mediator Pattern

The Mediator pattern [17] is a behavioural design pattern that promotes loose coupling of code by providing a central mediator object that handles all communication between objects. This pattern removes interdependencies which is a good software development practice as it improves the maintainability and testability of the source code. This pattern also allows us to specify behaviour for each interaction, such as authentication, logging, performance monitoring, etc.

The following figure compares objects with interdependencies and objects communicating through a Repository pattern. This figure demonstrates the Repository pattern removes chaotic interdependencies and provides a systematic structure for the code.

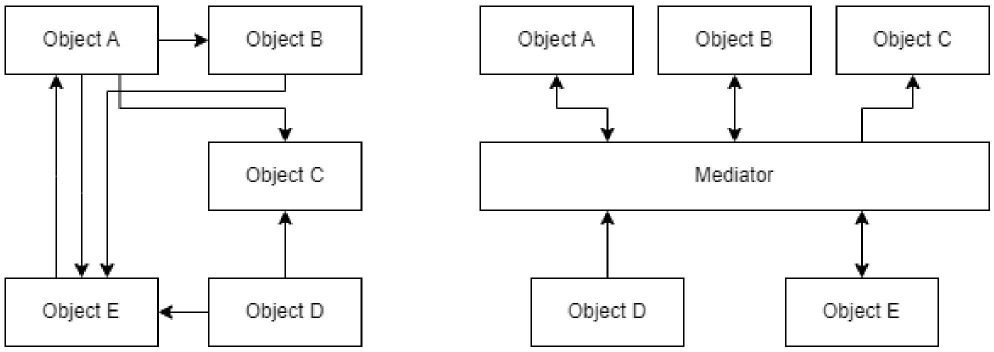


Figure 5.2: Mediator Pattern.

5.3.4 Command and Query Responsibility Segregation Pattern

The Command and Query Segregation (CQRS) pattern [18] separates read and write database operations. Queries represent read operations and cannot change the state of the system. Commands represent write operations (create, read, delete) and change the system state. This pattern often utilises the Mediator pattern described in the previous section to distribute commands and queries to handlers.

The following diagram shows how can presentation layer access and modify the data store using multiple commands and queries.

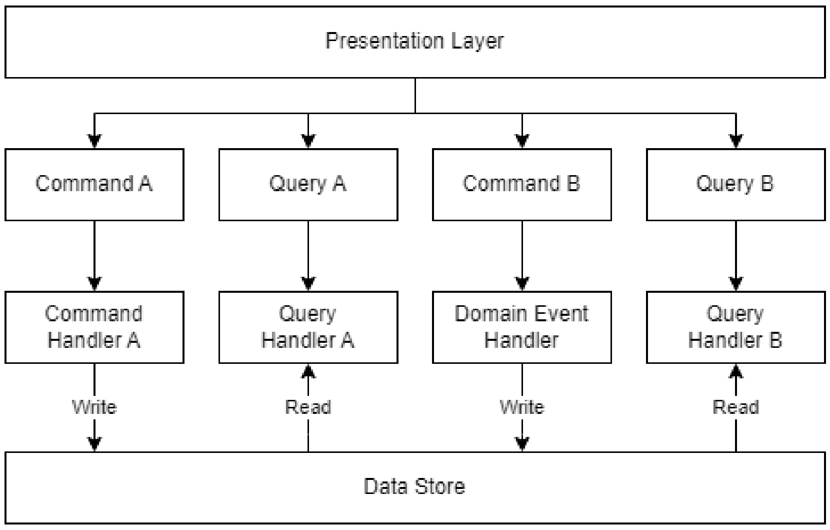


Figure 5.3: Command and Query Responsibility Segregation pattern.

5.3.5 Domain Event Pattern

The Domain Event pattern [19] is used for the distribution of events within the same domain to propagate changes. Each Event can have one or more Event Handlers who capture and handle it. This pattern promotes the decoupling of code to improve maintainability and testability. Like the CQRS pattern, this pattern often utilises the Mediator pattern to broadcast Events to the Event Handlers.

5.4 Libraries

The project uses the following libraries in its implementation. Some of those libraries implement the design patterns described in the section above.

5.4.1 Entity Framework Core

This project uses Entity Framework Core (EF Core) [20] to store and query data from an MS-SQL database. EF Core is an Object-Relational Mapping (ORM) framework for .NET applications, which maps data from a database to application-defined models and abstracts raw SQL queries from the developers.

EF Core implements a Unit of Work pattern via a DbContext class and a Repository pattern via a DbSet class. Although the EF Core implements the Repository pattern and allows to use of multiple databases using various drivers, the web application adds another layer of abstraction on top of the EF Core-provided Repository pattern. This extra layer of abstraction allows for easy unit testing, where it is not desirable to use a database. Furthermore, if the requirements change in the future, the extra abstraction enables the project to migrate to a different ORM framework without refactoring other layers.

Samples of code with EF Core and Repositories are available in the appendix A.1.

5.4.2 MediatR

The MediatR library [21] implements the Mediator pattern, which the project uses to implement the CQRS and Domain Event patterns. Samples of code implementing the CQRS and Domain Event pattern are in the appendix A.2.

5.4.3 Ardalis.Specification

The project uses the Ardalis.Specification library [22] to implement the Specification pattern, which is described in section 5.3.2, for database queries. Examples of Specification objects are available in appendix A.3.

5.4.4 AutoMapper

The project uses the AutoMapper library [23] to create a mapping between database entities, defined in the Domain layer, and Models in the Application Layer. The AutoMapper uses Mapping Profiles to specify a mapping of properties between source and destination objects and any transformation and conversion of values. Code samples from the project are available in appendix A.4.

5.4.5 FluentValidation

The FluentValidation library [24] provides data validation capabilities using the Fluent Interface [25]. The Fluent Interface, also called Fluent API, is a programming style in which methods return appropriate objects and allow developers to write them up using method chaining. Fluent Interface provides more readable code and is less prone to programming errors. Samples of code using the FluentValidation library are available in appendix A.5.

5.4.6 MudBlazor

The front-end implementation of the project uses a MudBlazor [26], a component library for Blazor applications which provides ready-to-use web components. These components include everything from simple text elements to complex tables and layouts. This use of this library significantly sped up the development of the front-end part of the project. A sample of a code using the MudBlazor is available in appendix A.6.

5.5 Allocation Algorithm

The project implements a naive algorithm for allocating Teaching Assistants (TA) to modules and labs based on the TA's time availability, module preferences and requirements of modules.

