T.C.
BAHÇEŞEHİR UNIVERSITY

FACULTY OF ENGINEERING AND NATURAL SCIENCES

CAPSTONE PROJECT PROPOSAL

AN OPTIMIZATION-BASED PROJECT ASSIGNMENT SYSTEM I

Emirhan Erol - Software Engineering
Ramazan Oğuzhan Kesgin - Software Engineering
Necati Berkan Şafak - Industrial Engineering
Onur Türeyen - Industrial Engineering
Yiğit Efe Yılmaz - Software Engineering

Advisors: Prof. Sabri Tankut Atan
Assist. Prof. Tamer Uçar

ISTANBUL, May 2024

STUDENT DECLARATION

By submitting this report, as partial fulfillment of the requirements of the Capstone course, the students promise on penalty of failure of the course that

- they have given credit to and declared (by citation), any work that is not their own (e.g. parts of the report that is copied/pasted from the Internet, design or construction performed by another person, etc.);
- they have not received unpermitted aid for the project design, construction, report or presentation;
- they have not falsely assigned credit for work to another student in the group, and not take credit for work done by another student in the group.

ABSTRACT

AN OPTIMIZATION-BASED PROJECT ASSIGNMENT SYSTEM I

Emirhan Erol - Software Engineering

Ramazan Oğuzhan Kesgin - Software Engineering

Necati Berkan Şafak - Industrial Engineering

Onur Türeyen - Industrial Engineering

Yiğit Efe Yılmaz - Software Engineering

Faculty of Engineering and Natural Sciences

Advisors: Prof. Sabri Tankut Atan

Assist. Prof. Tamer Uçar

May 2024

This project presents an Optimization-based Project Assignment System shaped for academic environments. Developed as a .NET Core MVC Web application, it integrates optimization algorithms coded with Python using the Gurobi optimizer to streamline the process of assigning projects to students. The system offers a secure login interface by using the ASP.NET Core Identity library, distinguishing between administrators and students, granting varying levels of access and control. Key functionalities of the product include project submission, distribution, tracking, and progress monitoring, providing a streamlined workflow for project assignment procedure. Overall, the project involves a combination of technologies and practices in order to deliver an efficient and scalable solution for academic project management.

Administrators wield authority to manage users, projects, teams, departments, and optimization rulesets and carry out administrative operations. Students, on the other hand, engage in team formation within their departments, either by joining existing teams or establishing new ones. Upon team formation, students can make their project choices through a dedicated interface, utilizing dropdown menus for selection.

The system's core functionality lies in its integration with the Gurobi optimization engine. Upon project selection, this engine dynamically accesses appropriate data from the hosted database, executes optimization algorithms, and sends optimized results back into the

system. Post-optimization, both students and administrators can display final result summaries through a dedicated results page including tables and graphs.

Several different methods of testing are applied to the product during and after the development stage of this project. To assess the system's performance, test runs were conducted using randomly generated datasets. The parameters and the data was changed according to the results of the tests.

In conclusion, the Optimization-Based Project Assignment System enhances academic project management by integrating a .NET Core MVC Web application with Python-coded optimization algorithms using the Gurobi optimizer. The system ensures secure, role-based access via ASP.NET Core Identity, offering distinct functionalities for administrators and students. Administrators manage users, projects, teams, departments, and optimization rulesets, while students form teams and select projects through an interface. The Gurobi engine processes data for optimized project assignments, and extensive testing with varied datasets confirms the system's reliability. This solution effectively streamlines project submission, distribution, tracking and providing an efficient and scalable tool for academic environments.

KEY WORDS

Gurobi, .NET, Python, C#, MSSQL,

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

BL	Business Layer
CRUD	Create Read Update Delete
DAL	Data Access Layer
DB	Database
DBMS	Database Management System
EF	Entity Framework
EL	Entity Layer
FIFO	First in First Out
LP	Linear Programming
MVC	Model-View-Controller
SPA	Student-Project Allocation
UI	User Interface
WBS	Work Breakdown Structure

1. OVERVIEW

Optimization-Based Project Assignment System aims to streamline the process of assigning projects to students within an academic environment. By integrating optimization algorithms, the system optimally matches student preferences with projects, ensuring an equitable and efficient distribution of tasks. The solution includes a user-friendly interface that allows students to express their project preferences, while administrators can oversee and manage the assignment process. The system's architecture combines a database design and a web-based user interface. This solution enhances the overall project assignment experience, promoting fairness, transparency, and flexibility during the assignment of projects to students based on their preferences and the project's specific needs.

The project group includes 5 members in which 3 are Software Engineers and the other 2 are Industrial Engineers. As decided between the group members, while the website backend and the database architecture will be done by Software Engineering Students, the optimization algorithm will be prepared by Industrial Engineering students. The common tasks that will be done by both departments' students are user interface design and testing.

1.1. Identification of the need

The project assignment optimization system addresses the challenges connected to manual or less efficient methods of assigning projects to students in academic environments. Traditional assignment processes often lack fairness, transparency, and optimization, leading to mismatches between student preferences and project requirements. This can result in suboptimal project outcomes and decreased student satisfaction. It is believed that students are the primary beneficiaries of the system. They can input their project preferences, aligning assignments with their interests. This enhances their overall learning experience and engagement with projects. Also, academic instructors benefit from streamlined processes and reduced workload. They gain insights into the assignment optimization, allowing for better resource allocation and overall program management.

1.2. Definition of the problem

The system is required to perform better than the current system that the school is using.
The manual and time-consuming process of assigning capstone projects to student teams. Currently, students select their projects based on preferences, leading to potential inefficiencies and suboptimal outcomes. The aim is to automate and optimize the assignment process using an integer programming model solved by the Gurobi optimization solver. By implementing this solution, the project seeks to enhance the efficiency of capstone project allocation, ensuring that students are assigned to projects aligning with their preferences while adhering to various constraints.

1.2.1. Functional requirements

- Different user roles (students, instructors, and administrators) shall have appropriate access levels.
- Students shall be able to create teams with other students from their departments.
- The optimization algorithm shall distribute assignments to students with the best possible satisfaction of students.
- Students and instructors shall be allowed to view which projects they are assigned to.
- Students shall select their project choices based on their preferences.

1.2.2. Performance requirements

The basic equation that we currently have to define the performance of the product is this:

$$\text{Max} \sum_{i=1}^{S} \sum_{j=1}^{P} Lij * Xij$$

where S is the set of students and L is the set of projects.

L is the preference, 1st choice is 5, 2nd choice is 4 and so on. If the project is not in the list the value will just be 0.

Xij is a binary value and is 1 if student “i” is assigned to the project j and 0 if he is not.

This is very similar to the equation in Harper et al [1]’s article, as the basic form of an LP problem is usually similar among similar systems.

Teacher’s student preference and more parameters are likely to be added in the performance equation in the future.

The performance can also be evaluated by the ease of usage and practicality of the user interface which can be rated by the users.

1.2.3. Constraints

Scheduling: Since the development stage of this project will be done during the second term of an educational year, most of the team members will have tight schedules. Also, the programming languages which the team members plan to use are all new to them, so there will be a learning curve during the development stage.

Cost: Most of the time, budget will not be a deciding factor in this project. Most of the development tools that will be used during the development stage are chosen from the available free options on the Internet. One programming library, Gurobi, is originally only a paid service. However, Bahçeşehir University provides each and every team member with a student license of their own. Students are required to bring their own personal computers to the university building in order to get technical support.

User Skill Levels: The end product of this project will be designed for the usage of university students and instructors. Even though we, as software and industrial engineers, would easily figure out how this web-based user interface will work, this project and its user interface must be designed as user-friendly and easy-to-learn as possible.

Technology: It is decided by the team members with the help of project advisors that an online website with a user-friendly interface would be the best option for this project. However, a website interface would limit the available programming languages that can be used because the programming language of the website should both have the qualities for easy user interface design on the frontend and database manipulation on the backend side. Also, the team has to find an online database hosting service, possibly free, in order to keep the website online as much as possible.

Social Impact: It is assumed that students might resist the change on the project selection method since from the sociological point of view, people are resistant to changes in their lives. There will also be a learning process of how to use the website regarding making choices, checking results, and navigating around the website. However, it is believed by the team members that the overall satisfaction of users, both students and instructors, will show a potential increase due to the optimized automation of the process.

1.3. Conceptual Solutions

In this section, academic research concerning the optimization engine's algorithm is included. In addition to that, reasons for selecting which programming languages are chosen is explained with comparisons between possible other tools.

1.3.1. Literature Review

Harper et al. [1] have done a similar project using genetic algorithm instead of linear programing and found that it procures better results compared to a LP method, despite these findings we are more likely to still use the LP method as a genetic algorithm method is outside of our scope and may pose higher risks to our project yet if we have extra time this approach may be considered as an extra objective

Manlove and O'Malley [2] used a slightly more complex model that includes teacher preferences for both projects and students, in our school the teachers are connected to the projects to begin with so we are not required to assign them to a specific project. At the beginning of our project adding the preference of teachers over students wasn't in our plan, but as we looked through the literature, we have noticed the majority of articles using this system, as a result we are considering adding this to our system. If not, we are considering comparing a system without student preferences to a one with and evaluating the results.

There are more articles (Abraham et al [3], Manlove and Copper [4], which have tried differing methods to solve the SPA problem, although their methods and constraints can be used as an inspiration for us, the solution models are not useful as a basic LP model is most likely sufficient, although if we find unsatisfactory results with this method, we could give more attention to the mentioned models and try a different method.

1.3.2. Concepts

First, we have to choose the problem-solving method to use in our system, the first and foremost candidate as we mentioned is the LP model, The second method is the genetic algorithm used by Harper et al [1] and the 3rd is multiple methods combined into 1 row since all of them share similar features and can be interchanged if they are chosen. The 4th model is using a modified basic LP model which best suits our needs and systems. I have not considered the "cost" of these methods as they have no difference in cost and are all practically free, the only cost that can be considered is time cost and that depends on the complexity of the method. Features are the same among the candidates as the features of the system do not depend on the solving methods.

Basic LP has the benefit of being “good enough” and easy to do. The problem is that the solution that the system gives could be insufficient and as a result, less students than desired would get their preferred projects.

Genetic Algorithm is proven to be effective and according to Harper et al [1], user friendly and fast, this method is certainly a great option but has the problem of being too complex and requiring multiple iterations. Since our group has no prior experience in coding genetic algorithms, making a model based around genetic algorithms seems to be very time consuming and risky.

The other complex methods such as 3/2 approximation algorithm by Cooper and Manlove [4] and Student oriented SPA model and Abraham et al [3] have similar upsides and downsides to the genetic algorithm, they usually perform better than the basic LP model while being much more complex, if chosen we would have to experiment with many different methods and choose the best performing one.

Modified LP method is a great choice for us that we are currently leaning towards. The idea is that we could start with a basic LP model and modify the solution process as we need and try different methods and tactics from the other methods and incorporate the ones that synergise well with our system. This increases the complexity by enough that the performance will be satisfactory but doesn’t increase it so much that the deadline may become an issue.

Table 1. Comparison of the four conceptual solutions.

	Basic LP	Genetic Algorithm	Other Complex Methods	Modified LP Method
Complexity	Low	High	Medium/High	Medium
Performance	Medium	High	Medium/High	Medium
Features	Medium	Medium	Medium	Medium

The next concept to be chosen is the coding language that will be used for the optimization code, the following Table 2 illustrates the advantages and disadvantages of the three mentioned languages.

Table 2. Comparison of the three languages.

	Python	C#	C++
Complexity	Low	High	High
Performance	Medium	Medium	High

Features	Medium	High	High

Among three options shown in Table 2. Python is the most obvious choice for us as it is the language that the industrial engineers who will code the optimization engine are the most comfortable with. It is also compatible with .NET which will be used in the web-based part of the system. The only downside is that python is inherently slower than the other languages in execution but for a project this size, the difference will be negligible.

C# has the same features in integration with .NET and also has the added advantage of being the language that our software engineers are the most comfortable with, the biggest issue of this language is that it is not taught to the industrial engineers and coding the optimization engine in a foreign language will be extremely time consuming.

C++ is very similar to C# but one of our industrial engineers has prior knowledge in C++. As a result of this it is possible for us to use it, as the speed of this language is unmatched by the others but the speed difference is most likely overshadowed by how simple and less time-consuming python is.

The need of storing data naturally requires the creation of a database. There are countless DBMS on the Internet in one's reach. However, as always when a comparison is made, some things are better than the others depending on the usage. This is also no different in this case. It is agreed upon that three different DBMS are to be compared with each other in varying categories to find out which one is more suitable than the others for this project. As it can be seen in Table 3, the comparison is done considering DBMS' cost, complexity, performance, features, and community support.

Table 3. Comparison of the three conceptual solutions

	SQL Server Express 2022	Firebase	MySQL Community Edition
Cost	Low	Low	Low
Complexity	Medium	Low	Medium
Performance	High	Medium	Medium/High
Features	High	Medium	Medium/High
Community Support	High	High	High

Comparison of the three conceptual solutions shown in Table 3. SQL Server Express 2022 stands out with a high feature set, high performance, and moderate complexity. It excels in handling complex

queries and supporting transactions, making it ideal for academic projects with structured data requirements. The community support is robust, ensuring reliable assistance and resources. Despite its high performance, the cost remains relatively low, aligning seamlessly with the no-cost environment of the project.

Firebase, characterized by its low complexity, is easy to set up and maintain. While offering moderate performance and features, its real-time database capabilities make it suitable for collaborative projects. The active Firebase community provides strong support, aiding developers in resolving issues. With a cost-effective structure, Firebase aligns well with the no-cost nature of the project, making it an accessible option for specific scenarios.

MySQL Community Edition strikes a balance between complexity and performance, offering a moderate to high level of both. Its versatile feature set supports various project requirements, and the strong community support ensures resources and assistance are readily available. Cost-effective, MySQL Community Edition aligns seamlessly with the budget constraints of the project. Its compatibility with different server environments makes it a flexible choice for academic projects, offering a comprehensive solution for structured data management.

SQL Server Express 2022 was chosen for the project's complexity and the need for a feature-rich relational database, SQL Server Express is a suitable choice. It aligns with academic and industry standards for robust data management. And since we had previous experience of using it, we chose it because we would be more familiar with its use for the project.

The application type of this project has to be decided before the development stage of the product. What type of a product this project will produce at the end of the project life cycle is a critical factor that will decide the usability of this end product. Several ideas were argued within the team members and they are compared between each other considering the categories in Table 4.

Table 4. Comparison of the three platforms.

	Web Application	Mobile Application	Console Application
Cost	Low	Low	Low
Complexity	Medium	Medium	Low

Performance	High	Medium	High
Features	High	Medium/High	Low/Medium
Community Support	High	Medium	Medium

Web applications offer a balanced solution with a moderate level of complexity, ensuring accessibility through internet browsers on various devices. They excel in performance, providing high responsiveness and real-time updates. Feature-rich, web applications support interactive user interfaces, making them well-suited for the capstone project's academic context. Active community support ensures robust development and troubleshooting, all while maintaining cost-effectiveness within the project's no-cost environment.

Console applications, characterized by simplicity and low complexity, offer high performance, especially for specific tasks. However, their features are limited compared to web applications. Ideal for scripted tasks and automation, console applications may lack the visual elements required for effective student interaction in the capstone project. With moderate community support, their low complexity makes them a cost-effective option within the no-cost project environment.

Mobile applications provide portability and accessibility with a moderate level of complexity, catering to users preferring on-the-go access through smartphones or tablets. While offering medium performance and a range of capabilities, mobile apps may introduce complexities in development and maintenance. The features, community support, and cost are generally balanced, making mobile applications viable for specific scenarios.

Given the academic nature of the capstone project, a web application remains the optimal choice, ensuring broad accessibility and a user-friendly interface. One of the most important factors regarding why the web application type is selected is that students in the Software Engineering department of this project have already worked with either one or more of the website programming languages which are HTML, CSS, C#, and ASP.NET Core.

1.4. Architecture

As it can be seen in Figure 1, the system consists of three separate sides. The database will be designed by the team and put to an online database hosting service which will be an external service. There will also be a website side of the system which will be what the user only sees. The optimization engine must be able to interact with the database to be able to receive and send data to the database. As this system will not be used just by software programmers, an easy-to-use website interface is needed. Users will only interact with the website UI while the website backend will take inputs from users and make the necessary information exchanges with the database.