The diagram below shows the implementation of the algorithm. The algorithm enumerates over a set of labs, copies over existing allocation and selects TAs to fill in the remaining spaces. The selection of the TAs is in priority order.

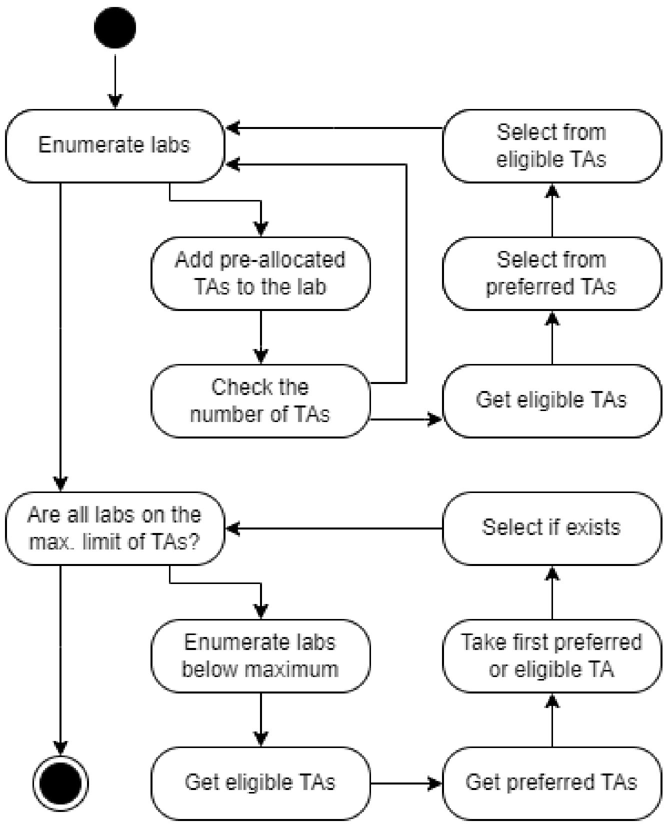

Enumerate labs
Select from eligible TAs
Add pre-allocated TAs to the lab
Select from preferred TAs
Check the number of TAs
Get eligible TAs
Are all labs on the max. limit of TAs?
Select if exists
Enumerate labs below maximum
Take first preferred or eligible TA
Get eligible TAs
Get preferred TAs

Figure 5.4: UML diagram of the allocation algorithm.

The algorithm first takes TAs who pre-agreed with the module coordinator before selecting from the remaining TAs eligible to teach in a given module.

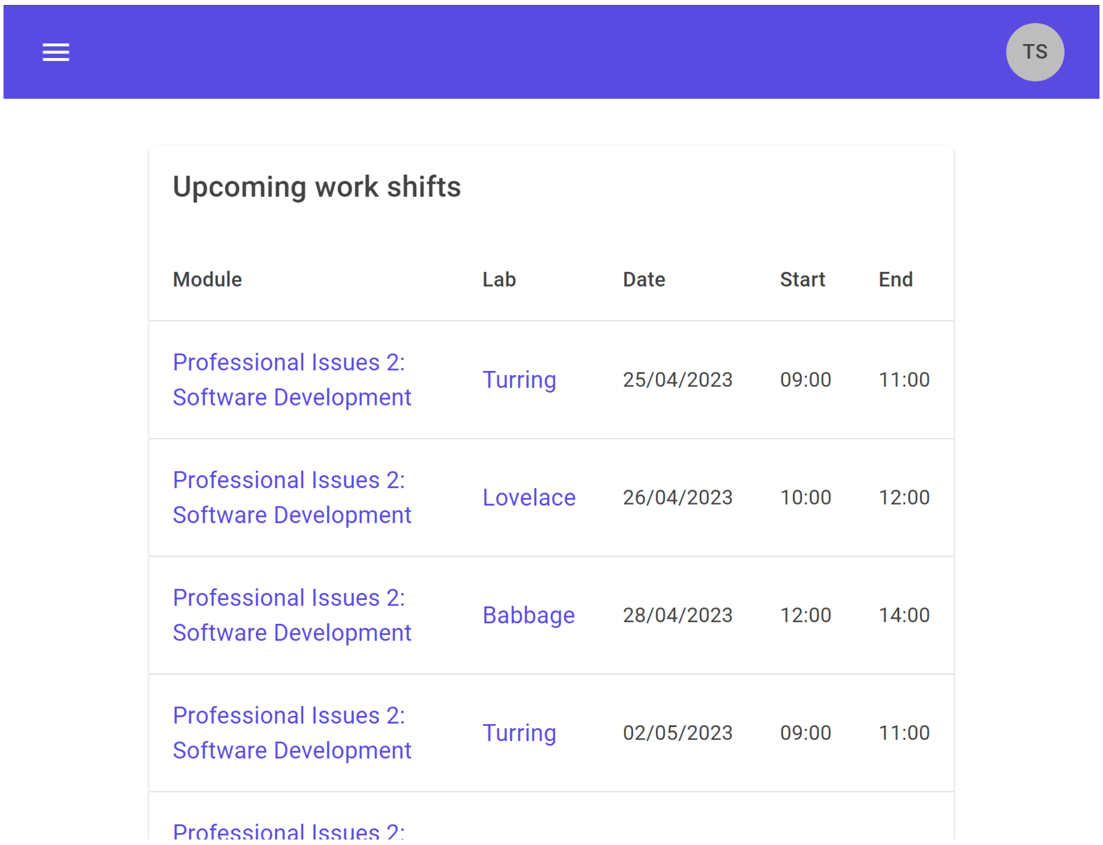
The algorithm obtains a list of eligible TAs by filtering the TAs already allocated to a lab, TAs who did not achieve the required year of study, TAs with declined requests to teach in a given module, TAs who reached their maximum number of hours and TAs who are not available to teach in the lab due to time constraints. This set is ordered by the TA's year of study in descending order and then by the number of allocated hours in ascending order. This ordering ensures the algorithm picks TAs with the minimum number of hours first.

After allocating the minimum number of TAs, the algorithm continues to assign any remaining eligible TAs to modules which have not yet reached the maximum number of allowed TAs. The algorithm takes the preferred TAs first and then any remaining TAs to fill in the rest of the places.

5.6 Web Application

The resulting product of the project is a responsive web application created using the frameworks, protocols, and design patterns described above.

The web application has a dashboard showing a table with information about modules, lab groups, and days and times of individual labs, allowing the teaching assistants to get up-to-date information about their upcoming work shifts.