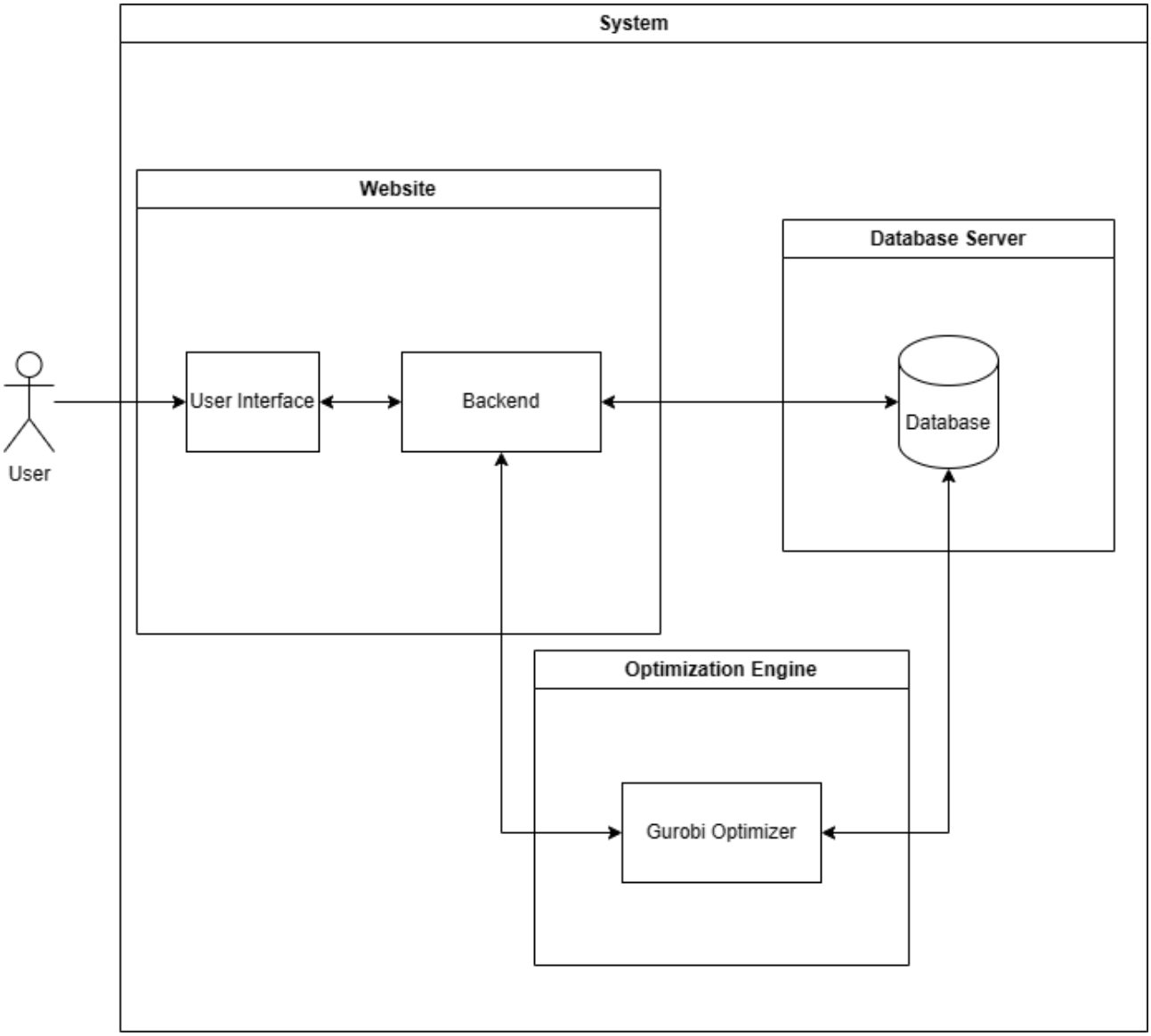


Figure 1 System Architecture

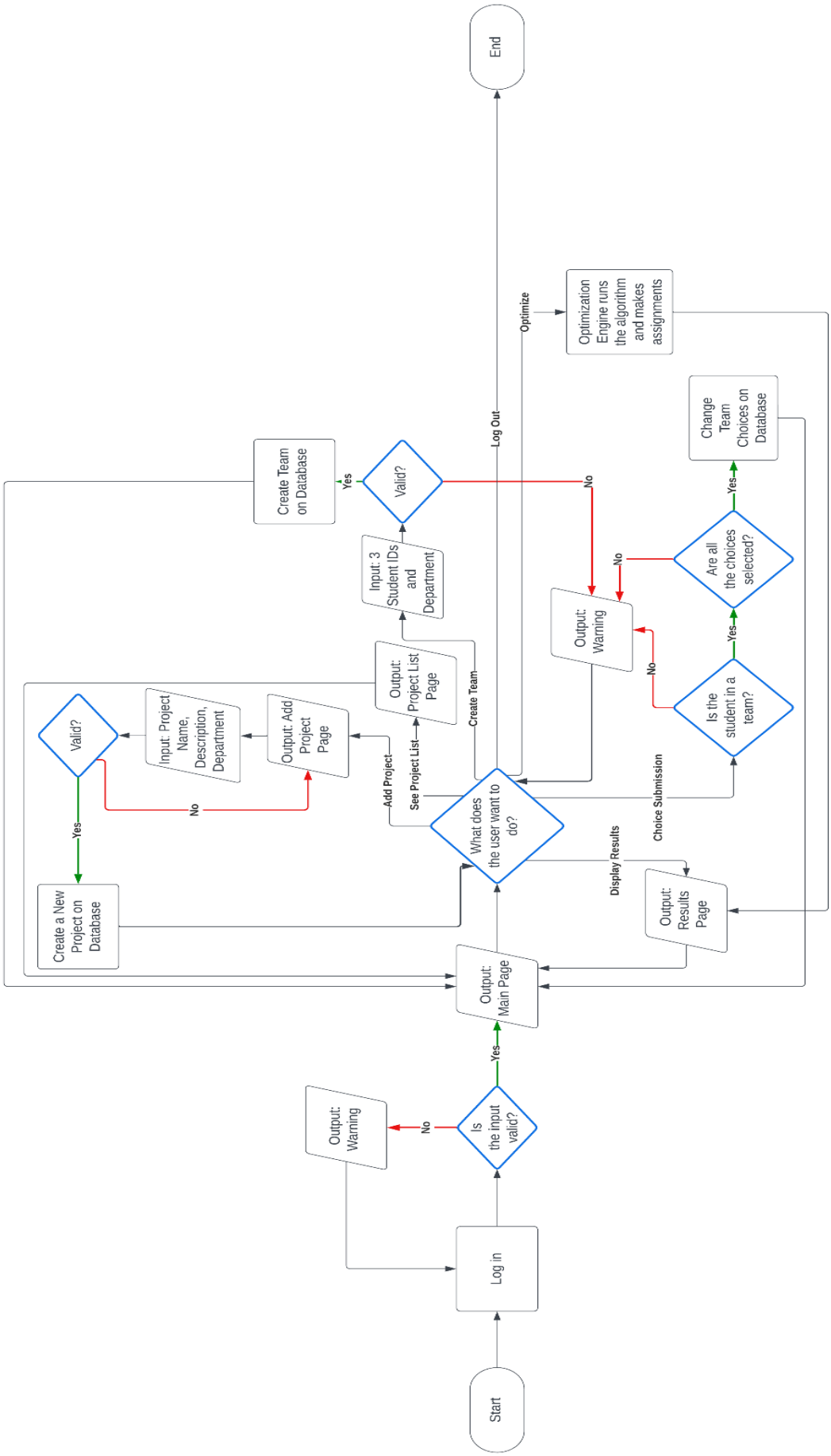


Figure 2 Process Chart

2. WORK PLAN

In this section of the report, the Project Assignment System is broken down to its smallest components and it is decided who will do them at when. Most of the work is divided by taking student departments into consideration. The development and testing stages of this project is planned to be held between the first and the fifteenth weeks of the second semester of 2023-2024 educational year.

2.1. Work Breakdown Structure (WBS)

The Project Assignment System can be divided into two categories. For this project, it is not necessary to start dividing the work with software and hardware since the only hardware requirement for this project is to acquire servers in order to host the database online. Since the online hosting process will be done by providing the service externally, it is not relevant to add the hardware part. The two categories are, which are implicated by orange boxes in the figure below, the Web-Based Application and the Optimization Engine. Certain tasks and work packages can also be found in Figure 3. below.

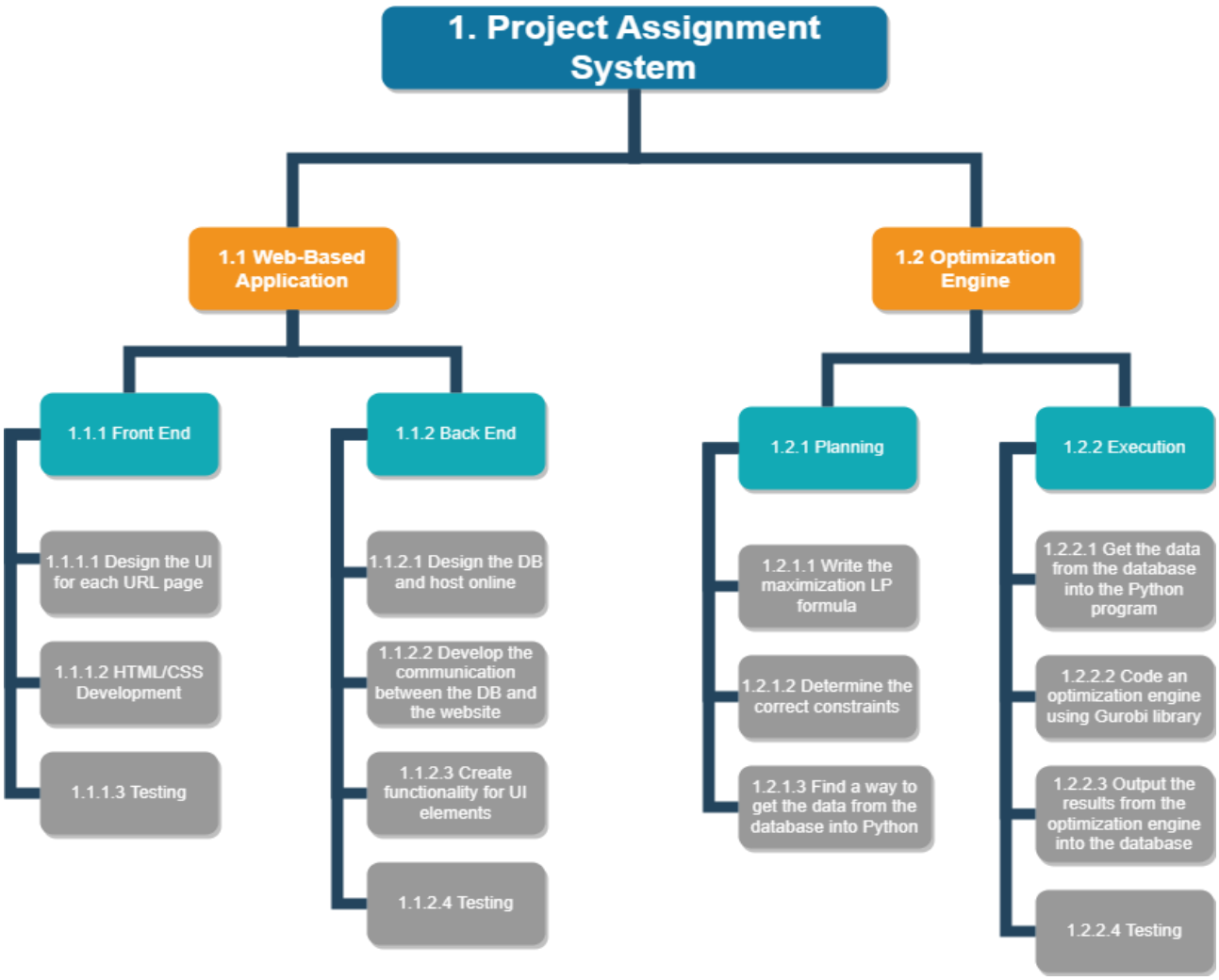


Figure 3 Work Breakdown Structure

2.2. Responsibility Matrix (RM)

Considering the WBS presented in Figure 3, major tasks are divided between the team members fitting their capabilities. Software Engineering students, Emirhan, Yiğit and Oğuzhan, are responsible for most of the front end and back end of the web site application while Industrial Engineering students, Berkan and Onur, are responsible for optimization engine planning and execution. However, team members will help and support their colleagues from their respective departments. Responsibilities are presented in more detail in Table 5. below.

Table 5. Responsibility Matrix Table.

Task	Emirhan	Yiğit	Oğuzhan	Berkan	Onur
Front End	S	**R**	S	S	
Back End	S	S	**R**		
Optimization Engine Planning				**R**	S
Execution				S	**R**
Planning	**R**	S	S	S	
Integration	S	S	**R**	S	
Reporting	**R**	S	S	S	S

R = Responsible S = Support

2.3. Project Network (PN)

The order parts of the development, design and mathematical calculations of the optimization engine and web application sections are carried out. Afterwards, the necessary design and development parts are tested after their functionality is created, and integration is provided between the web application and the optimization engine. While these operations are being carried out, optimization engine code is made using the Gurobi library and optimization, performance testing, verification and validation are performed together. Finally, after the report is drafted, it is concluded with the final report. Details can be found about the PN in Figure 4. below.

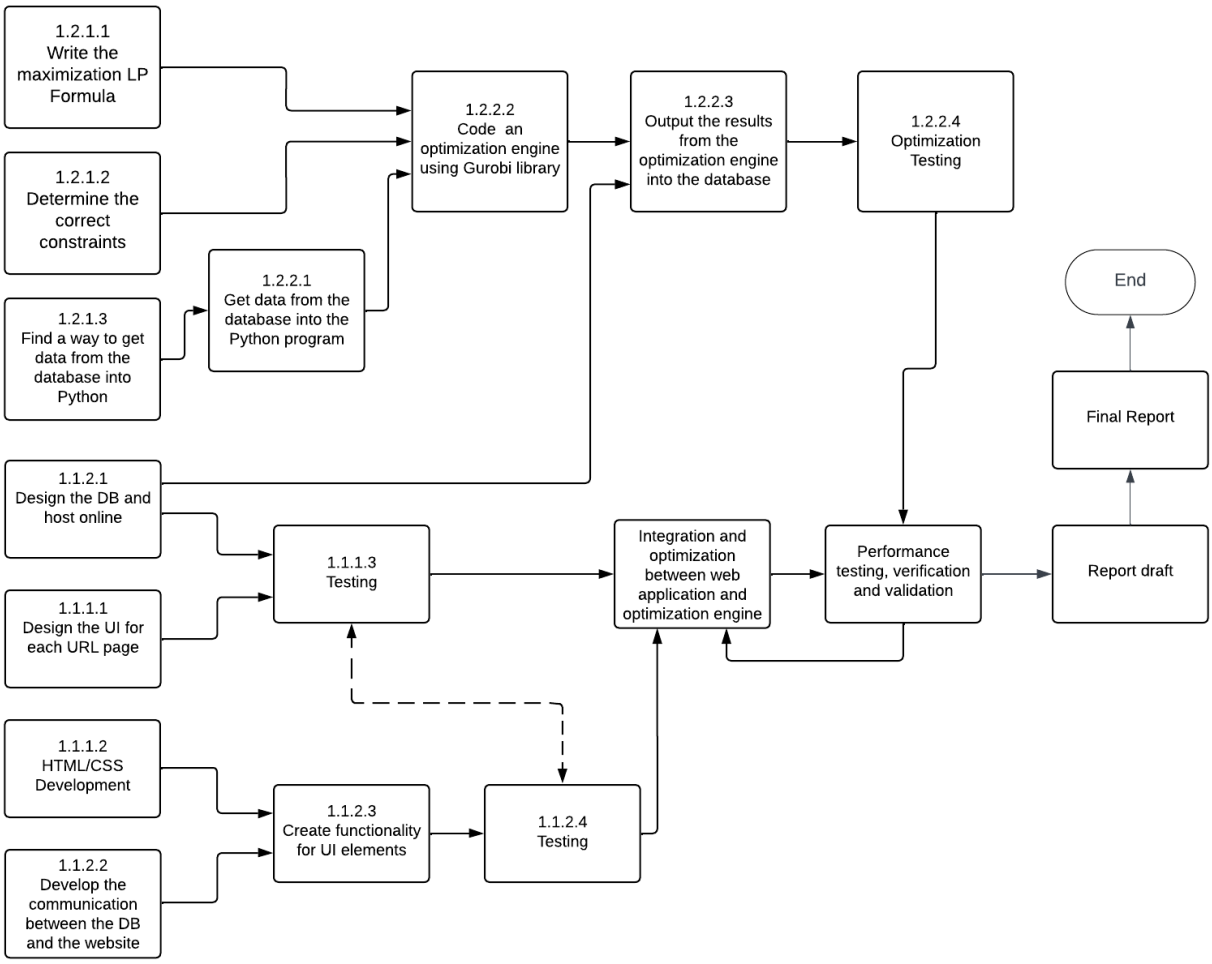


Figure 4 The Project Network

2.4. Gantt chart

Writing the LP formula and Designing the DB and hosting it will be done simultaneously in week 1 by different groups. In the second week, IE students will start and finish determining the constraints and finding a theoretical way to input data from the DB into python. At the same time, the SE students will be designing the UI for each URL page. In the 3rd week, SE students will start their front-end testing and HTML/CSS development which will continue until the end of week 6. The next major milestone is the coding of the optimization engine which will start in week 5 and continue until the end of week 7. At the same time SE students will develop communication between DB and the website, create functionality for UI elements and Back End Testing. When both of these tasks are completed, both teams will link up and start integrating the subsystems until the end of week 12. week 12 is planned to be the week where the project is finished. After week 12, there will only be performance testing, verification and validation, optimization testing which will need to be done. Furthermore, the project report will be finalized in the last 2 weeks. More detailed information on the working plan can be found in the Gantt Chart displayed in Figure 5.

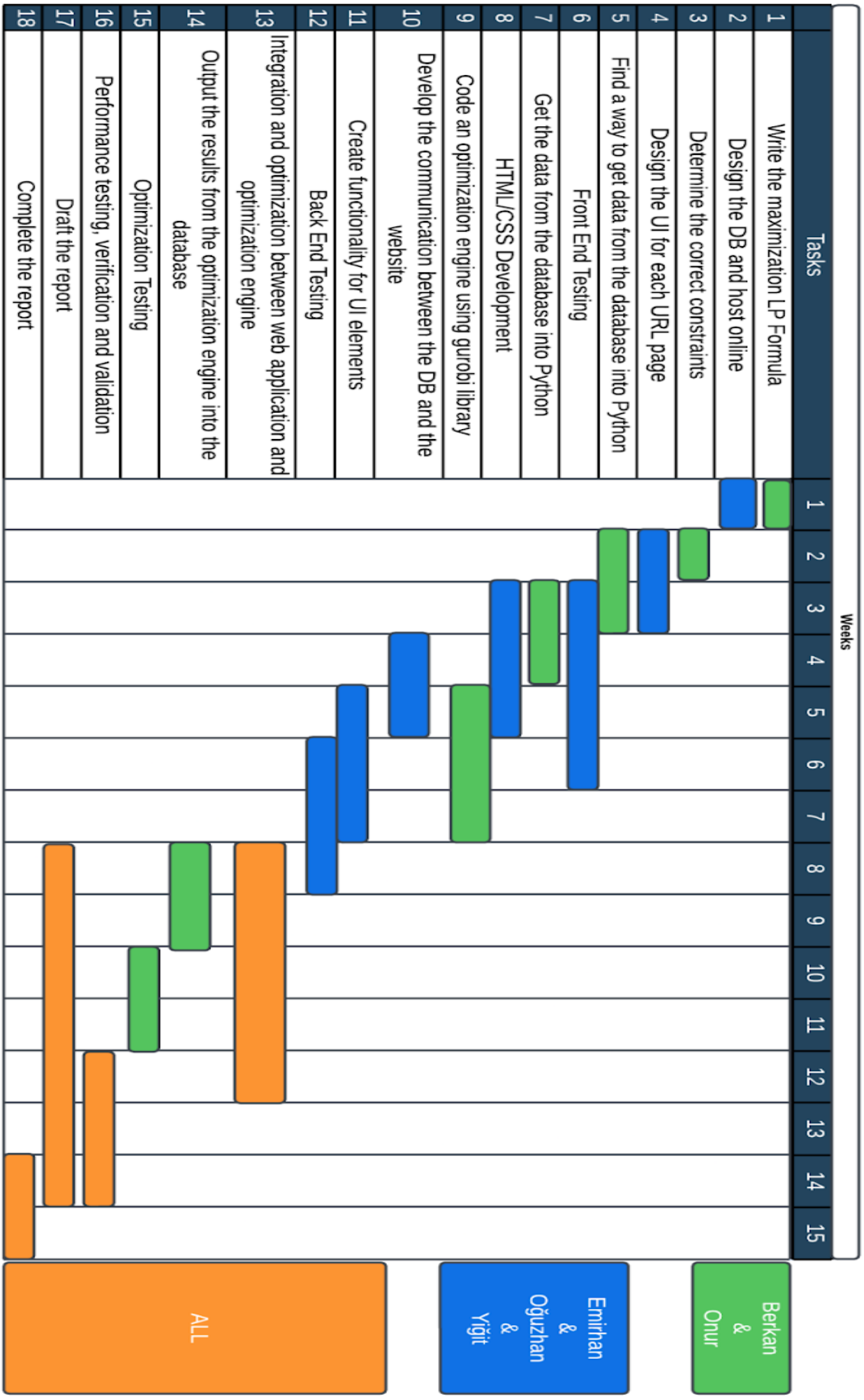


Figure 5 Gantt Chart

2.5. Costs

Optimization Based Project Assignment System has no physical parts in the team's responsibility. Therefore, many reasons to cause a cost on the team's budget is eliminated drastically. The only thing that could have created a cost is online database hosting. However, the online database server is to be provided for free and externally. Another thing that would have cost money if this project's aim was to earn money is development tools. All of the tools that will be used in this project are free for academic usage. Regarding the Gurobi licensing, it is possible to get a free Gurobi license by connecting to the university network with a laptop or contacting Gurobi support and proving that each member is a student at the Bahçeşehir university in order to get a free Gurobi license on a desktop computer.