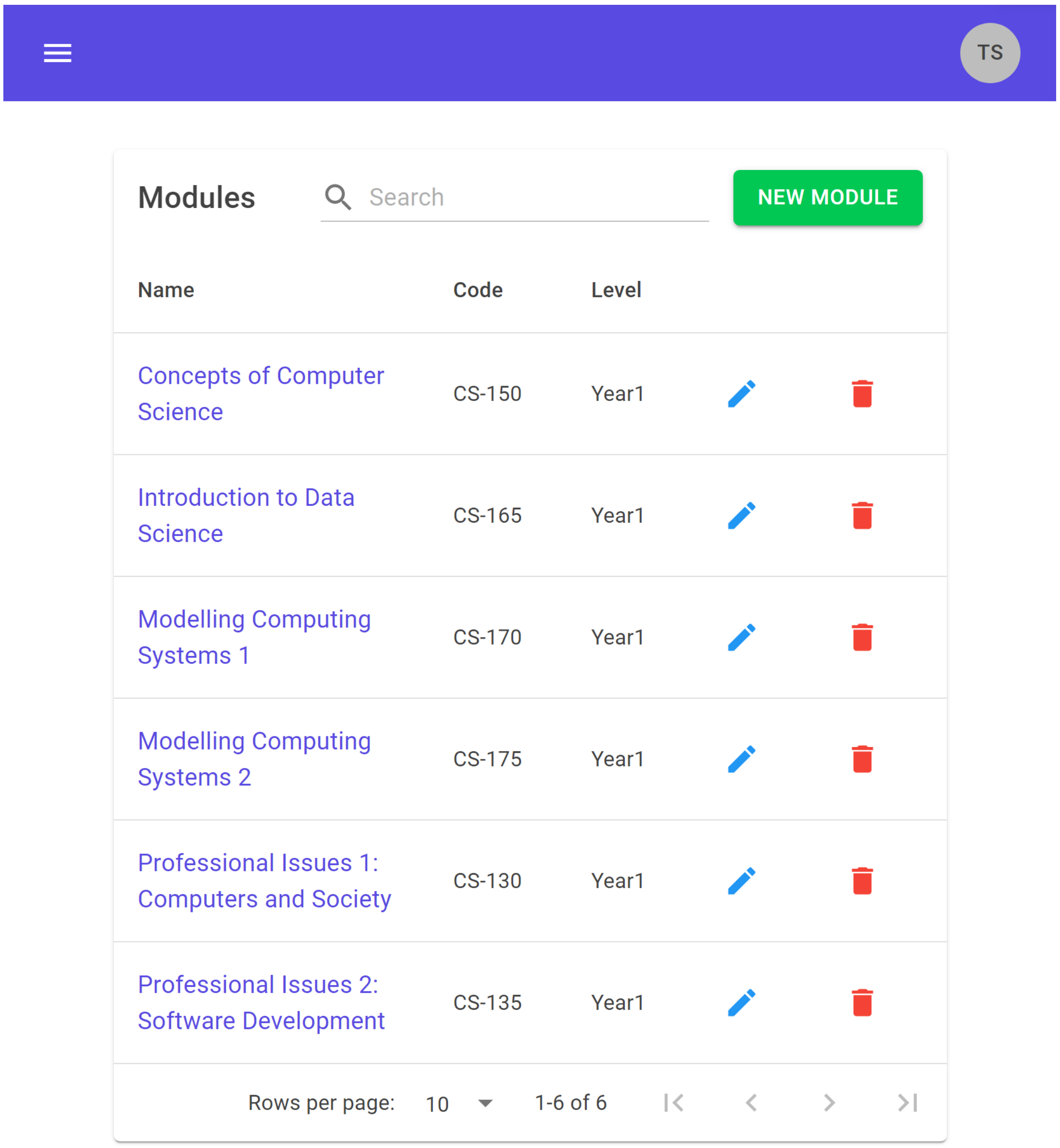


Upcoming work shifts

Module	Lab	Date	Start	End
Professional Issues 2: Software Development	Turring	25/04/2023	09:00	11:00
Professional Issues 2: Software Development	Lovelace	26/04/2023	10:00	12:00
Professional Issues 2: Software Development	Babbage	28/04/2023	12:00	14:00
Professional Issues 2: Software Development	Turring	02/05/2023	09:00	11:00
Professional Issues 2:				

Figure 5.5: Main Dashboard with upcoming work shifts.

Users can see a list of modules on the Modules page in a sortable, searchable and paginated table showing the name, code and level of all modules. The search functionality is case-insensitive and searches all columns.

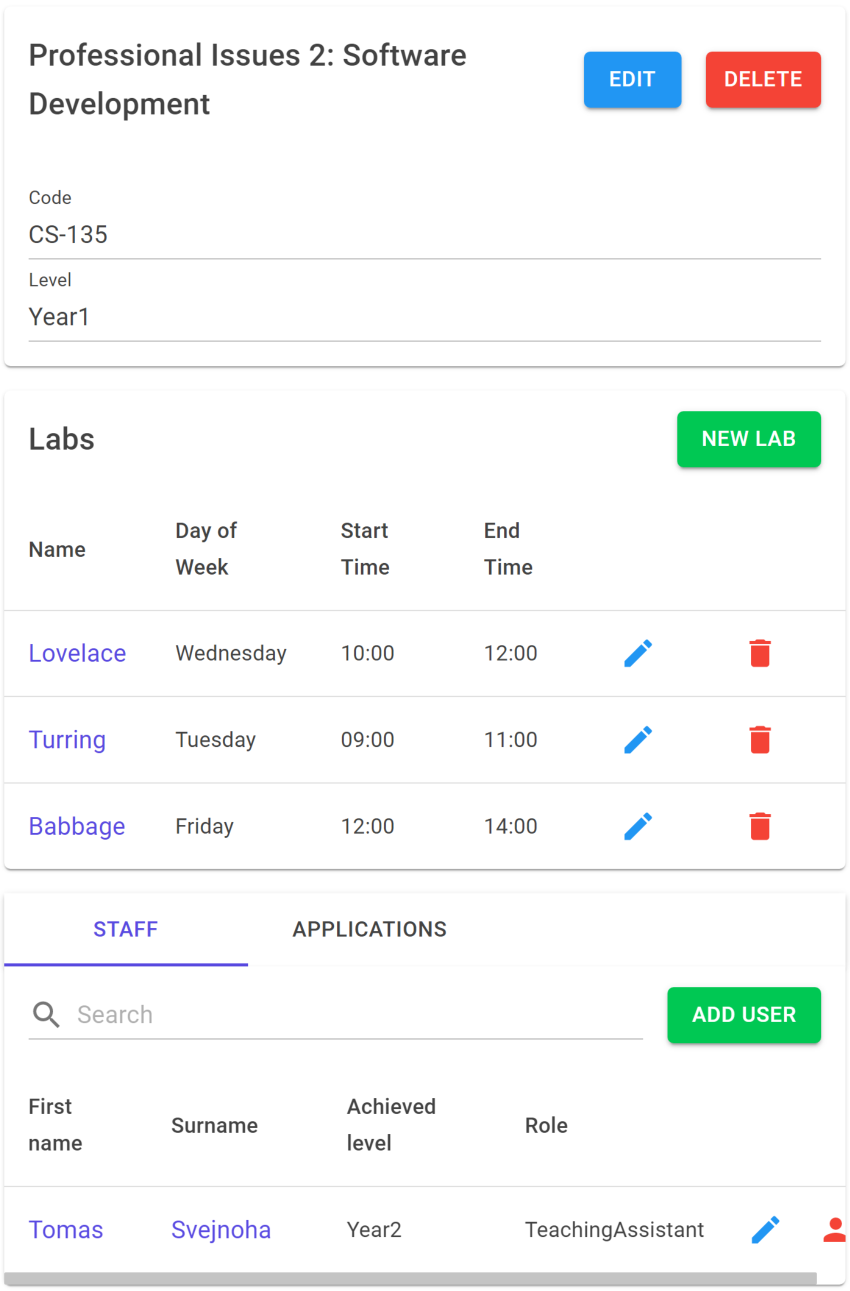


Name	Code	Level
Concepts of Computer Science	CS-150	Year1
Introduction to Data Science	CS-165	Year1
Modelling Computing Systems 1	CS-170	Year1
Modelling Computing Systems 2	CS-175	Year1
Professional Issues 1: Computers and Society	CS-130	Year1
Professional Issues 2: Software Development	CS-135	Year1

Rows per page: 10 1-6 of 6

Figure 5.6: Modules page.

Similarly, the users can see details of each module on a Module Details page, which shows the basic information about the module, like name, code and level, table with names, dates and times of all labs, members of staff who work in the module and requests to work in the module.



Professional Issues 2: Software Development

Code: CS-135

Level: Year1

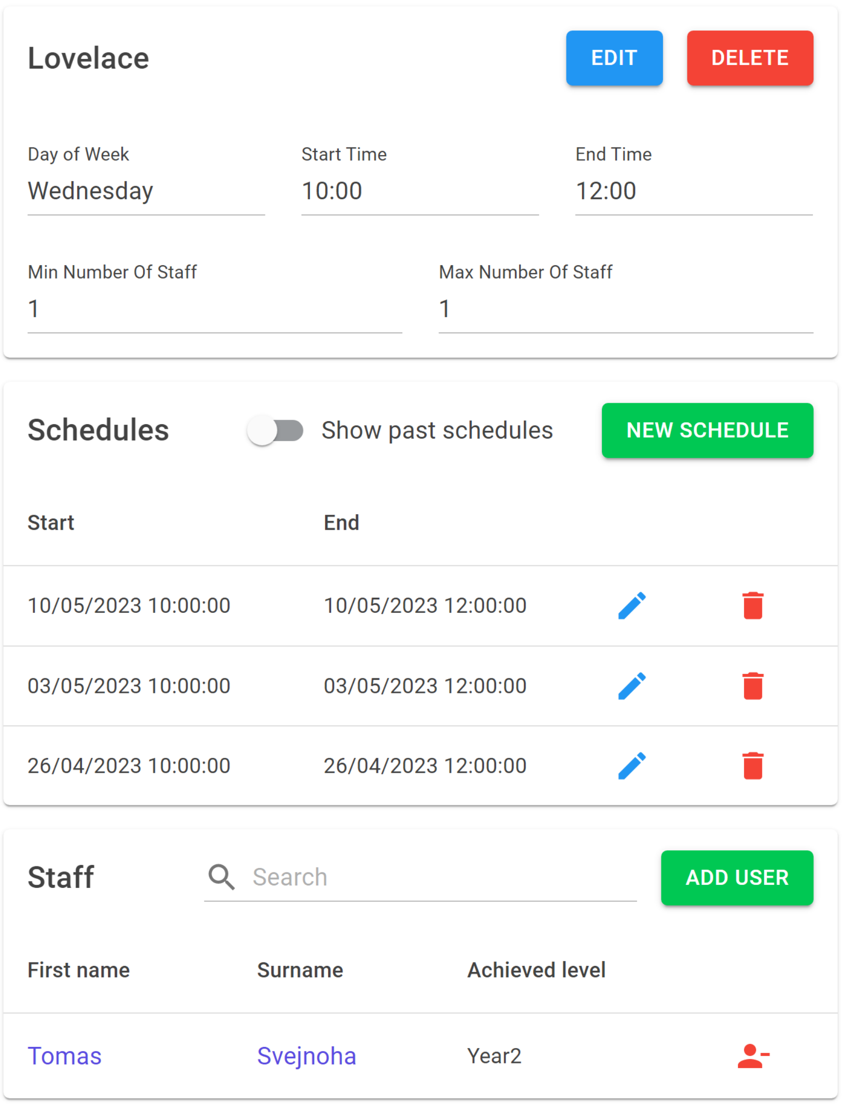
Labs

Name	Day of Week	Start Time	End Time
Lovelace	Wednesday	10:00	12:00
Turring	Tuesday	09:00	11:00
Babbage	Friday	12:00	14:00

First name	Surname	Achieved level	Role
Tomas	Svejnoha	Year2	TeachingAssistant

Figure 5.7: Module detail page.

The detail of each lab is on the Lab Detail page, which is available through the module detail page. The page shows the basic information about the lab, like the date, start and end times, and the limits for the number of staff who can work there. The page also shows a table of all future and past lab schedules, with the past ones hidden by default. Users can see the past labs by switching the "Show past schedules" switch. A table showing the staff members working in the lab is also present at the bottom of the page.