2.6. Risk assessment

Table 6. Risk Matrix

RISK LEVEL		Severity of the event on the project success: Minor	Moderate	Major
Probability of the event occuring	Unlikely	VERY LOW	LOW	MEDIUM
	Possible	LOW	MEDIUM	HIGH
	Likely	MEDIUM	HIGH	VERY HIGH

VERY LOW	This event is very low risk and so does not require any plan for mitigation. In the unlikely event that it does occur there will be only a minor effect on the project.
LOW	This event is low-risk; a preliminary study on a plan of action to recover from the event can be performed and noted.
MEDIUM	This event presents a signficant risk; a plan of action to recover from it should be made and resources sourced in advance.
HIGH	This event presents a very signficant risk. Consider changing the product design/project plan to reduce the risk; else a plan of action for recovery should be made and resources sourced in advance.
VERY HIGH	**This is an unacceptable risk. The product design/project plan must be changed to reduce the risk to an acceptable level.**

Table 7. Risk Assessment Table

Failure Event	Probability	Severity	Risk Level	Plan of Action
Database server shut-down	Unlikely	Major	MEDIUM	Since the database server is provided as an external service, the server provider will be contacted immediately. There is not much to do about this risk other than being informed when the server will be back again.
Loss of connection between the website and the database	Possible	Major	HIGH	The root of the problem must be identified if it is website or database sided. System is shut-down and the cause of the problem is fixed.
Dysfunctions in the website back-end and front-end integration	Unlikely	Moderate	LOW	While this problem occurs, the website can be kept online. The problematic UI elements or back-end code is reprogrammed and implemented to the website with a server restart.
Missing Constraints in the optimization engine leading to a wrong solution	Unlikely	Major	MEDIUM	If this mistake is not caught before the delivery date, it could result in a non-functional system. Fortunately, we will have many tests that can potentially catch this.
Delay in a deliverable from a subsystem	Likely	Minor	MEDIUM	We have enough leeway in our Gant chart to accommodate a delay and still finish the project in time.

Failure Event	Probability	Severity	Risk Level	Plan of Action
An unsatisfactory optimization result	Unlikely	Major	MEDIUM	If this happens, we would have to use a different optimization method which would require a significant amount of time.

3. SUB-SYSTEMS

3.1. Software Engineering

The Software Engineering subsystem is made of 3 members which are Emirhan Erol, Yiğit Efe Yılmaz, and Oğuzhan Kesgin. Software Engineering members will be mostly responsible for the UI, the website backend, and the DBMS. This subsystem will store data, take user inputs, and provide the required UI to users in order for them to manipulate the database.

3.1.1. Requirements

Behaviors of the Software Application

Table 8. List of Behaviors

Actor Name	Name of Behavior	Description of Behavior
Student	1. logIn() 2. logOut() 3. submitChoices() 4. checkProject() 5. seeProjectList() 6. createTeam()	1. Logs in the Student for further steps 2. Logs out the student from the system 3. Pushes student choices to the database 4. Returns assigned project, related student IDs, and related Instructor IDs 5. Returns all projects on a student's display 6. Creates a Team in the Database with entered inputs
Instructor	1. logIn() 2. logOut() 3. checkproject() 4. seeProjectList()	1. Logs in the Instructor for further steps 2. Logs out the

	5. addProject() 6. removeStudent() 7. addStudent()	instructor from the system 3. Returns projects, related student IDs and related Instructor IDs 4. Returns all projects on an Instructor's display 5. Creates a new project and pushes it to database 6. Manually removes a student from a project in the database 7. Manually adds a student to a project in the database
Administrator	1. logIn() 2. logOut() 3. seeProjectList() 4. removeProject() 5. addProject() 6. removeStudent() 7. addStudent() 8. removeInstructor() 9. addInstructor() 10. deleteStudent() 11. createStudent()	1. Logs in the Administrator with maximum authorization 2. Logs out the Administrator from the system 3. Returns all projects on an Administrator's display 4. Removes a project from the database 5. Adds a new project with the given input values 6. Manually removes a student from a project in the database 7. Adds a student to a project 8. Removes an instructor from the database 9. Adds a new

		Instructor with the given input values 10. Removes a student from the database 11. Adds a new Student with the given input values

Attributes of the Software Application

Table 9. List of Attributes

Actor Name	Name of Attribute	Description of Attribute
Student	1. StudentID 2. StudentName 3. DepartmentID 4. AssignedProjectID 5. Choice1 6. Choice2 7. Choice3	1. The unique ID of a student 2. The name of a student 3. The ID of the department that a student belongs to 4. The unique ID of a project which the student is assigned to 5. The first choice of a student 6. The second choice of a student 7. The third choice of a student
Instructor	1. InstructorID 2. InstructorName 3. DepartmentID	1. The unique ID of an Instructor 2. The name of an instructor 3. The ID of the department that an instructor belongs to
Project	1. ProjectID 2. ProjectName 3. InstructorID	1. The unique ID of a project 2. The name of a project 3. The ID of the Instructor who is responsible for the project.
Administrator	1. AdministratorID 2. AdministratorName	1. The unique ID of an Administrator 2. The name of an Administrator

Department	1. DepartmentID 2. DepartmentName	1. The unique ID of a department 2. The name of a department

Performance Requirements

Response Time: The system should provide project assignment results within 15 seconds of requesting results.

Optimization Engine Execution Time: The optimization engine should complete the assignment optimization for all students in less than 5 minutes.

Safety Requirements

Data Integrity: The system should implement robust data validation mechanisms to ensure the integrity of user-submitted data and prevent data corruption.

Error Handling: The system should gracefully handle unexpected errors and provide informative messages to users. It should not crash or lose data due to errors.

Security Requirements

Data Confidentiality: Access to student, project, advisor, and department data should be restricted to authenticated users with appropriate roles. Data confidentiality must be maintained.

Business Rules

Project Submission Rules: Each student must submit exactly three project preferences, prioritized as 1st, 2nd, and 3rd choice.

Conflict Resolution: In case of conflicting project preferences among students, the system should prioritize based on academic performance or other predefined criteria.

Departmental Constraints: The system should ensure that students are assigned projects related to their departments.

3.1.2. Technologies and methods

The .NET Core Web application project embraces a robust and contemporary tech stack to deliver seamless functionality and maintainability. Developed using Visual Studio 2019 with .NET 5.0 as the target framework, the backend is powered by the versatile C# language. The architecture follows the Model-View-Controller (MVC) layered approach, promoting a clean and scalable codebase. Adhering to the SOLID principles, project code is designed to be flexible, maintainable, and easily extensible. For data management, the Code First methodology was employed, enabling developers to define the data model in code, with Entity Framework Core serving as developers' object-database mapper. The project relies on essential NuGet packages, including Microsoft.EntityFrameworkCore, Microsoft.EntityFrameworkCore.Design, Microsoft.EntityFrameworkCore.SqlServer, and Microsoft.EntityFrameworkCore.Tools, all at version 6.0.20. These packages facilitate seamless integration with SQL Server and provide the necessary tools for efficient database operations.

The application employs a comprehensive approach to database connectivity. Connecting to the database is facilitated through the creation of a database context using Entity Framework Core. This context acts as a bridge between the application and the underlying database, enabling seamless data operations. Code-first methodology was followed, where the database context is generated based on model classes. These model classes represent the structure of the database tables and are created with precision to encapsulate data entities and relationships. The database connection details are stored securely by adding a connection string to the appsettings.json file, ensuring a clean separation of configuration from code. This configuration-centric approach not only enhances security but also facilitates easy maintenance and deployment. Leveraging this connection strategy, the application seamlessly interacts with the underlying database, allowing for efficient data retrieval and manipulation.

The web application communicates with the SQL Server database using ASP.NET, and administrators can manage the database using SQL Server Management Studio. The entire system operates within the Windows operating system environment. This structured communication and integration ensure the effective management and utilization of data.

.NET Application on the ASP.NET Core, Microsoft SQL Server on SQL Server 2022 Express Edition as a version was used in the project. A set of libraries named ASP.NET for data access and manipulation also provides a service that facilitates communication between .NET and SQL Server and acts as a data provider, enabling .NET applications to interact with SQL Server databases. A tool named SQL Server Management Studio for managing and interacting with SQL Server databases and also provides a service that gives a graphical interface for database management and allows administrators to visually manage and interact with the SQL Server database. Operating system is Windows and this provides the environment where both the .NET application and SQL Server

operate.

Connections and Communications:

1. **.NET Application to SQL Server:**
    - **Communication Method:** ASP.NET
    - **Purpose:** The .NET application sends queries and commands to the SQL Server to read, write, update or delete data.
2. **Data Items and Messages:**
    - **Read Data:**

      .NET Application requests student, project or instructor information from the SQL Server.

      SQL Server responds with the requested data.
    - **Write Data:**

      .NET Application sends requests to add new students, projects or instructors.

      SQL Server processes these requests and updates the database.
    - **Update Data:**

      .NET application requests modifications to existing records.

      SQL Server updates the data based on the provided instructions.
    - **Delete Data:**

      .NET application instructs SQL Server to remove unnecessary records

      SQL Server carries out the deletion operation.

Nature of Communications:

- **Structured Queries and Commands:**

  .NET application sends well-structured requests queries and commands) to the SQL Server.

  SQL Server processes these requests and responds accordingly.

  Ensures a clear and organized flow of information between the application and the database.

Database Management System

DBMS Tool:

Name: Microsoft SQL Server

Vendor: Microsoft Corporation

Version: SQL Server 2022 Express Edition

This SQL Server is a relational database management system developed by Microsoft. The Express Edition is a lightweight, free version suitable for small-scale applications.

Connectivity with Software (Entity Framework Core):

Purpose: Entity Framework Core (EF Core) is an Object-Relational Mapping (ORM) framework that simplifies database interaction by enabling developers to work with databases using strongly-typed .NET objects.

Integration: In your .NET Core web application, you can use Entity Framework Core to model and interact with your SQL Server database. EF Core uses a Code-First approach, where you define your data model in C# classes, and EF Core takes care of creating the database schema.

Connection String Configuration:

Purpose: The connection string contains the information needed to connect to the SQL Server database.

Integration: In your .NET Core application, you will have a configuration file where you specify the connection string. This file is read during application startup, and the connection string is used to establish a connection to the SQL Server database.

DbContext Class:

Purpose: The DbContext class in Entity Framework Core represents a session with the database and allows you to query and save instances of your entities.

Integration: A custom DbContext class is created that inherits from DbContext provided by EF Core. This class will include DbSet properties for each entity representing database tables.

Migration and Database Update:

Purpose: EF Core Migrations allow you to evolve the database schema as your application evolves.

Integration: After defining your entities, you can use EF Core Migrations to create and update the database schema based on your model changes.

Dependency Injection:

Purpose: Dependency injection is used to provide instances of your DbContext throughout your application.

Integration: In the Startup.cs file, you will configure dependency injection to ensure that instances of your DbContext are available where needed.

Microsoft SQL Server is a relational database management system (RDBMS) developed by Microsoft. The 2022 Express edition is a free, lightweight version designed for smaller projects due to its cost-effectiveness and resource efficiency. Vendor is Microsoft Corporation. SQL Server is chosen for its robustness, reliability and comprehensive features, making it suitable for managing complex data structures.

Steps to Achieve Connectivity [5]:

1. **Install Required Packages:**
    a. Open Visual Studio.
    b. Navigate to "Tools" > "NuGet Package Manager" > "Manage Packages for Solution."
    c. Search for and install the following NuGet packages:
    -Microsoft.EntityFrameworkCore
    -Microsoft.EntityFrameworkCore.SqlServer
    -Microsoft.EntityFrameworkCore.Tools
2. **Establish the Database Context:**
    a. Create a directory named "Data" in the root directory of the project.
    b. Within the "Data" directory, create a new C# file titled DBContext.cs.
    c. Add code to the DBContext.cs file to define a class named DBContext that inherits from DbContext.
3. **Define a Model for the Database Table:**
    a. Establish a directory named "Models" in the project's root.
    b. Inside the "Models" directory, create a C# file named Entity.cs.
    c. Add code to the Entity.cs file to define a class named Entity representing the database table.
4. **Incorporate a Connection String into appsettings.json:**
    a. Open the appsettings.json file in the root directory of your project.
    b. Add a JSON entry for "ConnectionStrings" with the desired connection string.

5. **Integrate Builder Service in the Program Class:**
    a. Access the Program.cs file within your project.
    b. Locate the section where services are configured.
    c. Add a configuration to include the database context in the dependency injection container.
6. **Execute the Initial Migration:**
    a. Open the Package Manager Console via "View" > "Other Windows" > "Package Manager Console."
    b. Execute a command to generate an initial migration for the defined models.
7. **Update the Database:**
    a. Execute a command in the Package Manager Console to apply the pending migration and update the database schema.

Database Schema

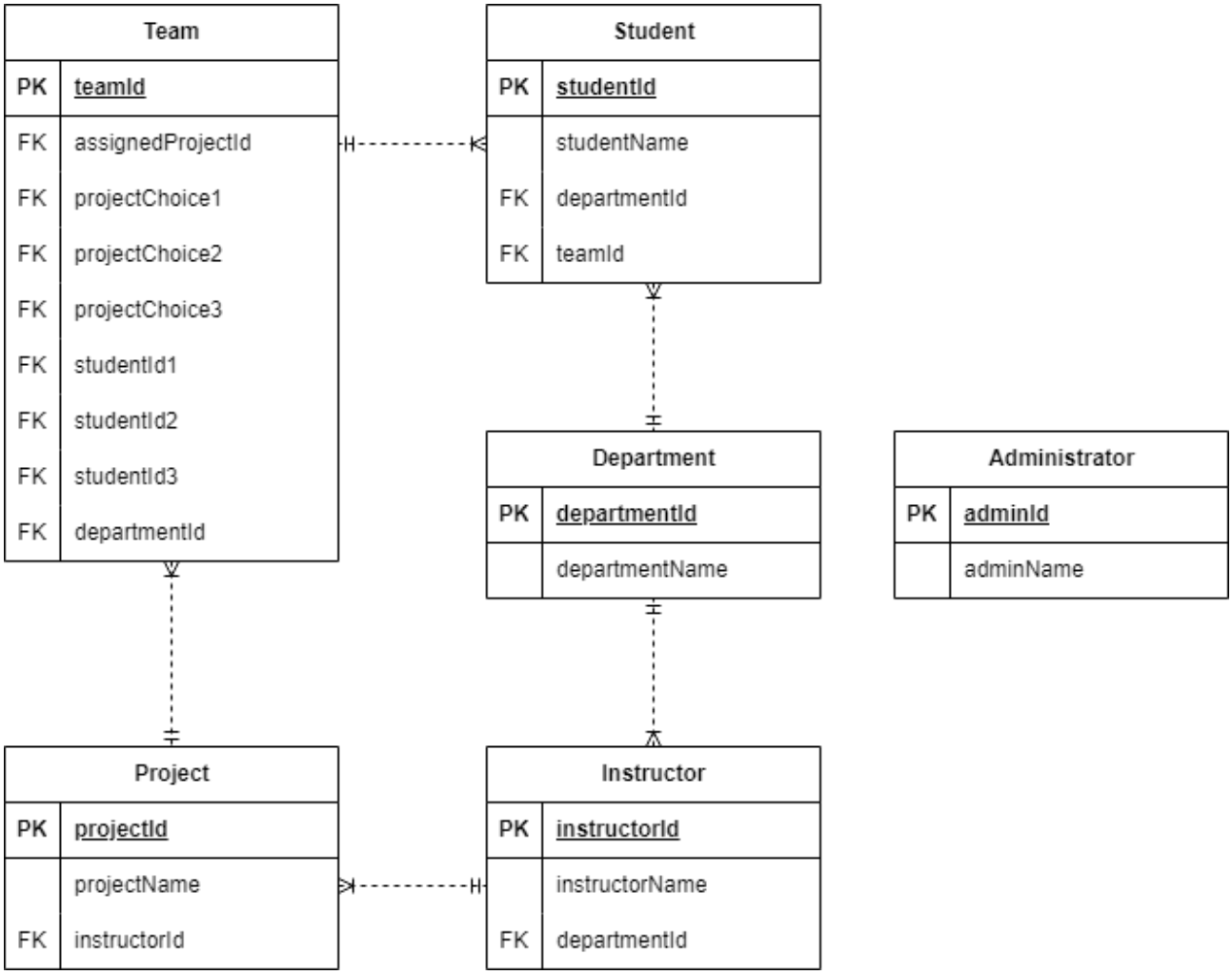


Figure 6 Database Schema

Student Table:

'studentId' (Primary Key): Unique identifier for each student.

'studentName': The student's name.

'teamId' (Foreign Key)**:** Referencing Team table, representing the team to which the student belongs.

'departmentId' (Foreign Key): Links to the Department table.

Project Table:

'projectId' (Primary Key): Unique identifier for each project.

'projectName': The name of the capstone project.

'instructorId' (Foreign Key): Links to the Instructor table.

Instructor Table:

'instructorId' (Primary Key): Unique identifier for each instructor.

'instructorName': The instructor's name.

'departmentId' (Foreign Key): Links to Department table.

Team Table:

'teamId' (Primary Key):

'assignedProjectId' (Foreign Key): Links to the Project table, representing the project assigned to the student.

'projectChoice1' (Foreign Key): Referencing Project table, representing the first preferred project.

'projectChoice2' (Foreign Key): Referencing Project table, representing the second preferred project.

'projectChoice3' (Foreign Key): Referencing Project table, representing the third preferred project.

'studentId1' (Foreign Key): Referencing Student table, representing the first team member.

'studentId2' (Foreign Key): Referencing Student table, representing the second team member.

'studentId3' (Foreign Key): Referencing Student table, representing the third team member.

Department Table:

'departmentId' (Primary Key): Unique identifier for each department.

'departmentName': Name of the department.

Administrator Table:

‘adminId’ (Primary Key): Unique identifier for each administrator.

‘adminName’: Name of administrator.

Relationships of Tables:

- **Student Table to Department Table:**
    a. **Type:** Many-to-One
    b. **Description:** Many students can belong to one department. The ‘departmentId’ in the Student table is a foreign key referencing the Department table.
- **Student Table to Team Table:**
    a. **Type**: Many-to-One.
    b. **Description:** Many students can belong to one team. The ‘teamId’ in the Student table is a foreign key referencing the Team table.
- **Project Table to Instructor Table:**
    a. Foreign Key ‘instructorId’ in the Project table links to the ‘instructorId’ in the Instructor table.
    b. Represents the instructor assigned to a project.
    c. **Type**: Many-to-One.
- **Instructor Table to Department Table:**
    a. Foreign Key ‘departmentId’ in the Instructor table links to the ‘departmentId’ in the Department table.
    b. Represents the department to which an instructor belongs.
    c. **Type**: Many-to-One.
- **Team Table to Student Table:**
    a. **Type:** One-to-Many
    b. Description: One team can have many students. The ‘teamId’ in the Team table is the primary key, and ‘studentId1’, ‘studentId2’, and ‘studentId3’ are foreign keys referencing the Student table.
- **Team Table to Project Table:**
    a. **Type**: Many-to-One
    b. **Description:** Many teams can be assigned to one project. The ‘projectId’ in the Team table is a foreign key referencing the Project table.