Lovelace

Day of Week: Wednesday
Start Time: 10:00
End Time: 12:00
Min Number Of Staff: 1
Max Number Of Staff: 1

Schedules — Show past schedules

Start	End
10/05/2023 10:00:00	10/05/2023 12:00:00
03/05/2023 10:00:00	03/05/2023 12:00:00
26/04/2023 10:00:00	26/04/2023 12:00:00

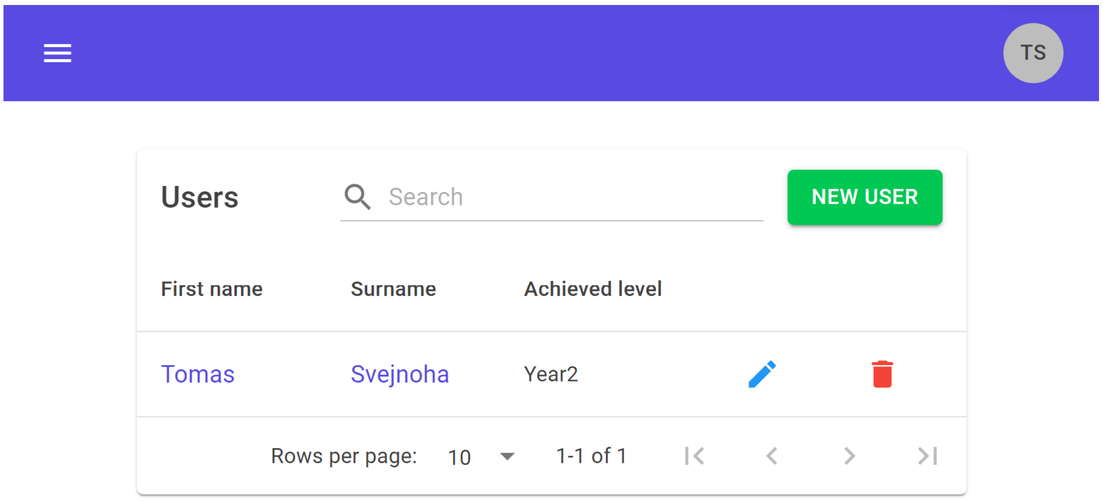
Staff

First name	Surname	Achieved level
Tomas	Svejnoha	Year2

Figure 5.8: Lab Detail page.

The web application enforces different levels of access control in line with the requirements defined in chapter 3. The screenshots display pages under administrator privileges, and some information and buttons are not visible to users with lower permissions.

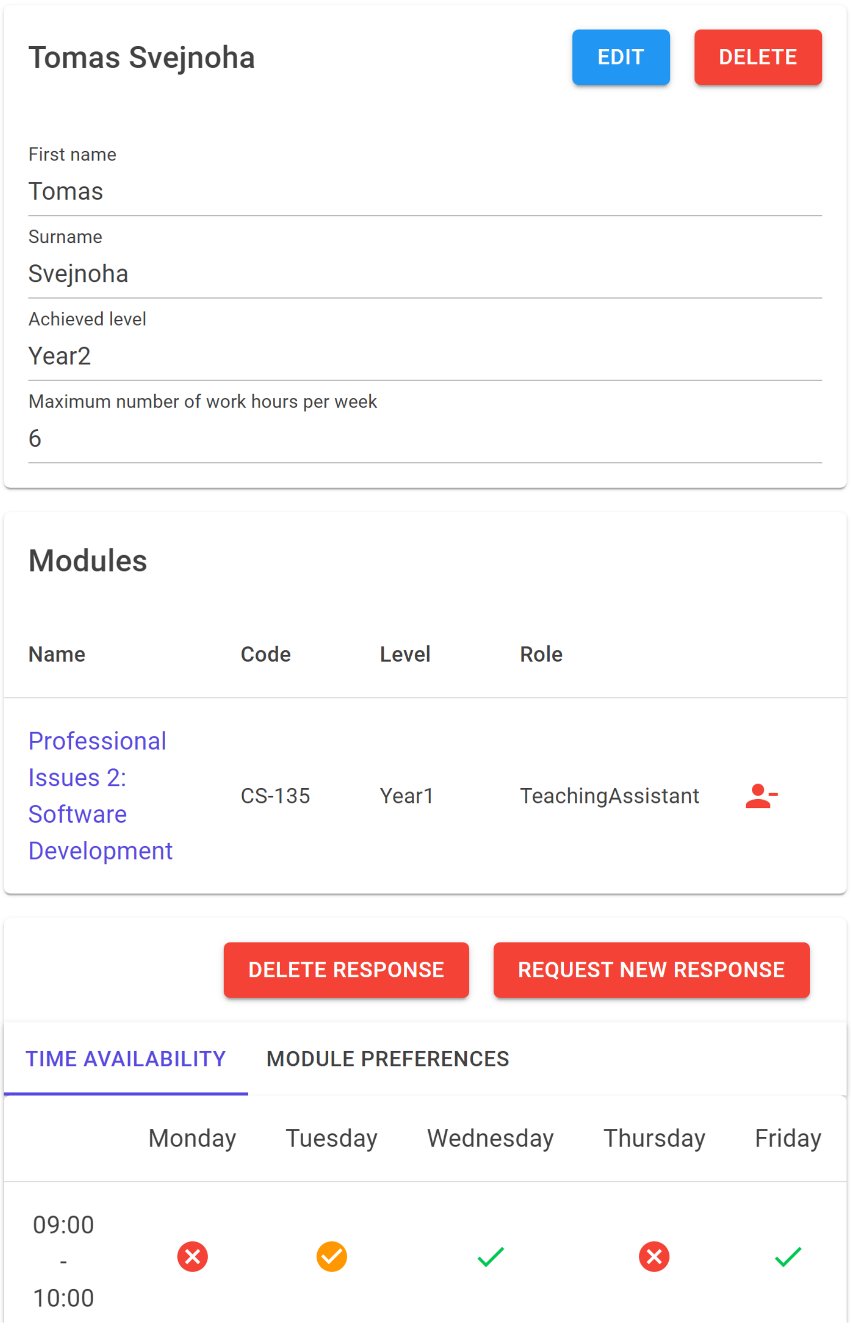
The web application displays users in a sortable, searchable and paginated table, showing the first names, surnames and achieved levels of study.



First name	Surname	Achieved level
Tomas	Svejnoha	Year2

Figure 5.9: Users page.

User details are on a separate page showing information about the user, allocated modules with roles, time availability reflecting all allocations and module preferences selected in the questionnaire.



Tomas Svejnoha

First name
Tomas

Surname
Svejnoha

Achieved level
Year2

Maximum number of work hours per week
6

Modules

Name	Code	Level	Role
Professional Issues 2: Software Development	CS-135	Year1	TeachingAssistant

	Monday	Tuesday	Wednesday	Thursday	Friday
09:00 - 10:00	✗	✓	✓	✗	✓

Figure 5.10: User Detail page.

When the administrators add new users, the web application queries Azure Active Directory (AAD) with a given search string. The AAD searches for a given query string in a display name, first name, surname or mail of all users in an AAD tenant. That enhances the security by not allowing administrators to add users outside the University not cleared by the Human Resources department.