Database Physical Model

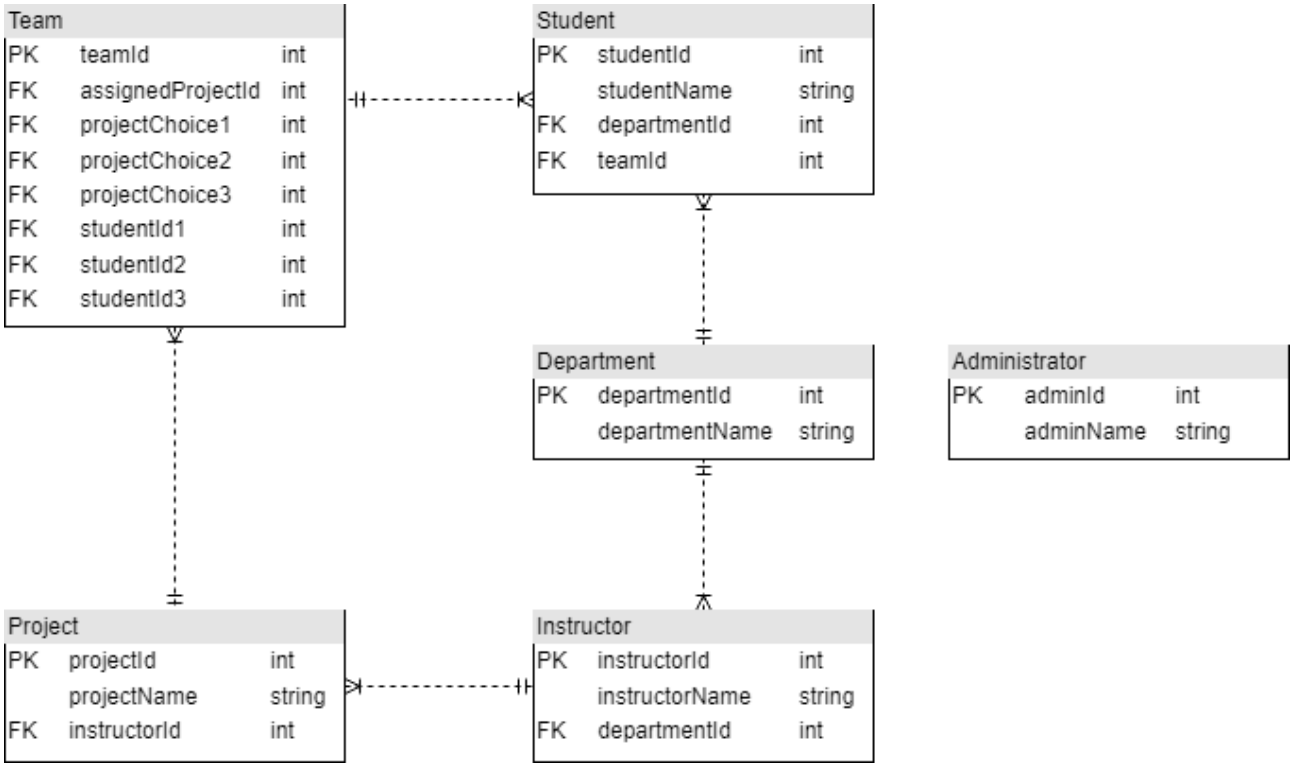


Figure 7 Database Physical Model

1. Student Table:

Entities:

'studentId' (Primary Key): Unique identifier for each student.

'studentName' (String, NOT NULL): Full name of the student.

'teamId' (Foreign Key, NOT NULL): Unique identifier referencing the team to which the student belongs.

'departmentId' (Foreign Key, NOT NULL): Unique identifier referencing the department to which the student belongs.)

Primary Key: 'studentId'

Secondary Keys: None

Constraints/Exceptions:

Each student must have a unique **'studentId'**.

The **'teamId'must** reference a valid team in the Team table.

The **'departmentId'** must reference a valid department in the Department table.

Value Model:

‘studentId’ : Integer values

‘studentName’ : String values

‘teamId’ : Integer values

‘departmentId’ : Integer values

2. Project Table:

Entities:

‘projectId’(Primary Key): Unique identifier for each project.

‘projectName’ (String, NOT NULL): Name of the project.

‘instructorId’ (Foreign Key, NOT NULL): Unique identifier referencing the instructor assigned to the project.

Primary Key: ‘projectId’

Secondary Keys: None

Constraints/Exceptions:

Each project must have a unique **‘projectId’**.

The **‘instructorId’** must reference a valid instructor in the Instructor table.

Value Model:

‘projectId’: Integer values

‘projectName’ : String values

‘instructorId’ : Integer values

3. Instructor Table:

Entities:

‘instructorId’ (Primary Key): Unique identifier for each instructor.

‘instructorName’ (String, NOT NULL): Full name of the instructor.

‘departmentId’ (Foreign Key, NOT NULL): Unique identifier referencing the department to which the instructor belongs.

Primary Key: ‘instructorId’

Secondary Keys: None

Constraints/Exceptions:

Each instructor must have a unique **‘instructorId’** .

The **‘departmentId’** must reference a valid department in the Department table.

Value Model:

‘instructorId’ : Integer values

‘instructorName’ : String values

‘departmentId’ : Integer values

4. Team Table:

Entities:

‘teamId’ (Primary Key): Unique identifier for each team.

‘assignedProjectId’ (Foreign Key, NOT NULL): Unique identifier referencing the project assigned to the team.

‘projectChoice1’, **‘projectChoice2’**, **‘projectChoice3’** (Foreign Key, NOT NULL): Unique identifiers referencing the preferred projects of the team.

‘studentId1’, **‘studentId2’**, **‘studentId3’** (Foreign Key, NOT NULL): Unique identifiers referencing the team members.

Primary Key: ‘teamId’

Secondary Keys: None

Constraints/Exceptions:

Each team must have a unique **‘teamId’** .

The **‘assignedProjectId’** must reference a valid project in the Project table.

‘projectChoice1’, **‘projectChoice2’**, and **‘projectChoice3’** must reference valid projects in the Project table.

‘studentId1’, **‘studentId2’**, and **‘studentId3’** must reference valid students in the Student table.

Value Model:

‘teamId’ : Integer values

‘assignedProjectId’ : Integer values

‘projectChoice1’, **‘projectChoice2’**, **‘projectChoice3’** : Integer values

‘studentId1’, **‘studentId2’**, **‘studentId3’** : Integer values

5. Department Table:

Entities:

‘departmentId’ (Primary Key): Unique identifier for each department.

‘departmentName’ (String, NOT NULL): Name of the department.

Primary Key: ‘departmentId’

Secondary Keys: None

Constraints/Exceptions:

Each department must have a unique **‘departmentId’** .

Value Model:

‘departmentId’ : Integer values

‘departmentName’ : String values

6. Administrator Table:

Entities:

‘adminId’ (Primary Key): Unique identifier for each administrator.

‘adminName’ (String, NOT NULL): Name of the administrator.

Primary Key: ‘**adminId’**

Secondary Key: None

Constraints/Exceptions:

Each administrator must have a unique **‘adminId’**

Value Model:

‘adminId’: Integer values

‘adminName’: String values

3.1.3. Conceptualization

Actor Glossary

Students are actors who decide on their project choices and submit them to the database. After the optimization process, Students are allowed to see their assigned projects, their group members, and their advisory Instructors.

Instructors are actors who will be mainly moderating the system with optimal authorization rights. They are allowed to adjust Student assignments manually after the optimization process and add or remove projects. However, they are not allowed to change anything related to the system’s codes and programming.

Administrators are actors who have every authorization to do anything on the system. Only the developers will have access to this user type. Administrators are given this power to provide help manually on the request of Instructors or Students.

Use-case Glossary

Table 10. Use-case Glossary Table

Use-case Name	Description	Participating Actors
Display Results	*Students and advisors view the results of project assignments.*	*Student, Instructor*
Display Project List	*Students and Advisors view the list of available projects.*	*Student, Instructor*
Submit Preferences	*Students submit their project preferences.*	*Student*
Login	*Users authenticate themselves to access the system.*	*Student, Instructor*
Add Project	*Administrators or Instructors add new projects to the system for students to choose from.*	*Administrator, Instructor*
Create Team	*Students create a new Team to be able to make choices.*	*Student*

Use-case glossary presented in Table 10.

Use-case Scenarios

Use case scenarios are presented in Table 11.

Table 11. Use Case Scenario Tables

Use-case Name	*Display Results*
Use-case Description	*Students and Instructors view the results of project assignments.*
Actors	*Student, Instructor*
Pre-Condition	*Project assignments have been optimized.* *The user is already logged in.*
Post-Condition	*Results are displayed to the user.*
Normal Flow	*Step 1: Student or Instructor clicks on the results button.* *Step 2: Student or Instructor navigates to the results page.* *Step 3: System retrieves and displays the optimized project assignments.*
Alternate Flow	*Alt-Step 3: If there is an issue with result retrieval, display an error message.*
Business Rules	*The display results button is deactivated until the system is done with the optimization.*

Use-case Name	*Display Project List*
Use-case Description	*Students and Instructors view the list of available projects.*
Actors	*Student, Instructor*
Pre-Condition	*Project information is available in the system.* *The user is already logged in.*
Post-Condition	*Project list is displayed to the user.*
Normal Flow	*Step 1: Student or Instructor clicks on the project list button.* *Step 2: Student or Instructor navigates to the project list page.* *Step 3: System retrieves and displays the list of available projects.*
Alternate Flow	*Alt-Step 3: If there is an issue with project list retrieval, display an error message.*
Business Rules	*Students are allowed to see every detail of a project on the project list page. This is important as Teams make their choices depending on the information on this page.*

Use-case Name	*Login*
Use-case Description	*Users authenticate themselves to access the system.*
Actors	*Student, Instructor, Administrator*
Pre-Condition	*The user already has a registered account on the system.* *The user is not logged in.*
Post-Condition	*The user gains access to the system upon successful authentication.*
Normal Flow	*Step 1: The user navigates to the login page.* *Step 2: The system presents a login form requesting ID.* *Step 3: The user enters their valid ID.* *Step 4: The system verifies the entered credentials against the stored user data.* *Step 5: If the credentials are valid, the system authenticates the user and grants access to the system.* *Step 6: The user is redirected to the system's main page related to their roles.*
Alternate Flow	*Alt-Step 5: If the entered credentials are invalid, the system notifies the user and prompts them to re-enter the correct information.*
Business Rules	*Users must have been registered to the system before logging in.*

Use-case Name	*Submit Choices*
Use-case Description	*Students submit their project choices.*
Actors	*Student*
Pre-Condition	*The Student who submits project choices must belong to a Team.* *The Student must have chosen 3 projects before submission.* *The user is already logged in.*
Post-Condition	*Team project choices are stored in the system.*
Normal Flow	*Step 1: The Student selects three projects from the project list and prioritizes them.* *Step 2: The Student submits choices.* *Step 3: The system stores preferences in the database.*
Alternate Flow	*Alt-Step 1: If a student doesn't choose exactly three projects, display an error message to the student.* *Alt-Step 3: If there is an issue with preference submission, display an error message to the student.*
Business Rules	*Submit button will not be functional until exactly 3 projects are chosen.*

Use-case Name	*Add Project*
Use-case Description	*Instructors add new projects to the system for Students to choose from.*
Actors	*Instructor*
Pre-Condition	*The Instructor is already logged in.*
Post-Condition	*New projects are successfully added to the system and are available for student selection.*
Normal Flow	*Step 1: The Instructor clicks on the add project button.* *Step 2: The system presents a form for adding new projects, including fields for project name, description, Instructor ID, and other relevant details.* *Step 3: The Instructor enters the required information for the new project.* *Step 4: The Instructor submits the form to add the projects to the system.* *Step 5: The system validates the entered information and adds the new projects to the project list.* *Step 6: The Instructor receives a confirmation message indicating successful addition.*
Alternate Flow	*Alt-Step 2: If the Instructor exceeds the character limit for new project description or enters an invalid Instructor ID, the system gives an error.*
Business Rules	*Project description can not exceed a certain character limit.* *Created project must have a valid Instructor assigned to it.*

Use-case Name	*Create Team*
Use-case Description	*Students create a new Team to be able to make choices.*
Actors	*Student*
Pre-Condition	*The Student is registered.*
Post-Condition	*A new group is successfully created with at least one and at most three Students.*
Normal Flow	*Step 1: The Student logs into the system and navigates to the main page.* *Step 2: The Student chooses a department that will be assigned to the Team and enters between one or three Student IDs.* *Step 3: The system checks if the Student IDs match with the related department.* *Step 4: The system creates the Team on the database and relates the Student IDs with the Team ID.* *Step 5: The system notifies the Student that the process is successfully completed.*
Alternate Flow	*Alt-step 1: If there is an issue with the entered information (e.g. wrong or non-existent student ID), the system provides an error message, and the Student corrects the information before logging in.* *Alt-step 3: If the chosen department and any Student ID does not match or any Student ID is already assigned to a Team, the system sends a warning to the Student and returns the Student to Step 2.*
Business Rules	*Teams can only have between one and three Students.* *A Student can not join more than one Team*

Use-case Diagram

Major actions that can be performed by users are implied in this Use Case Diagram in Figure 8.

Figure 8 Use Case Diagram

Interface Design

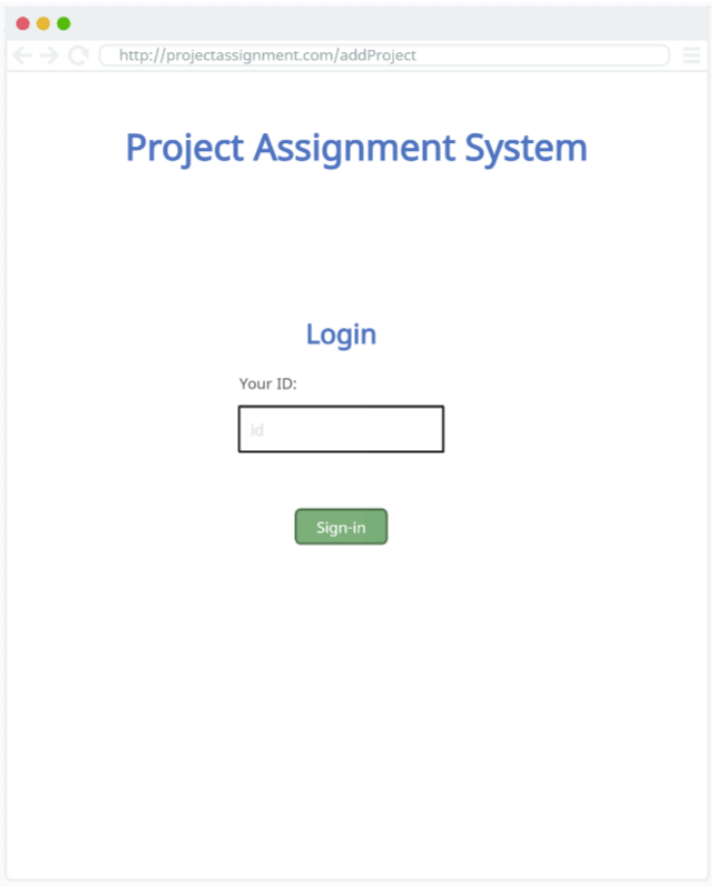


Figure 10 Mockup Login Page

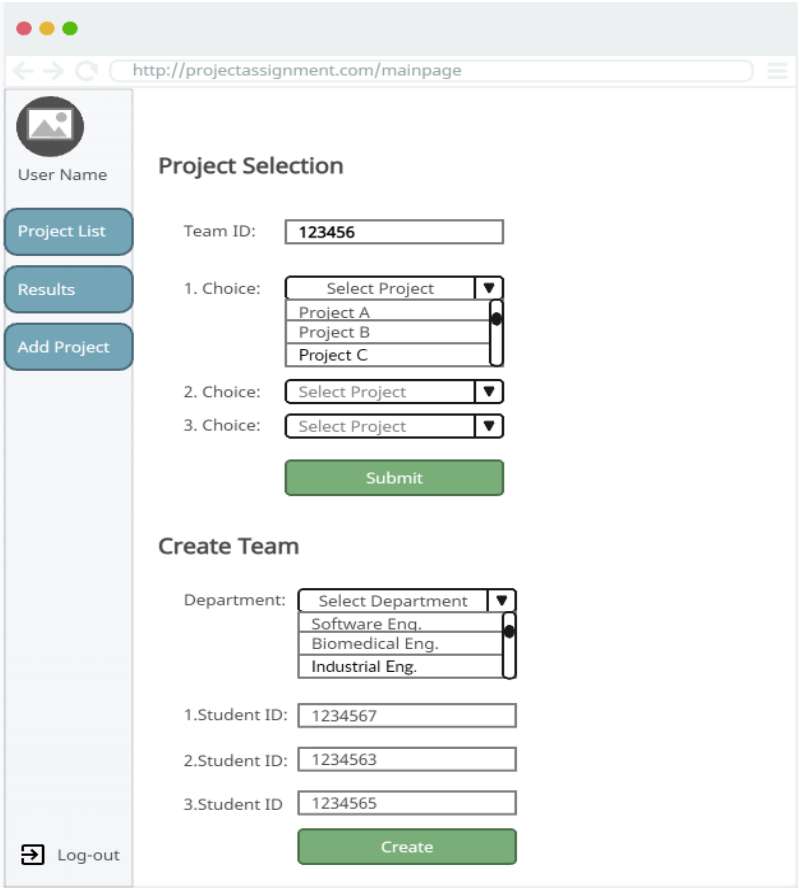


Figure 9 Mockup Main Page

Figure 10. presents the mockup UI design of the web application's login page. This is the first page user will see when engaging with the application. The user will need to enter his/her ID and click on the Sign-in button to proceed to the main page. If the user's ID is incorrect, the system will display a warning message to the user.

Figure 9. presents the mockup UI design of the web application's main page, this is the page user will navigate to once authenticated by the system. On the left bar of the main page, the user will see his name and photo and can navigate to Project List Page, Results Page and Add Project Page and can Log-out when clicked on the corresponding button. On the middle section of the page, users can create a team by selecting the department from the dropdown list and entering team members' student IDs. Once the user creates a team, the team will be allowed to make project choices from the dropdown list and submit their choices. The user will be warned by the system in any case of wrong input entry.