Administrators can use tools to manage the system using various tools. One allows the administrators to send questionnaires to all users to fill in their time availability and module preferences before allocating them to modules. Another one automatically assigns users to modules based on their time availability and module preferences collected using the questionnaires, considering any existing allocations. The last tool wipes the database and prepares the web application for the next academic year.

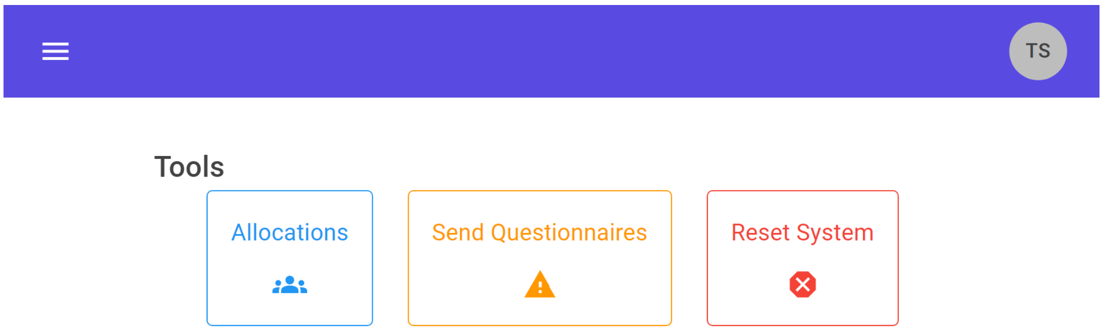


Figure 5.11: Administrator tools.

Chapter 6

Testing

This chapter describes the testing part of the project, including Unit Testing, Integration Testing, test results and libraries used in implementing test suites.

6.1 Unit Testing

This project uses Unit Testing [27] as the first testing approach. Unit Testing is a software testing methodology that tests individual units of source code in isolation and compares actual and expected outputs without any external dependencies.

To resolve the problem of external dependencies, Unit Testing uses mocks, stubs, fakes and dummies to substitute them. This project uses Mocks, which emulate other objects needed to test a given Unit.

6.2 Integration Testing

Integration Testing [28] is the following testing methodology this project uses to ensure the correctness of the implementation. Unlike Unit Testing, it tests Units in groups to ensure they work together seamlessly.

It is important to note that Integration Testing does not use mocks, stubs, fakes or dummies for its test suites. Integration Testing should also only come after Unit Testing, which cannot test for interactions between objects and therefore is prone to missing logical bugs in the implementation of the system.

6.3 Libraries

The project uses NUnit, Moq, FluentAssertions, and Respawn libraries to implement test suites.

6.3.1 NUnit

The NUnit library [29] is an open-source Unit Testing framework for .NET applications. It provides capabilities comparable to Unit Testing frameworks like MSTest and xUnit for .NET or JUnit for Java. These capabilities mainly focus on running test cases and comparing expected and actual outputs. This document does not provide a comparison of testing frameworks mentioned above, nor does it go into details of the NUnit framework, as it is out of scope for this document.

6.3.2 FluentAssertions

The FluentAssertions library [30] provides methods following the Fluent Interface in the same way as the FluentValidation library described previously in this document in section 5.4.5. Samples of test cases using the NUnit framework and FluentAssertions library are available in appendix B.1.

6.3.3 Moq

The project uses the Moq library [31] for mock implementations in test suites. As described previously in this document, mocks emulate the behaviour of other objects.

In Integration Testing, the project implements mock authentication services to prevent pollution of AAD and a DateTime service, which keeps the test results consistent.

In Unit Testing, the test suites implement mocks of a database and DateTime service for the same purposes as Integration Tests.

6.3.4 Respawn

In Integration Testing, it is crucial to ensure that test cases do not affect the outcome of other test cases. This project uses a database for Integration Testing, which can lead to this undesirable behaviour. The project resets the database to its last known good state using a Respawn library [32] to prevent this and ensures that each test case starts with the same database state. Each test case is then responsible for inserting the data it needs.

6.4 Testing Approach

The project does not define set coverage criteria for the test suites commonly used in formal software testing. Instead, the test cases try to test qualitatively different behaviours of the critical parts of the system without introducing unnecessary test cases.

Each test case has three main steps: Preparation, Execution and Evaluation. The Preparation step prepares data for a given test case. The Execution step executes the System Under Test (SUT) with the data, and the Evaluation step evaluates the output of the SUT against expected results.

6.5 Test Results

Not many software products in non-critical business environments achieve 100% test coverage. The costs of developing and maintaining such test suites outweigh their benefits and are not financially viable. It is better to focus on testing complex and critical parts of software products and leave the non-critical parts with little to no changes and a very low probability of bugs to manual testing. Furthermore, enforcing unnecessarily high test coverage requirements can lead to useless test cases whose sole purpose is the required test coverage.

This project has test suites for business logic and leaves the user interface to manual testing by the testers and developers. It has 344 test cases, 183 for Unit Testing and 161 for Integration Testing. These test cases provide an overall 77.72% coverage with 6060 covered lines, 27 partially covered lines, and 1710 lines not covered by any test cases. The following table provides a high-level overview of code coverage for each part of the project.

Table 6.1: A table with code coverage results.

Namespace	Covered (Lines)	Partially Covered (Lines)	Not Covered (lines)	Covered (% Lines)
Core.Application	2656	11	39	98.15%
Core.Application.Allocation	278	12	13	91.75%
Core.Domain	262	0	39	87.04%
Infrastructure.Persistence	2839	4	1516	65.13%
Infrastructure.Shared	25	0	103	19.53%

The table above does not provide a detailed insight into the code coverage of individual classes and methods. It would be unfeasible to provide the full code coverage report in detail in this document. Therefore, this document only provides a high-level overview of the system and leaves it to the testers and developers to explore the test suite in an IDE.

Chapter 7

Conclusion

In this chapter, we will reflect on the outcomes of this project, its successes, failures, project management, and the work required for future improvements.

7.1 Project Management

The project followed the Waterfall and Incremental Build software development lifecycle (SDLC) models as planned in the Initial Document [1] in October. The Waterfall model was the dominant SDLC for this project, while the Incremental Build model played a minor part towards the end when delivering improvements and optional features.

The project is slightly behind schedule against the original plan from October. Since projects are known to run over budget and time, this project anticipated delays and mitigations were in place to compensate for them. These mitigations allowed the project to produce the minimum viable product without impacting the deadline.

7.2 Project Outcomes

The project produced three out of four of its planned deliverables. It successfully delivered a web application, software tests and technical reference documentation. The deliverables will be published in June on GitHub for other students to contribute.