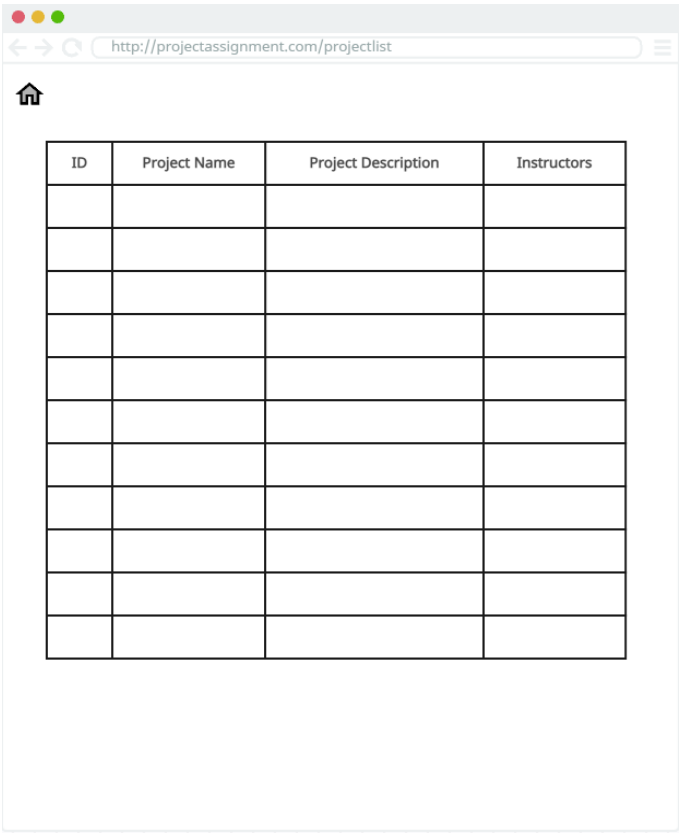


Figure 12 Mockup Project List Page

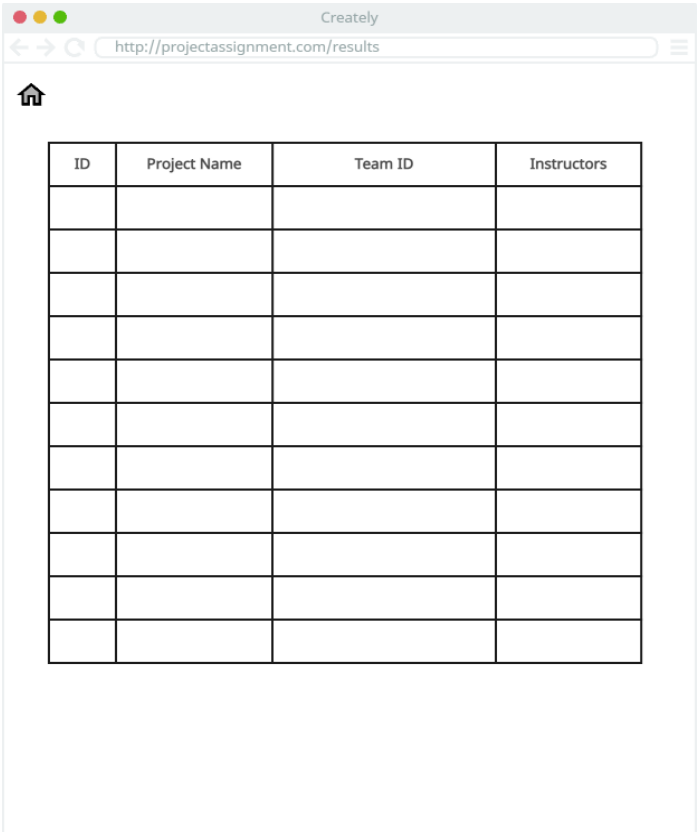


Figure 11. Mockup Result Page

Figure 12. presents the mockup UI design of the web application's Project List page. Users can display the projects in a table with the project's ID, name, description and Instructors. Users can navigate back to the Main Page by clicking on the home button.

Figure 11. presents the mockup UI design of the web application's Results page. Users can display the results in a table with Project's ID, name, assigned team's ID and Instructors of the project. Users can navigate back to the Main Page by clicking on the home button.

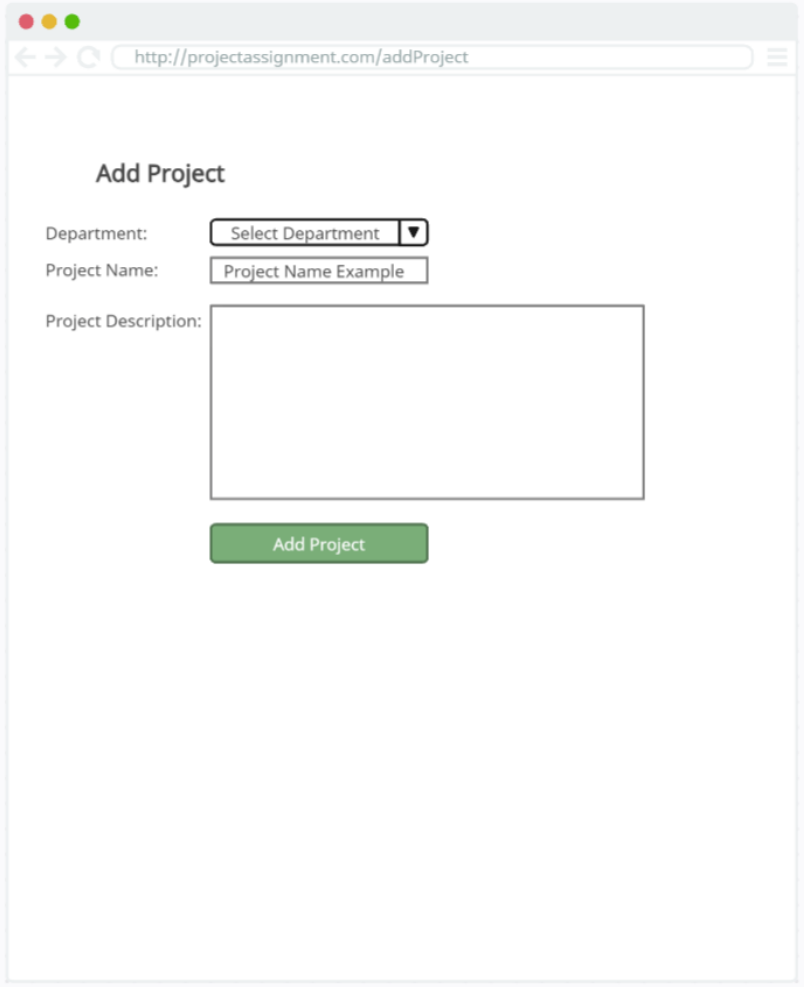


Figure 13. Mockup Add Project Page

Figure 13. presents the mockup UI design of the web application's Add Project page. Users can add projects to the list by selecting the department from the drop-down list and entering the project's name, description. Once clicked on the add project button, the user will be notified about the saved result.

UML Class Diagram

The planned objects are shown in this UML Class Diagram in Figure 14. These object classes include the object names and their attributes. Also, on the bottom of these object classes, the expected functions, mostly getters and setters, are present.

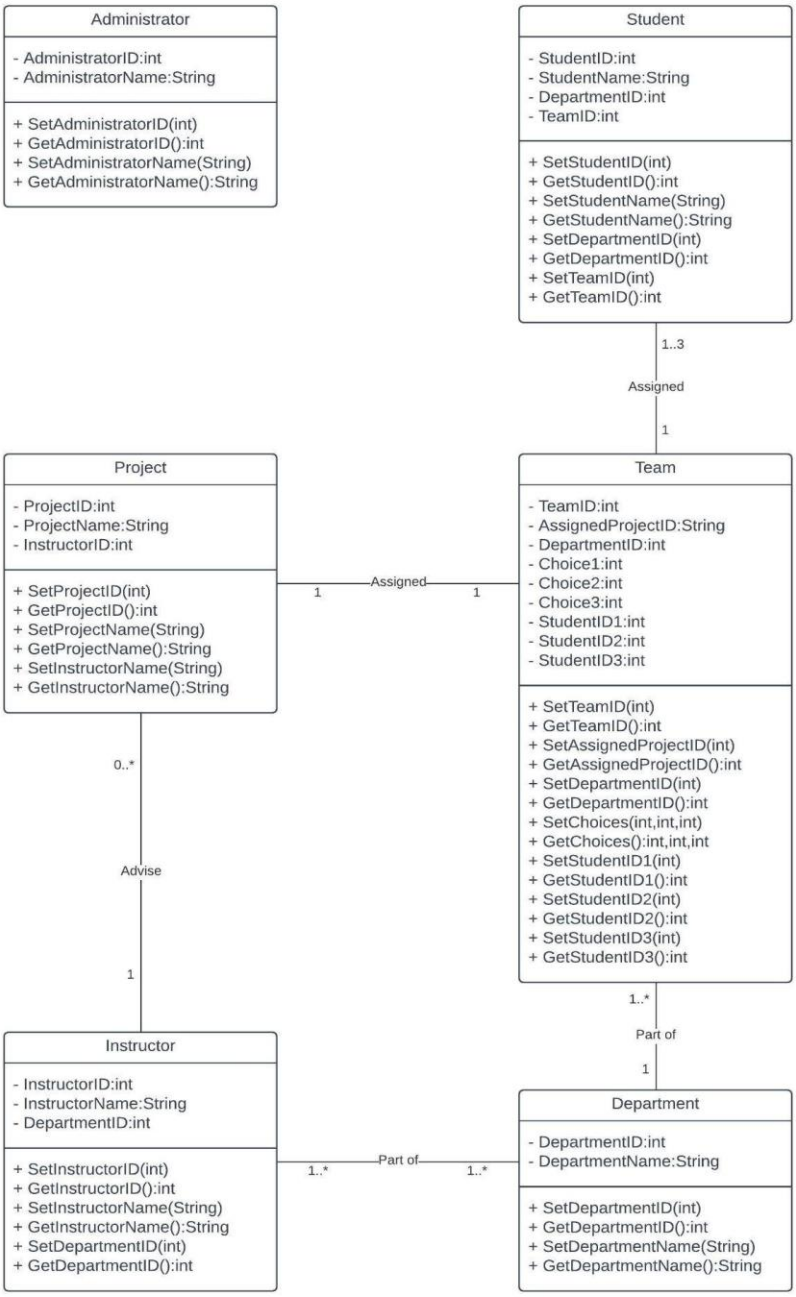


Figure 14 UML Class Diagram

Activity Diagram

This Activity Diagram in Figure 15 shows the planned navigation of user activities. A general idea about the connection of web pages can also be found here.

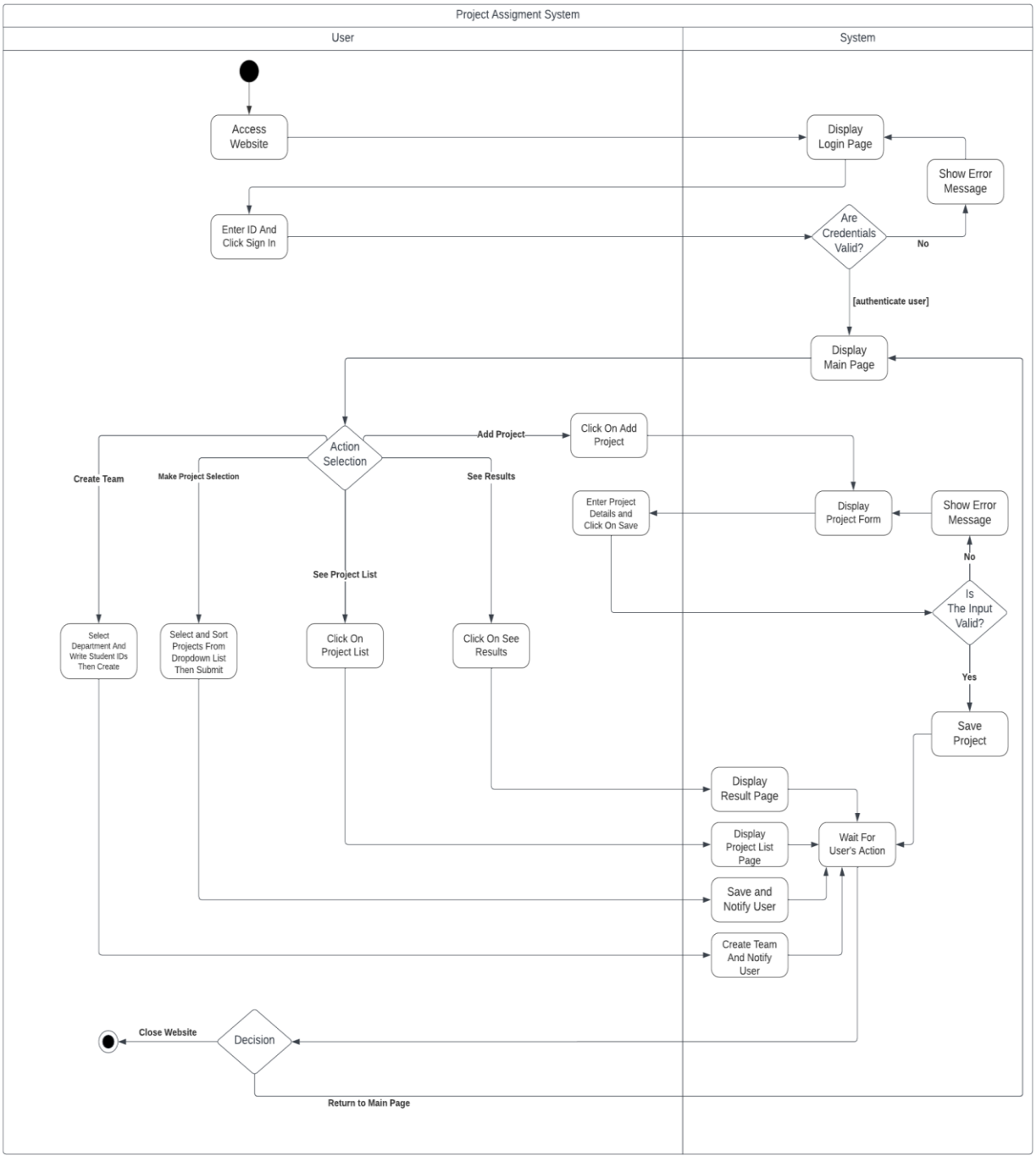


Figure 15 Activity Diagram

Sequence Diagram

The Sequence Diagram in Figure 16 down below shows a possible interaction timeline between the user and the system components. Users can trigger certain functions in order to change pages or change data on the database.

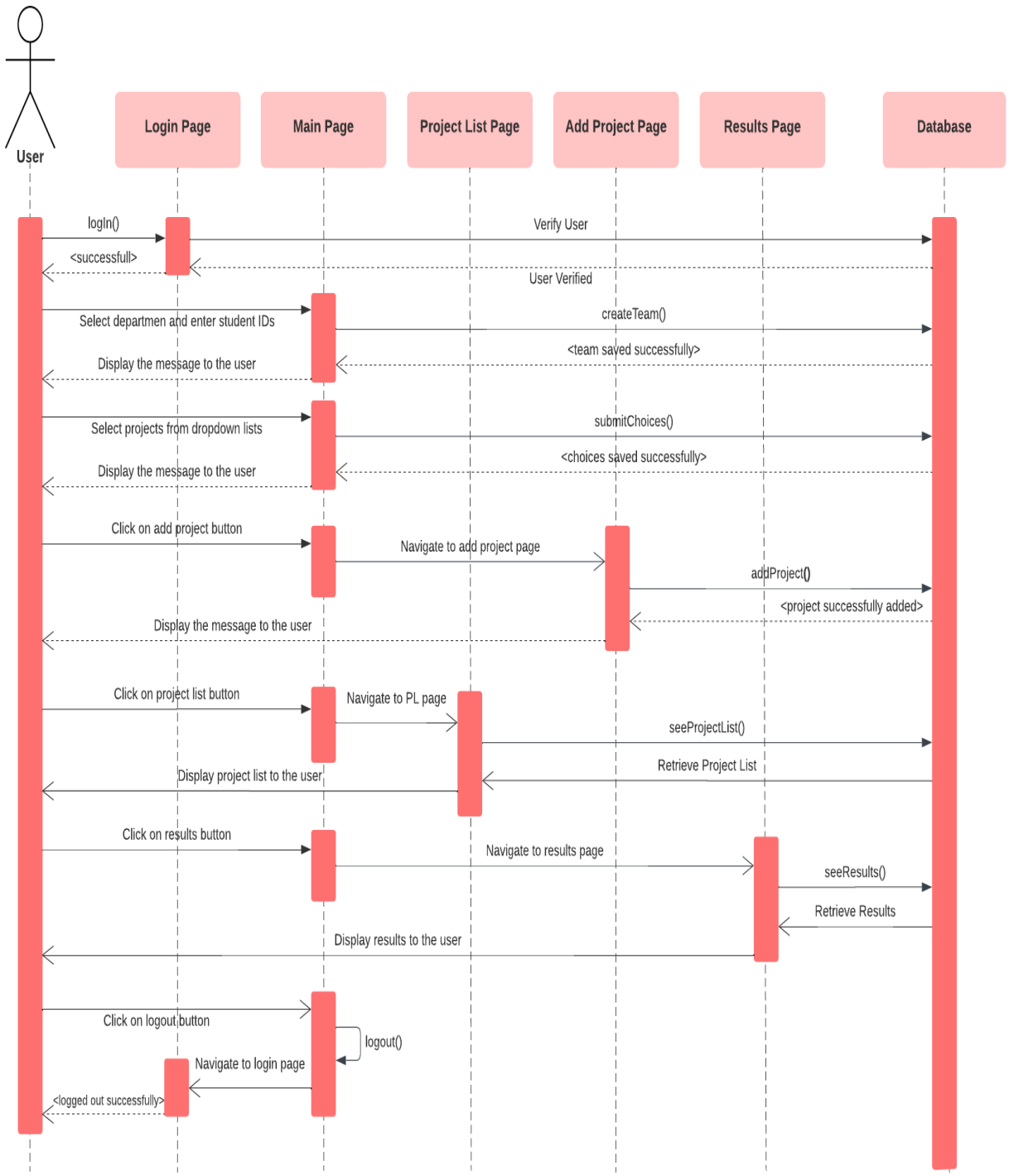


Figure 16. Sequence Diagram

Data Flow Diagram

The Data Flow Diagram in Figure 17 shows the interaction of data between users, functions, and the database. This diagram also shows the required inputs and result outputs of different functions of the system.

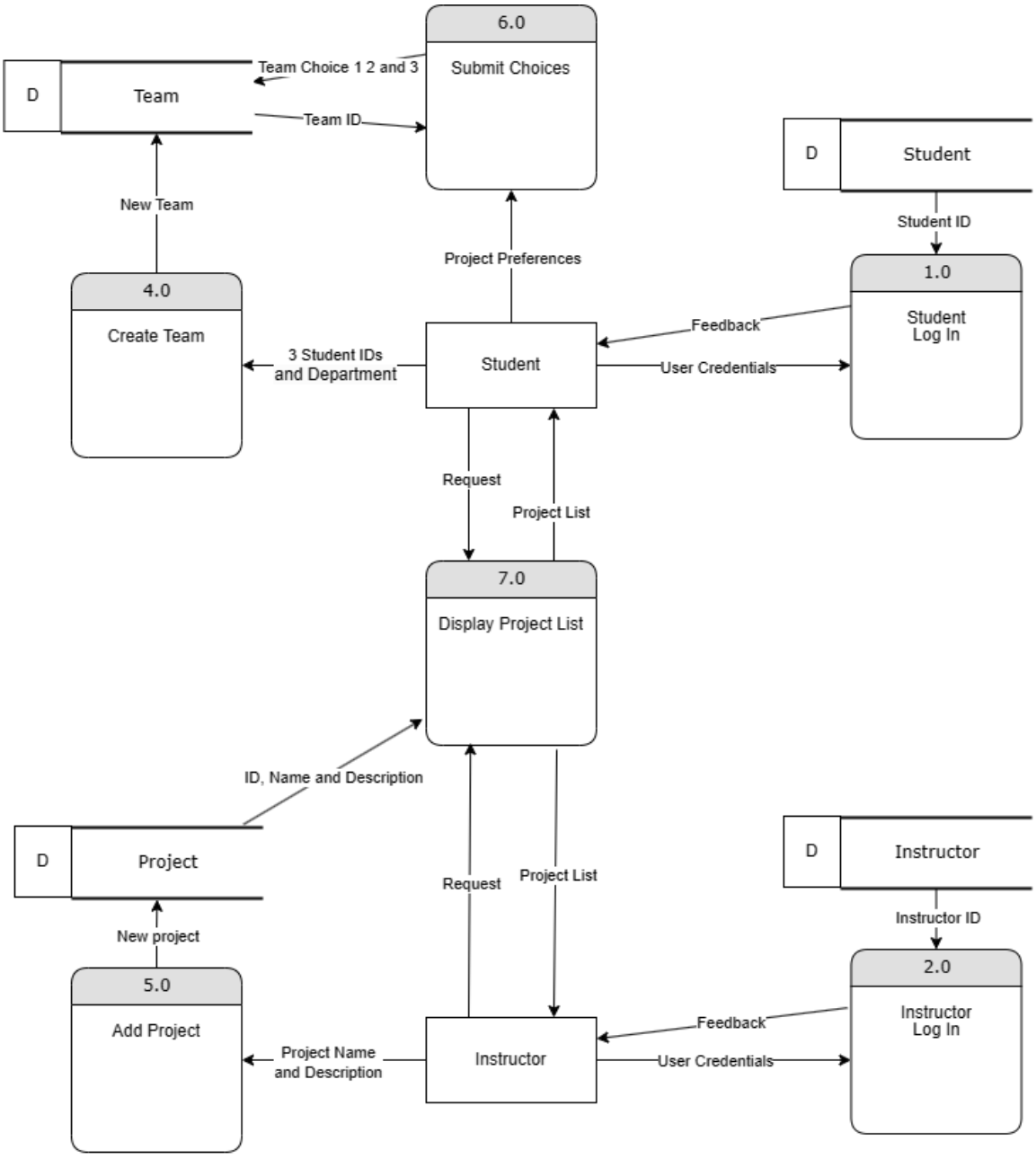


Figure 17. Data Flow Diagram

3.1.4. Software Architecture

As briefly mentioned in the 1.3.2, 1.4 and 3.1.2, User Interface and Backend parts of the software are going to be a .NET Core web application built with Model-View-Controller design and N-tier architectural patterns. In this part, these two patterns will be explained in detail.