The project failed to deliver user documentation due to more work on implementation, software tests and other University projects. More specifically, Advanced Object Oriented Programming and Web Applications coursework.

7.3 Future Work

Due to the nature of this project and the fact there were previous attempts to implement similar systems, which failed to make it to production, the focus shifted from delivering production-ready software to writing clean code and demonstrating the use of multiple design patterns. This shift will allow the next student to easily continue working on this project after reading this document and the source code of the web application.

Future work should mainly focus on the allocation algorithm and the user interface (UI), which does not provide a great user experience (UX). The decision to use MudBlazor caused most of these UX problems as it restricts the ability to customise the UI. The next major problem in the UX is the navigation between pages, as the user cannot go to a module page from a lab page without using the ability of the browser to go back and has to navigate to the module through the Modules page.

If this project were to be deployed and used by the department, it would also be worth exploring the option of connecting to other applications and APIs within the University to exchange data. That could be obtaining a list of teaching assistants hired by the human resources department, synchronising with timetables or feeding data into a payroll system.

7.4 Closing Remark

In conclusion, the project was successful and allowed me to improve my software engineering skills. There were some challenges when choosing technology and balancing work on this project with other university work. I also had to abandon ideas for implementing additional features like the ones suggested above as I did run out of time.

Bibliography

[1] Tomas Svejnoha. (2022) Initial Document.

[2] Petr Hoffmann. (2021) Student Demonstrator Manager. Accessed on 07.04.2023. [Online]. Available: https://gitlab.com/uni.hoffic.cz/student-demonstrator-manager

[3] UniTime, s.r.o. (2022) UniTime. Accessed on 07.04.2023. [Online]. Available: https://www.unitime.org/

[4] G. Blinowski, A. Ojdowska, and A. Przybyłek, "Monolithic vs. microservice architecture: A performance and scalability evaluation," *IEEE Access*, vol. 10, pp. 20 357–20 374, 2022.

[5] Microsoft Corporation. N-tier architecture style. Accessed on 07.04.2023. [Online]. Available: https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/n-tier

[6] ——. Event-driven architecture style. Accessed on 07.04.2023. [Online]. Available: https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/event-driven

[7] .NET Foundation and Contributors. ASP.NET Core. Accessed on 07.04.2023. [Online]. Available: https://github.com/dotnet/aspnetcore

[8] ——. Blazor. Accessed on 07.04.2023. [Online]. Available: https://dotnet.microsoft.com/en-us/apps/aspnet/web-apps/blazor

[9] ——. Blazor Hosting Models. Accessed on 07.04.2023. [Online]. Available: https://learn.microsoft.com/en-gb/aspnet/core/blazor/hosting-models

[10] ——. SignalR. Accessed on 07.04.2023. [Online]. Available: https://github.com/dotnet/aspnetcore/tree/main/src/SignalR

[11] Internet Engineering Task Force. The WebSocket Protocol. Accessed on 07.04.2023. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8441

[12] D. Hardt, A. Parecki, and T. Lodderstedt, "The OAuth 2.1 Authorization Framework," Internet Engineering Task Force, Internet-Draft draft-ietf-oauth-v2-1-08, Mar. 2023, work in Progress, Accessed on 08.04.2023. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-oauth-v2-1/08/

[13] OpenID Foundation. OpenID Connect Specifications. Accessed on 07.04.2023. [Online]. Available: https://openid.net/developers/specs/

[14] Martin Fowler. Unit of Work. Accessed on 07.04.2023. [Online]. Available: https://martinfowler.com/eaaCatalog/unitOfWork.html

[15] ——. Repository. Accessed on 07.04.2023. [Online]. Available: https://www.martinfowler.com/eaaCatalog/repository.html

[16] Steve Smith. Specification Pattern. Accessed on 07.04.2023. [Online]. Available: https://specification.ardalis.com/

[17] Data & Object Factory, LLC. C# Mediator. Accessed on 07.04.2023. [Online]. Available: https://www.dofactory.com/net/mediator-design-pattern

[18] Martin Fowler. CQRS. Accessed on 07.04.2023. [Online]. Available: https://martinfowler.com/bliki/CQRS.html

[19] Microsoft Corporation. Domain events: Design and implementation. Accessed on 07.04.2023. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/domain-events-design-implementation

[20] .NET Foundation and Contributors. Entity Framework Core. Accessed on 07.04.2023. [Online]. Available: https://github.com/dotnet/efcore

[21] Jimmy Bogard. MediatR. Accessed on 07.04.2023. [Online]. Available: https://github.com/jbogard/MediatR

[22] Steve Smith. Specification. Accessed on 07.04.2023. [Online]. Available: https://github.com/ardalis/Specification

[23] Jimmy Bogard. AutoMapper. Accessed on 07.04.2023. [Online]. Available: https://github.com/AutoMapper/AutoMapper

[24] Jeremy Skinner. Fluent Validation. Accessed on 07.04.2023. [Online]. Available: https://github.com/FluentValidation/FluentValidation

[25] Martin Fowler. Fluent Interface. Accessed on 07.04.2023. [Online]. Available: https://martinfowler.com/bliki/FluentInterface.html

[26] Gardnet AB. MudBlazor. Accessed on 07.04.2023. [Online]. Available: https://github.com/MudBlazor/MudBlazor

[27] M. Olan, "Unit testing: test early, test often," *Journal of Computing Sciences in Colleges*, vol. 19, no. 2, pp. 319–328, 2003.

[28] P. C. Jorgensen and C. Erickson, "Object-oriented integration testing," *Communications of the ACM*, vol. 37, no. 9, pp. 30–38, 1994.

[29] Charlie Poole, Rob Prouse. NUnit 3 Framework. Accessed on 07.04.2023. [Online]. Available: https://github.com/nunit/nunit

[30] J. N. Dennis Doomen and Contributors. Fluent Assertions. Accessed on 07.04.2023. [Online]. Available: https://github.com/fluentassertions/fluentassertions

[31] Daniel Cazzulino and Contributors. moq v5. Accessed on 07.04.2023. [Online]. Available: https://github.com/moq/moq

[32] Jimmy Bogard. Respawn. Accessed on 07.04.2023. [Online]. Available: https://github.com/jbogard/Respawn

Appendix A

Implementation Libraries

A.1 Entity Framework Core

A.1.1 IApplicationDbContext Interface

```
public interface IApplicationDbContext {
  public DbSet<Module> Modules { get; }
  // Other DbSets; shortened for brevity

  Task<int> SaveChangesAsync(CancellationToken cancellationToken = default);
  void Migrate();
}
```

Listing A.1: An implementation of IApplicationDbContext interface.

A.1.2 IRepository Interface

```
public interface IRepository<T> where T : BaseEntity {
  Task<T?> GetItemAsync(ISingleResultSpecification<T> specification,
      CancellationToken cancellationToken);

  IEnumerable<T> GetItems(ISpecification<T> specification);

  Task<int> GetItemsCountAsync(ISpecification<T> specification, CancellationToken
      cancellationToken);

  Task<IEnumerable<T>> DeleteAllAsync(CancellationToken cancellationToken);
}
```

Listing A.2: An implementation of a generic IRepository interface.

A.2 MediatR

A.2.1 CQRS Pattern

```
public sealed class Delete {
  public sealed class Command : IRequest<Response> {
    public Command(Guid labId) {
      LabId = labId;
    }

    public Guid LabId { get; }
  }

  public sealed class Response {
    public Response(LabModel? resource) {
      Resource = resource;
    }

    public LabModel? Resource { get; }
  }

  internal sealed class CommandHandler : IRequestHandler<Command, Response> {
    public CommandHandler(ILabRepository repository, IMapper mapper) {
      Repository = repository;
      Mapper = mapper;
    }

    private ILabRepository Repository { get; }
    private IMapper Mapper { get; }

    public async Task<Response> Handle(Command request, CancellationToken
        cancellationToken) {
      var lab = await Repository.DeleteItemAsync(id: request.LabId,
                              cancellationToken: cancellationToken);

      return lab is not null
          ? new Response(Mapper.Map<Lab, LabModel>(lab))
          : new Response(null);
    }
  }
}
```

Listing A.3: An implementation of a CQRS pattern.

A.2.2 Domain Event Pattern

```
public sealed class UserRemovedFromModuleDomainEvent : DomainEvent {
  public UserRemovedFromModuleDomainEvent(Guid userId, Guid moduleId) {
    UserId = userId;
    ModuleId = moduleId;
  }

  public Guid UserId { get; }
  public Guid ModuleId { get; }
}

public sealed class UserRemovedFromModuleDomainEventHandler :
  INotificationHandler<DomainEventNotification<UserRemovedFromModuleDomainEvent>>
{
  public UserRemovedFromModuleDomainEventHandler(
      IUserLabRepository userLabRepository) {

    UserLabRepository = userLabRepository;
  }

  private IUserLabRepository UserLabRepository { get; }

  public async Task Handle(DomainEventNotification<
      UserRemovedFromModuleDomainEvent> notification, CancellationToken
      cancellationToken) {

    var specification = new GetUserLabsWhereModuleSpecification(
      userId: notification.Event.UserId,
      moduleId: notification.Event.ModuleId);

    var userLabs = UserLabRepository.GetItems(specification: specification);

    _ = await UserLabRepository.DeleteRangeAsync(items: userLabs,
                                          cancellationToken: cancellationToken);
  }
}
```

Listing A.4: An implementation of a Domain Event Pattern.

A.3 Ardalis.Specification

```
public sealed class SearchForUsersInModuleButNotInLabSpecification
  : Specification<User>
{
  public SearchForUsersInModuleButNotInLabSpecification(
    Guid moduleId, Guid labId, string searchExpression)
  {
    Query.Include(x => x.UserModules);
    Query.Where(x => x.UserModules.Select(um => um.ModuleId).Contains(moduleId));

    Query.Include(x => x.UserLabs);
    Query.Where(x => !x.UserLabs.Select(x => x.LabId).Contains(labId));

    Query.Search(x => x.FirstName, "%" + searchExpression + "%");
    Query.Search(x => x.Surname, "%" + searchExpression + "%");
  }
}
```

Listing A.5: An implementation of a Specification Pattern.

A.4 AutoMapper

```
public class LabDetailModel : LabModel
{
  public IEnumerable<UserModel> Users { get; set; } = null!;
  public IEnumerable<LabScheduleModel> LabSchedules { get; set; } = null!;
}

public sealed class LabDetailModelMappingProfile : Profile
{
  public LabDetailModelMappingProfile()
  {
    CreateMap<Lab, LabDetailModel>()
      .IncludeBase<Lab, LabModel>()
      .ForMember(x => x.Users, m => m.MapFrom(s => s.UserLabs.Select(x => x.User))
          );
  }
}
```

Listing A.6: An implementation of AutoMapper.

A.5 FluentValidation

```
public sealed class Delete
{
  public sealed class Command : IRequest<Response>
  {
    public Command(Guid labId)
    {
      LabId = labId;
    }

    public Guid LabId { get; }
  }

  public sealed class CommandValidator : AbstractValidator<Command>
  {
    public CommandValidator()
    {
      RuleFor(x => x.LabId).NotEmpty();
    }
  }
}
```

Listing A.7: An implementation of FluentValidation.

A.6 MudBlazor

A.6.1 Error Dialog (Razor)

```
<MudDialog>
    <DialogContent>
        <MudText Align="Align.Center">@ContentText</MudText>
    </DialogContent>
    <DialogActions>
        <MudContainer Class="d-flex justify-center gap-4 mb-2">
            <MudButton OnClick="Close" Variant="Variant.Filled">
                @CloseButtonText
            </MudButton>
        </MudContainer>
    </DialogActions>
</MudDialog>
```

Listing A.8: Razor template for Error Dialog.

A.6.2 Error Dialog (C#)

```
public partial class ErrorDialog
{
  [CascadingParameter] public MudDialogInstance MudDialog { get; set; } = null!;

  [Parameter] public string ContentText { get; set; } = string.Empty;
  [Parameter] public string CloseButtonText { get; set; } = string.Empty;

  void Close() => MudDialog.Close();
}
```

Listing A.9: Error Dialog class.

Appendix B

Testing Libraries

B.1 FluentAssertions

```
public class TestsDeleteCommandValidator
{
  private Delete.CommandValidator Validator { get; set; } = new();

  [Test]
  public void Command_Valid()
  {
    var command = new Delete.Command(moduleId: Guid.NewGuid());

    Validator.Validate(command).IsValid.Should().BeTrue();
  }

  [Test]
  public void Command_Invalid()
  {
    var command = new Delete.Command(moduleId: Guid.Empty);

    var result = Validator.Validate(command);

    result.IsValid.Should().BeFalse();

    result.Errors.Count.Should().Be(1);
    result.Errors[0].ErrorMessage.Should().Be("'Module Id' must not be empty.");
  }
}
```

Listing B.1: An implementation of FluentAssertions.