Model-View-Controller (MVC) is a popular architectural pattern that improves software development by separating issues and providing clarity. It separates the application into three interconnected parts:

1. Model: The business logic and data of the application are represented by the Model. This usually involves creating entities and managing data-related operations within our application. It guarantees the data consistency and integrity of the application.
2. View: The View is in charge of gathering user input and displaying the data to the user. This relates to the user interface (UI) components in our application. It guarantees that the application logic underlying and the presentation layer are kept separated.
3. Controller: The Controller serves as a link for the View and the Model. After handling user input, it modifies the View by triggering actions on the Model. Controllers manage the data flow in our project, updating the Model and View as needed and making choices depending on user interactions.

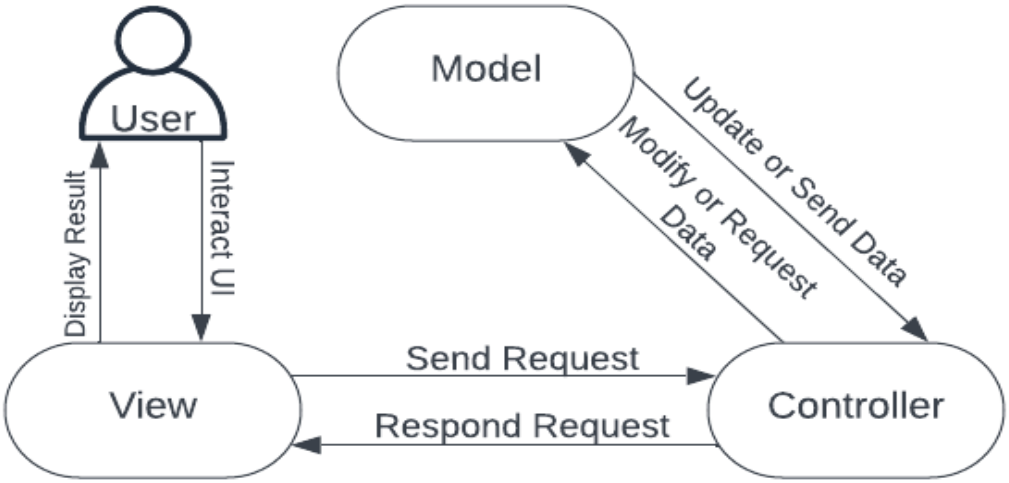


Figure 18. MVC Diagram

Figure 18. shows how the components of MVC communicates when user interacts with the system.

Benefits of Using MVC:

Separation: MVC makes the software easier to understand, maintain, and expand by separating the application into three discrete components, each of which has a specific function.

Collaboration: Because the code is separated into three tiers, developing web applications using the MVC architecture allows developers to work on different areas of the project at the same time. This feature speeds the development phase significantly.

Testability: One of the main benefits of the MVC pattern is that it makes testing easier. Unit testing of separate components is made possible by MVC, which facilitates the process of verifying that each component of the application is correct.

N-tier Architecture:

The N-tier architecture expands on MVC by introducing additional layers, each with its distinct responsibilities:

1. Entity Layer: This layer represents the data entities and their relationships. In our application, this involves defining classes that correspond to database tables. The Entity Layer ensures a clear and consistent representation of data throughout the application.
2. Data Access Layer (DAL): The DAL is responsible for interacting with the database. It includes functions for data retrieval, storage, and manipulation. In our application, the DAL often utilizes technologies like Entity Framework Core to provide a seamless interface between the application and the underlying database.
3. Business Layer (BL): The Business Layer contains the core business logic of the application. It defines how data is processed and enforces business rules. In C#, this layer encapsulates the application's unique functionality, ensuring that it remains independent of the data access and presentation layers.
4. View Layer: The View Layer encompasses the user interface components responsible for rendering the application. In our project, this includes UI elements and their interactions.

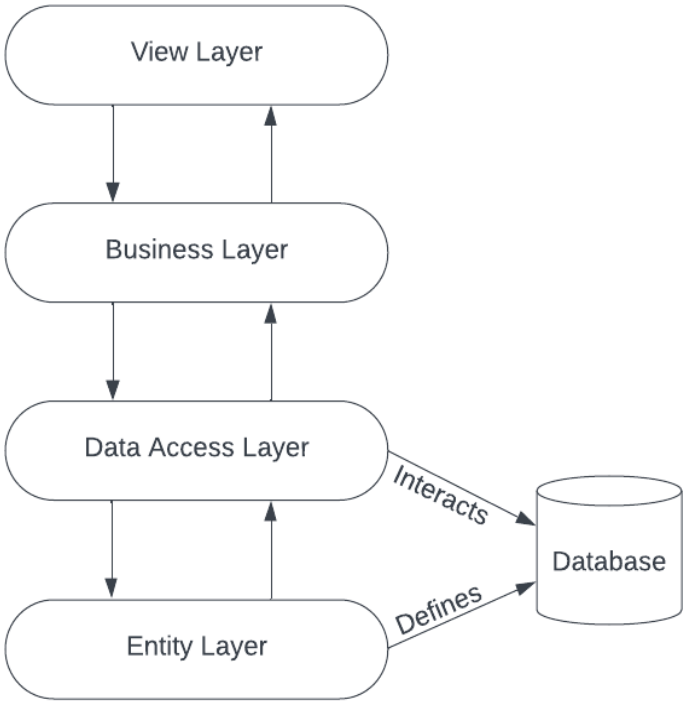


Figure 19. N-Tier Architecture

Figure 19. shows the layers of the software architecture and how they interact with each other and the database.

Benefits of Using N-tier Architecture:

N-tier architecture improves scalability and maintainability by distributing the application logic across different layers. It enables code reuse and allows changes in one layer to be made without affecting others. This separation improves security by limiting access to sensitive data via well-defined interfaces.

Combining MVC with an N-tier architecture in our .NET Core Web Application provides a structured and scalable foundation, promoting maintainability, code reuse, and efficient development practices.

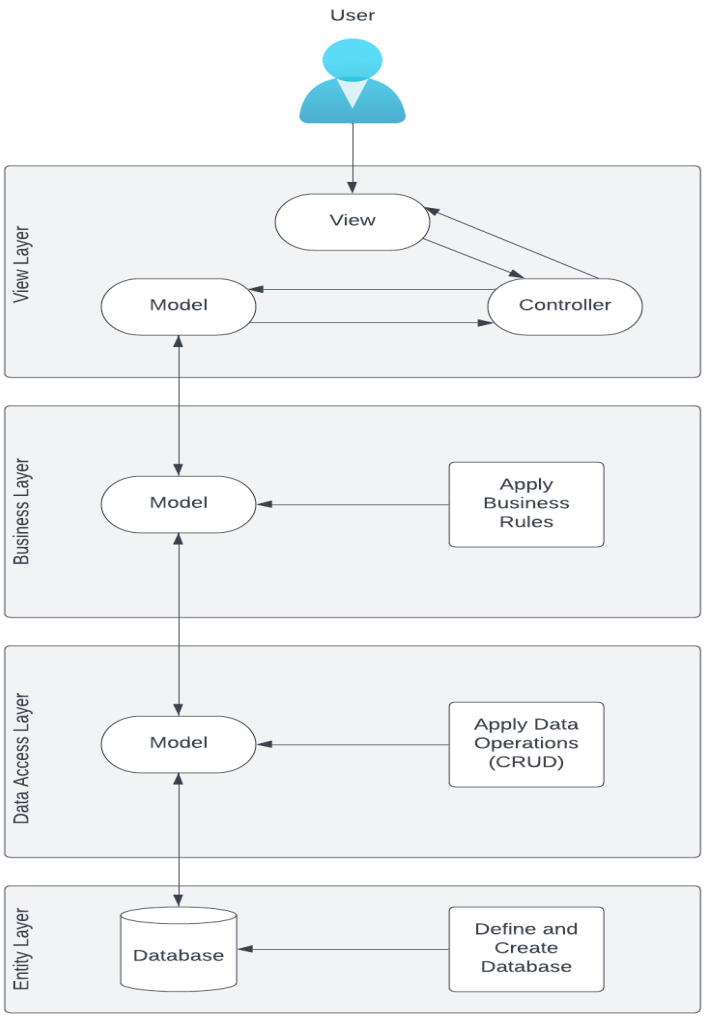


Figure 20. Combined Software Architecture

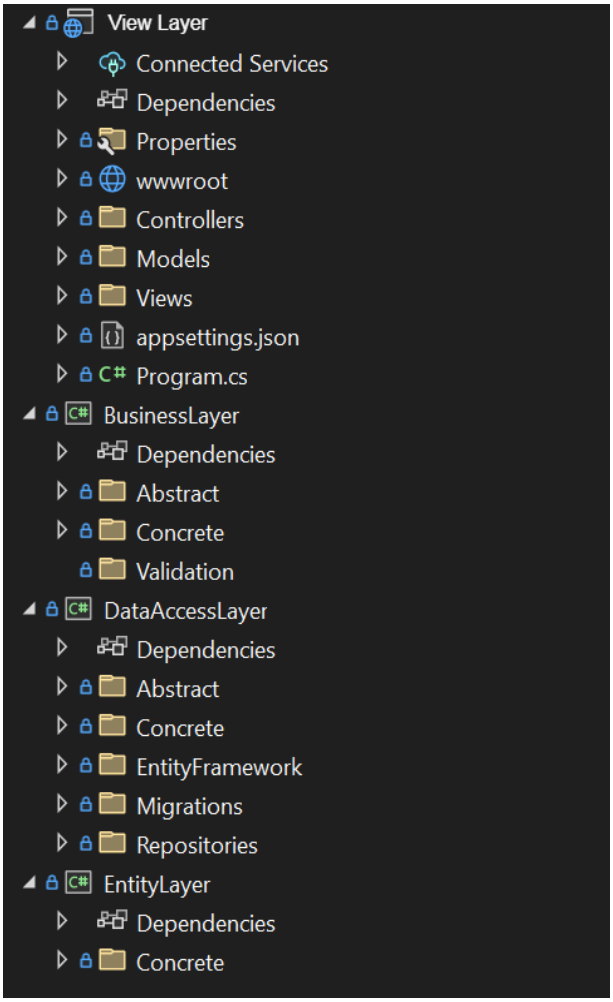


Figure 21. Project Structure Example

Figure 20. shows how will software operate when MVC and N-tier architecture patterns combined.

Figure 21. shows how the project structure will look like in the Solution file of the project.

3.1.5. Implementation

The implementation process began by creating an ASP.NET Core Web App (MVC) project with version .NET 8.0. Necessary NuGet packages have been installed in order to provide additional functionality to the project. These packages can be found in Figure 22. below.

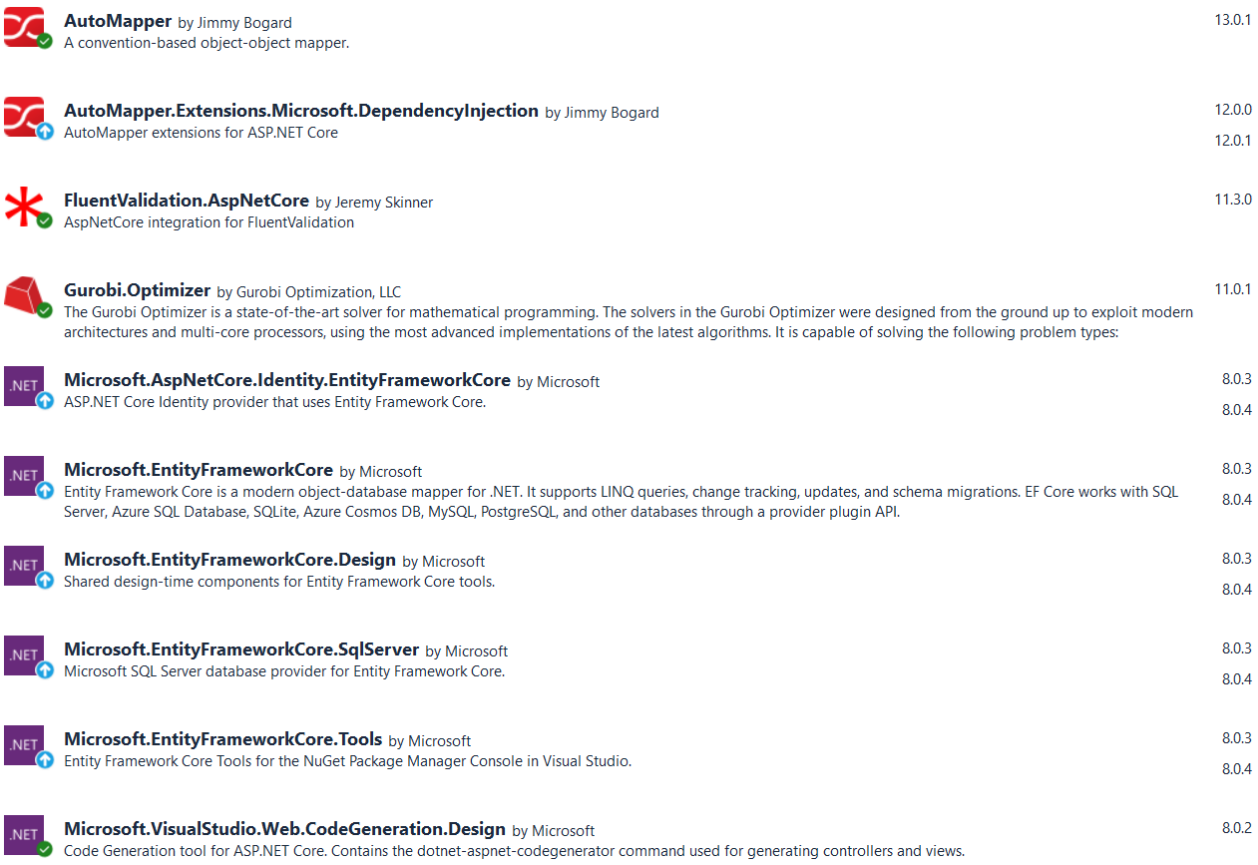


Figure 22. Installed NuGet Packages

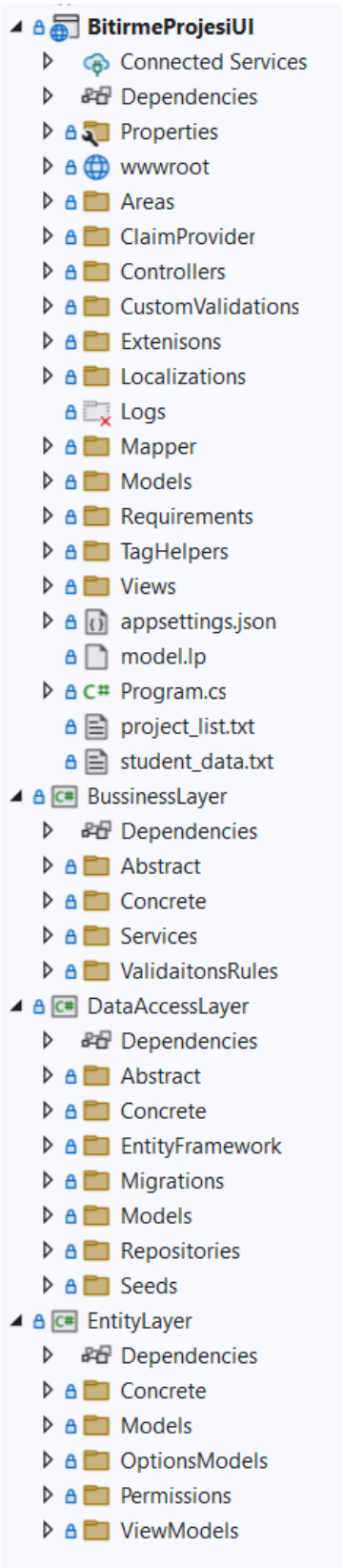


Figure 23. Implemented Layers

First, the n-tier architecture was created as proposed early in 3.1.4. Four layers were made, each serving different purposes. Figure 23. presents all the layers of the project.

Entity Layer contains the necessary attributes for object types of different kinds. They are placed in the Concrete folder in the Entity Layer. The object classes residing in the Concrete folder define the database tables.

In the Data Access Layer, interface and repository files were created which define CRUD operations. The migration was done, the tables were sent to the database with the attributes inside which provides connectivity to the database.

In the Business Layer, services, managers and validators were created in order to set business rules of the website.

In the View Layer, views and controllers were created which provides UI design and UI element functionalities.

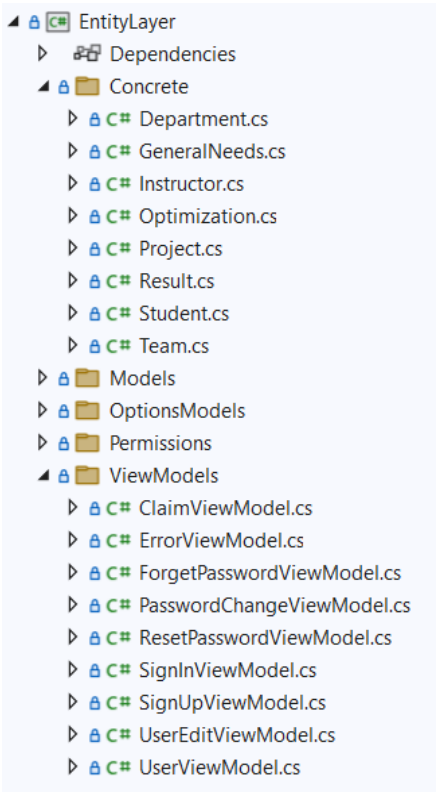


Figure 24. Entity Layer Details

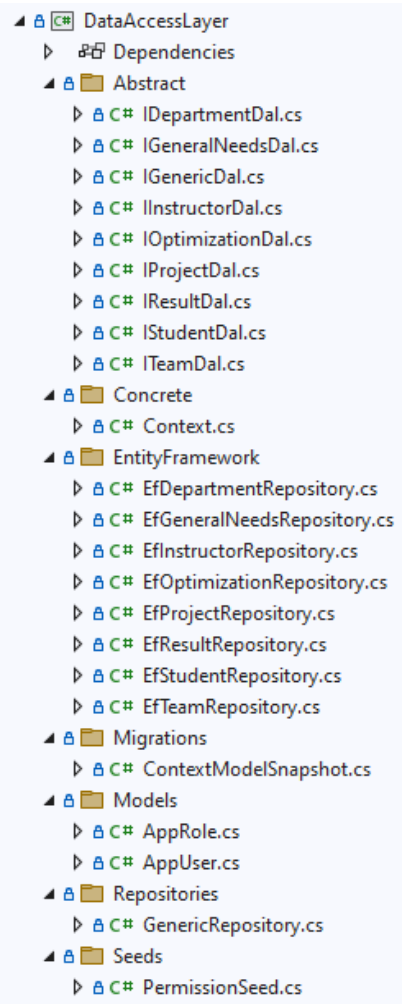


Figure 25. Data Access Layer Details

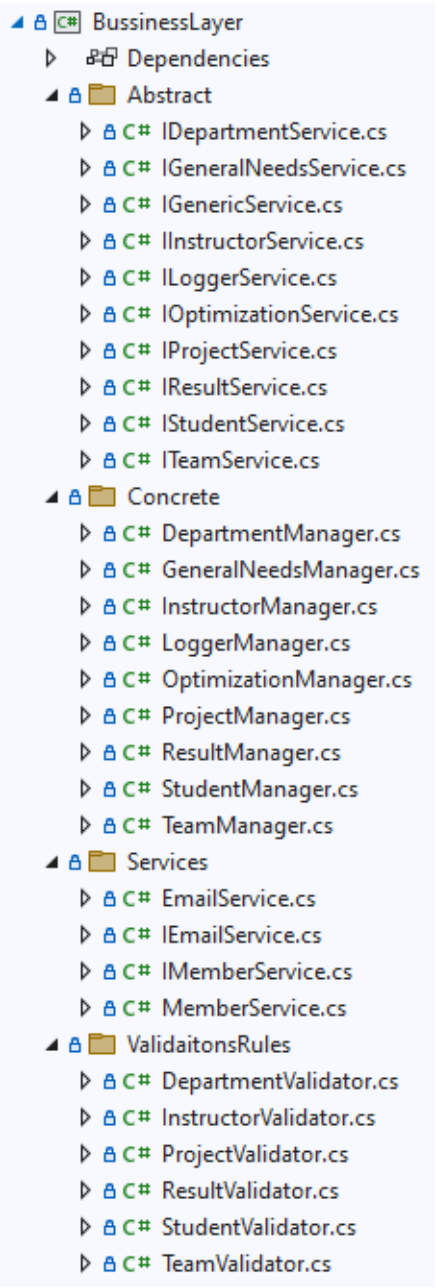


Figure 26. Business Layer Details

Managers, services, interfaces, repositories and validators were created for each entity in related layers shown in Figures 24. 25. 26. Generic classes created to achieve code reusability. ViewModels and Models imported from Core Identity Library. Approle and Appuser were used in the data access layer for user and role management. Again, there are viewmodel classes in the entity layer for the Identity library. When accessing and performing operations on all entities on the business layer and data access layer, instead of writing for each entity one by one, by using Generic and assigning variables in Generic, the relevant classes in the data access layer and business layer were enabled to automatically retrieve the necessary data manipulation functions.

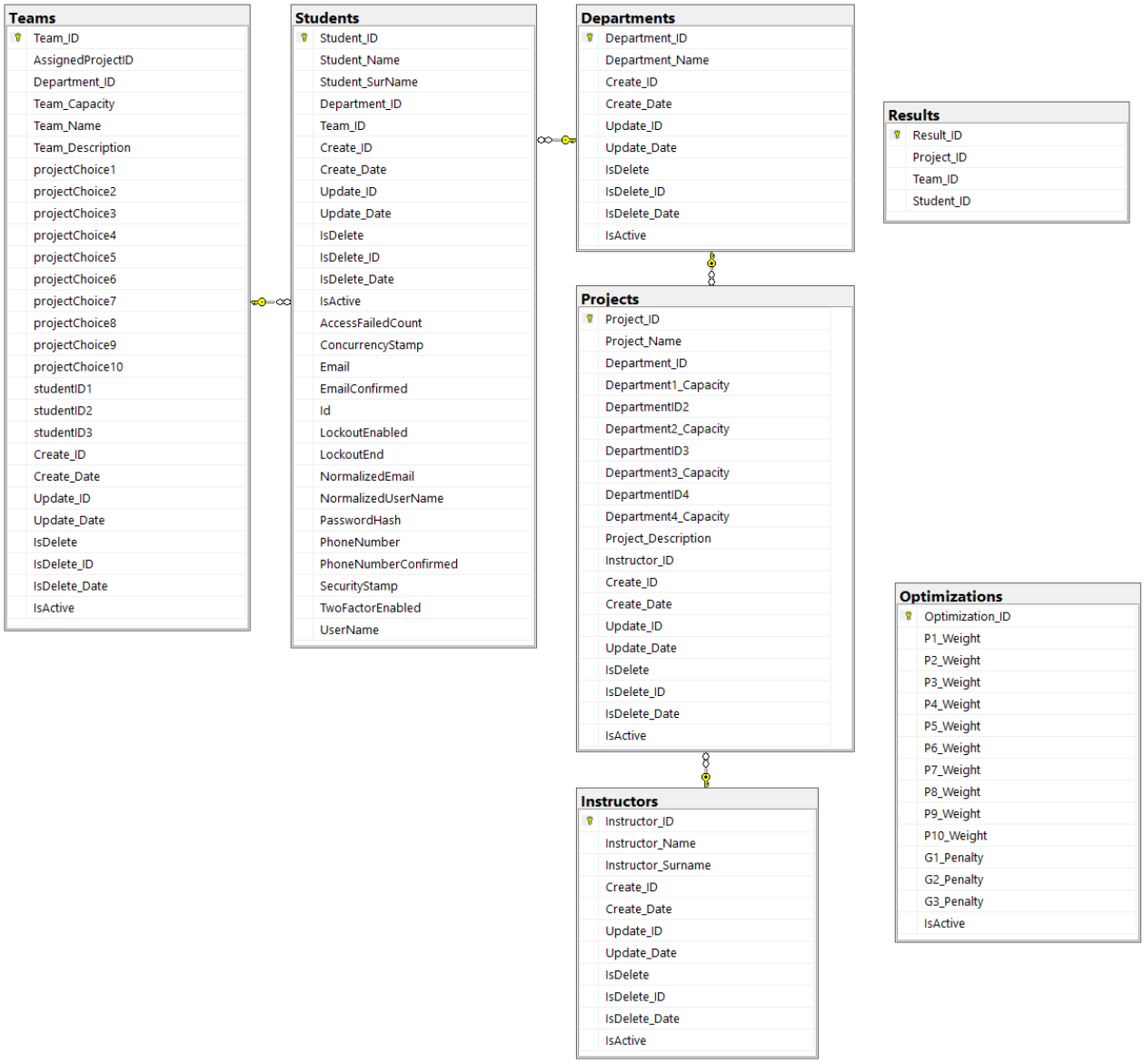


Figure 27. Updated DB Diagram

Updated DB shown in Figure 27. is created by using the methods we explained in 3.1.2. and it is fully functional with .NET Core Framework. Relations between data tables have been set. At first, a sample data set was entered into the DB manually. DB is redesigned based on the feedback given by project advisors and Industrial Engineering team. There are also other additions to the DB designed in 3.1.2 due to necessities.

To be able to log user activities, “Create_ID”, “Create_Date”, “Update_ID”, “Update_Date”, “isDelete”, “isDelete_ID”, “isDelete_Date” and “isActive” attributes are added. Attributes from “AccessFailedCount” to “UserName” in the Student data table are added for Microsoft.AspNet.Identity.Core usability.

Team data table is additionally given “Team_Name”, “Team_Description”, “Team_Capacity” and 7 more “projectChoice” attributes. The first two are added for better communication between students while the rest are added with the request of our advisors for better optimization results.

In the Project data table, 4 new “Department_ID” and “Department_Capacity” attributes are added by the request of the Industrial Engineering team as constraints for the optimization engine.

Lastly, two new data tables, results and optimization, are added. Results data table is filled with data coming from the optimization engine. Optimization data table contains the settings for optimization rules.

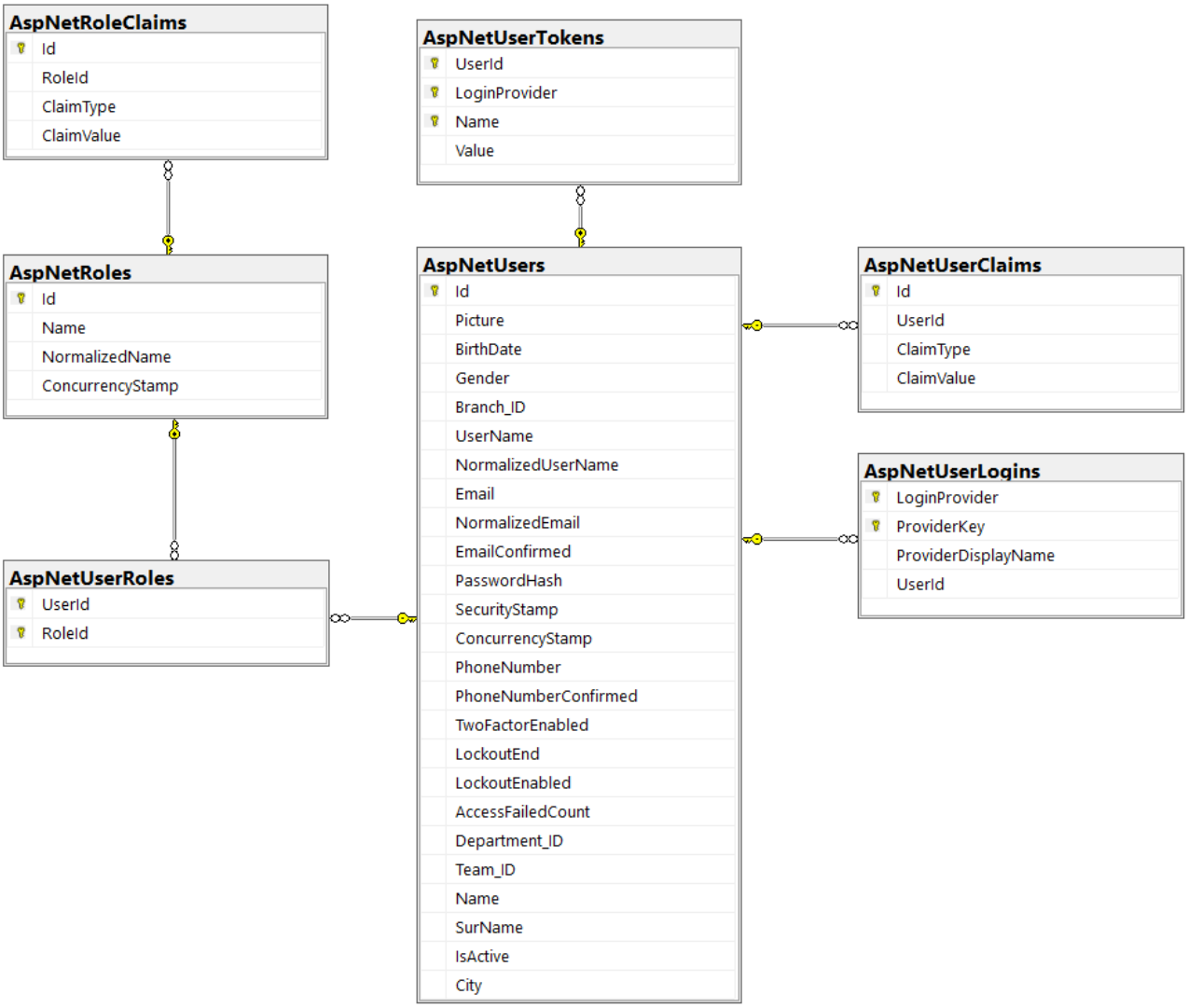


Figure 28. Core Identity DB Diagram

In addition to the project proposal, Microsoft.AspNet.Identity.Core library was used for user authorization and management. This framework provided authentication integration and identity management functionality much more efficiently and connectivity to the database provided and

increased.

The implementation of .NET Core Identity-related tables shown in Figure 28. is done by the same methods, these tables are integral to user authentication, role management, and access control. The AspNetUsers table stores user attributes like usernames, emails, and hashed passwords, while AspNetRoles defines various user roles with specific permissions. AspNetUserRoles establishes user-role relationships, AspNetUserClaims holds additional user profile information, AspNetUserLogins manages login providers, AspNetUserTokens secures authentication with tokens, and AspNetRoleClaims stores role-specific permissions. Together, these tables form a comprehensive identity management system, enabling secure and customized user experiences within the application.

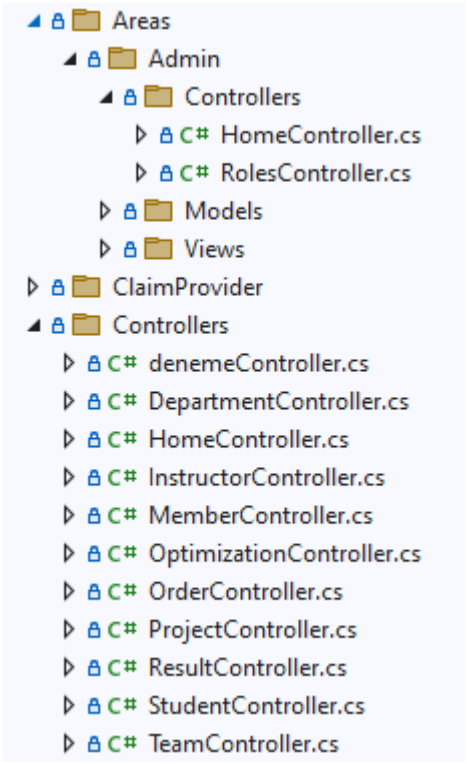


Figure 29. Controller Classes

After completing the implementation of the Entity, Data Access and Business layers. We proceeded with the View Layer.

Controller classes shown in Figure 29. are created that use services and managers defined in the Business Layer and interfaces to control the operation requests that come from UI like List, Add, Update, Delete via HttpGet, HttpPost method

After creating all the controllers needed as shown in Figure 29., we started the implementation process of the User Interfaces (Views).

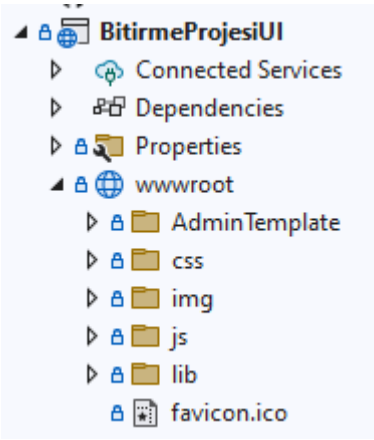


Figure 30. Theme Package

At the start of the UI implementation process, we successfully imported the theme package shown in Figure 30. to use for the implementation of the UI. It includes all the necessary JS and CSS elements that we used for the implementation of the user interfaces.

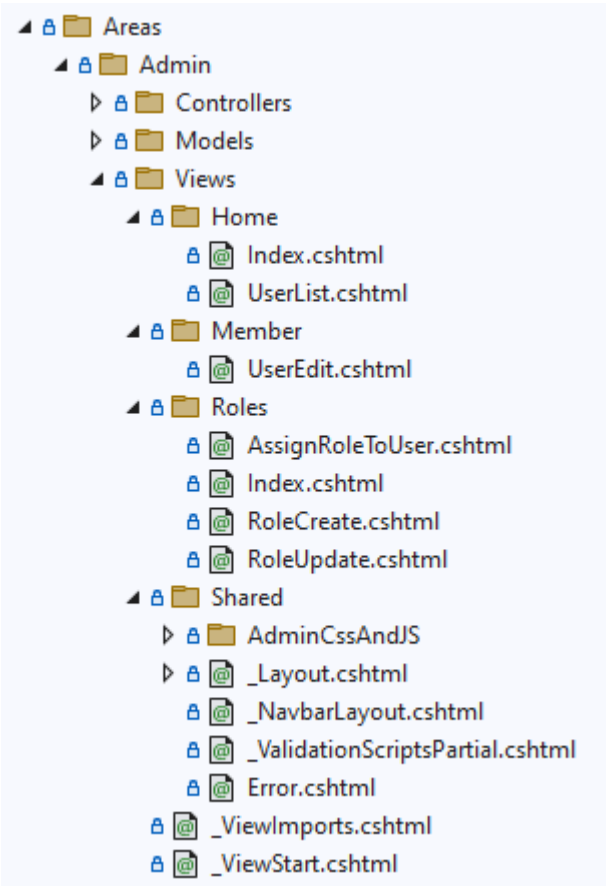
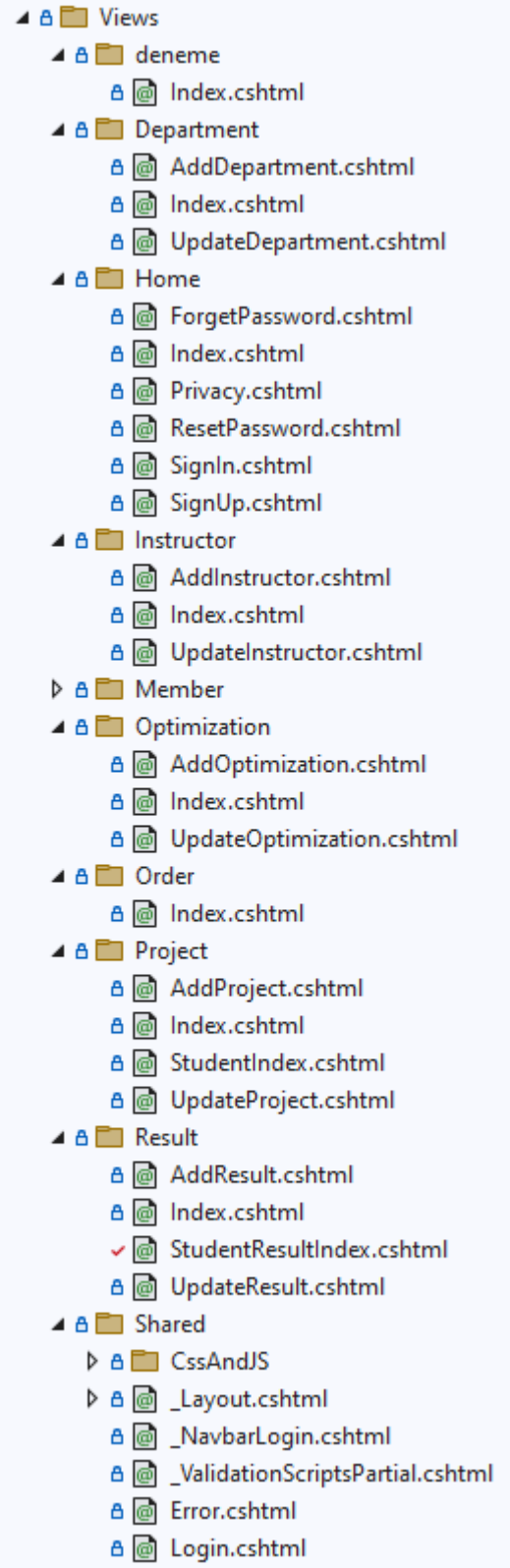


Figure 31. Admin Area Views

The views of the interfaces we defined in the Admin area controllers were created. We also created partial and layout views in the shared folder for the admin layout, navigation bar and scripts for achieving more organized code as shown in Figure 31.



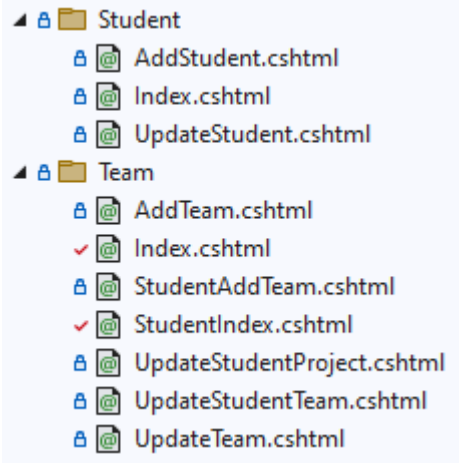


Figure 32. All the Views Created

All other views of the HttpGet and HttpPost interfaces that were defined on the controllers were created. Also, from the Identity library models, sign in, sign up, and password management related views were created. Figure 32. presents all the views created.

The coding of these views was done simultaneously with the controllers' coding. After coding the necessary interfaces in the controllers, for example AddDepartment [HttpPost], a related view was created and coded simultaneously. The process of view creation was repeated for every other interface in the project and completed successfully.

Lastly, views were designed to be shown differently based on the authenticated users' roles. This was done by using methods imported from the ASP.NET Core Identity library.

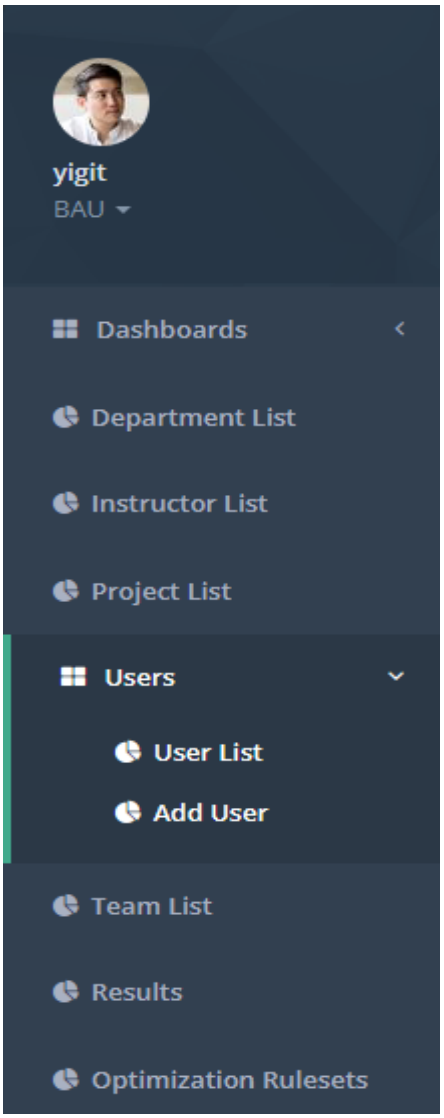


Figure 36. Admin Nav Bar

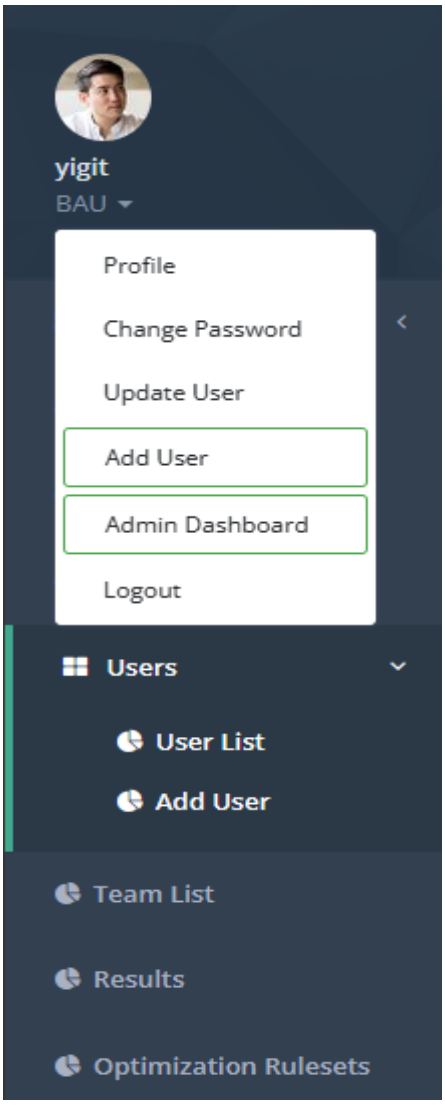


Figure 34. Admin Dropdown Nav Bar

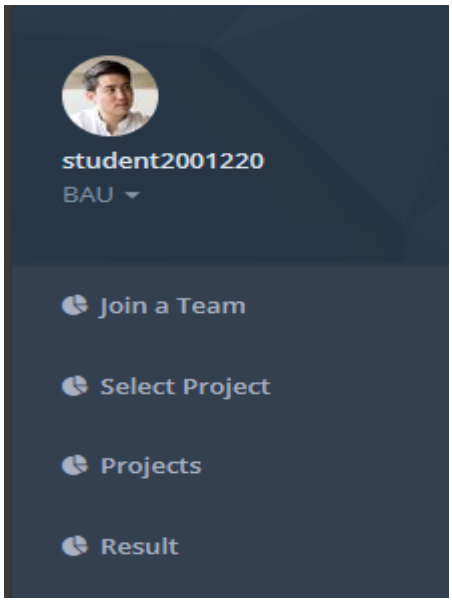


Figure 35. Student Nav Bar

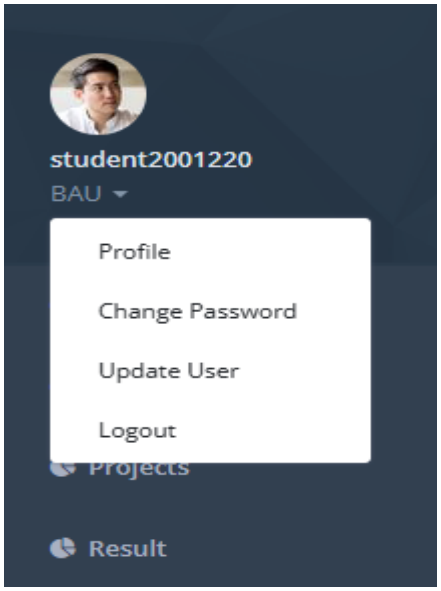


Figure 33. Student Dropdown Nav Bar

Two different navigation bars were implemented for two different user roles. If the authenticated user's role is Admin, Figure 34. And Figure 35. Is displayed. If the authenticated user's role is Basic(student), Figure 33. And Figure 36. is displayed. Users can navigate to the preferred pages using these navigation bars shown in figures.

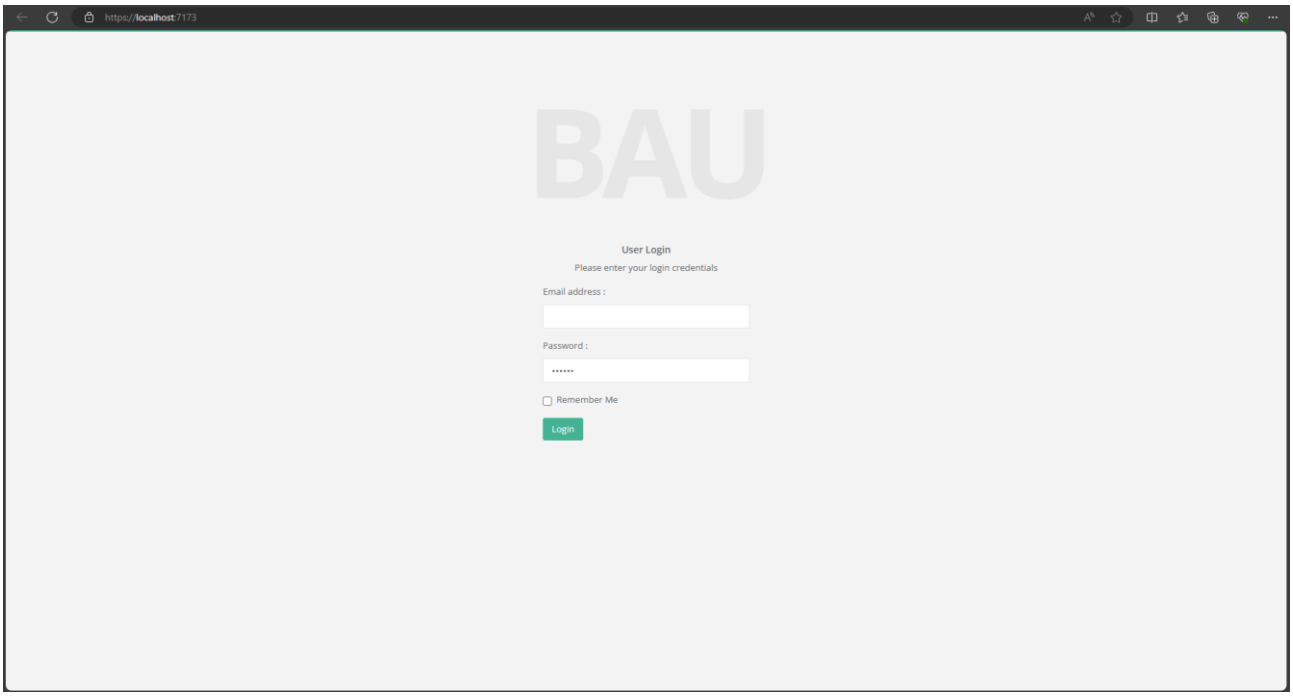


Figure 37. Login Page

Login page is implemented for users to sign in with their credentials as shown in Figure 37. Identity library is used for the authentication process of the users. If the authentication process is successful Users will be directed to pages set according to their roles. Otherwise, users will be displayed an error message saying email or password is wrong and a failed attempt count. Users who fail to authenticate 3 times will be given a 3-minute timeout.

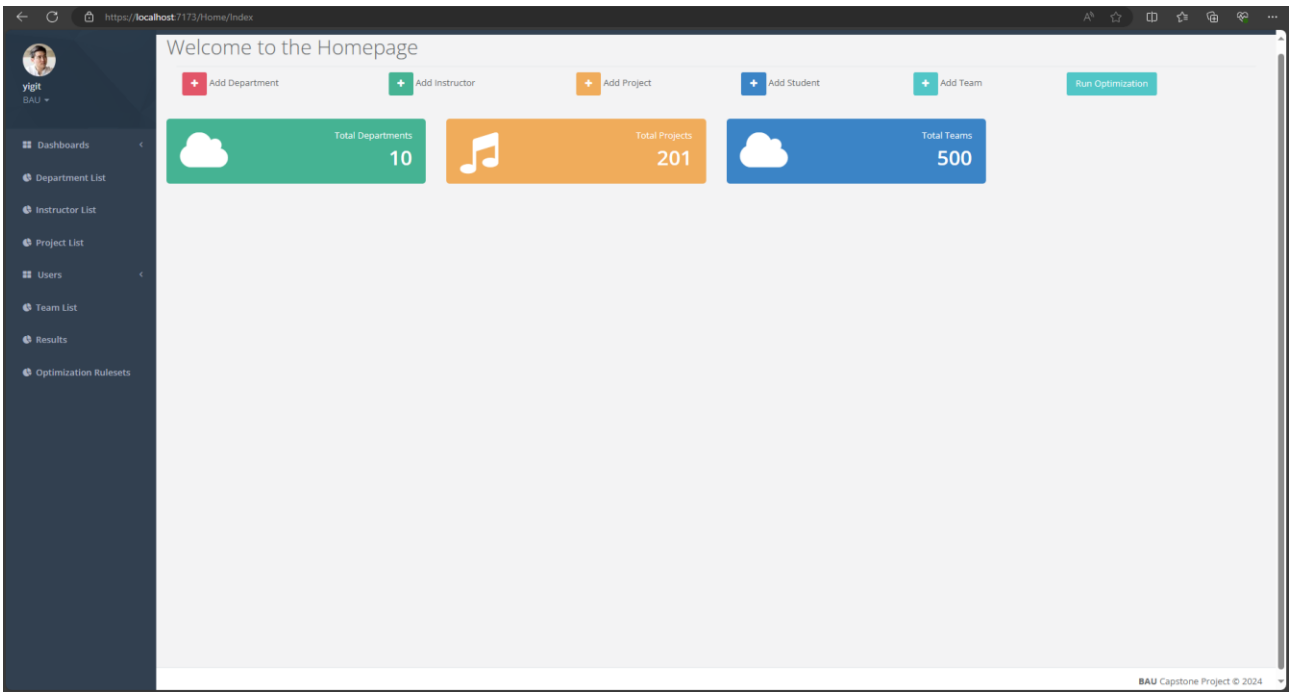


Figure 38. Admin Landing and Dashboard Page

If the authenticated user has the admin role, the landing page in Figure 38. is displayed. Admin dashboard is implemented with shortcuts for Add pages and widgets that show the total number of teams, projects and departments.

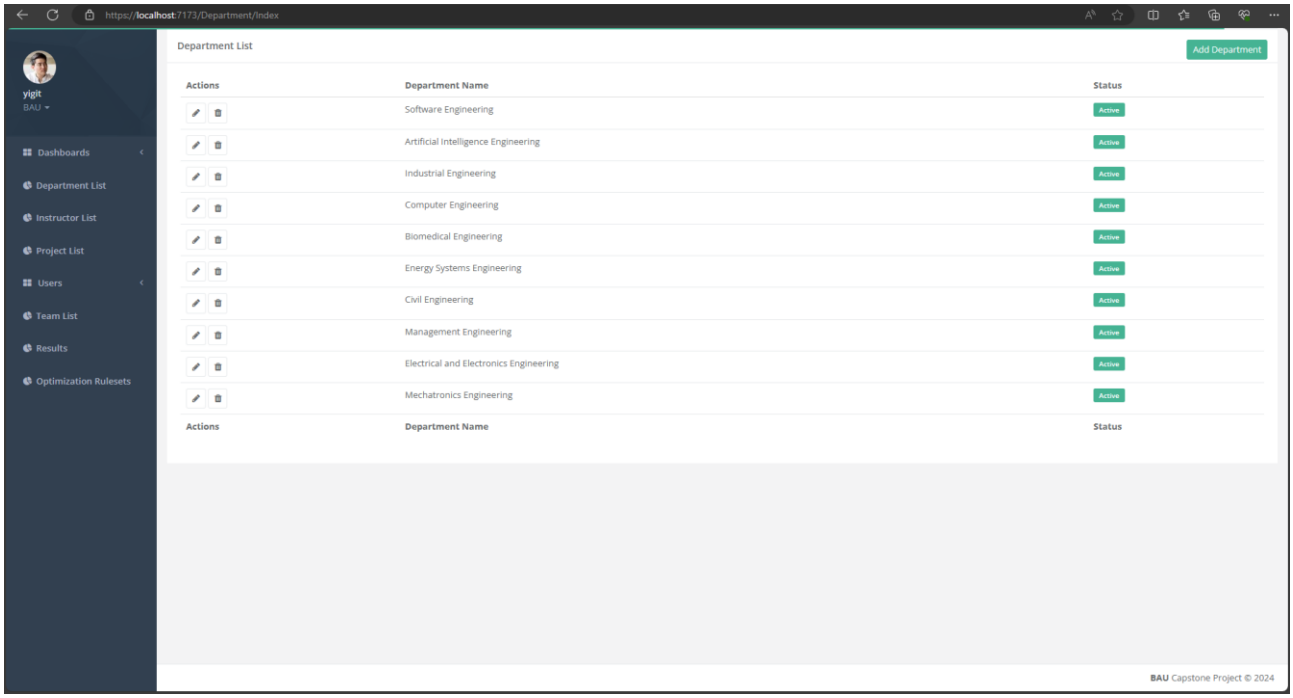


Figure 39. Department List Page

The Department List Page shown in Figure 39. is implemented for admins with an add department button and a table that shows the registered departments' name, status, and action buttons. Users can navigate to the update and add pages through buttons displayed.

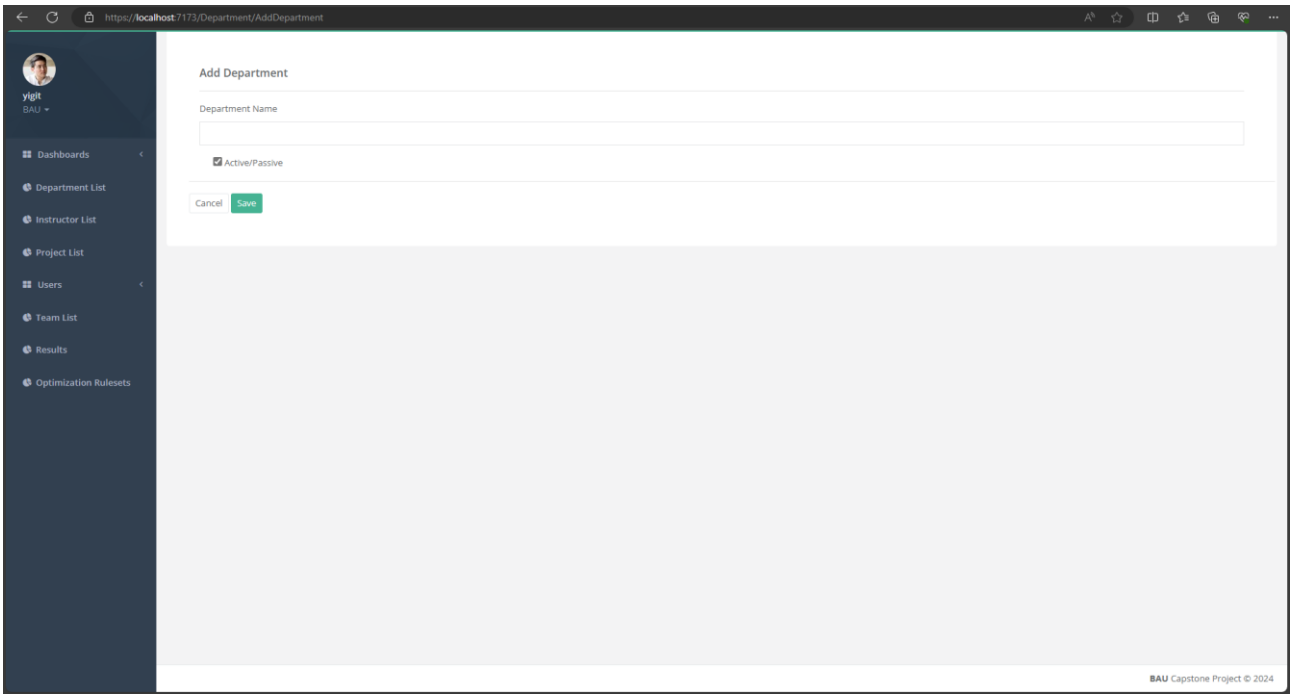


Figure 40. Add Department Page

From this page shown in Figure 40., admins can add new departments with desired names to the system. Department name field cannot be left empty.

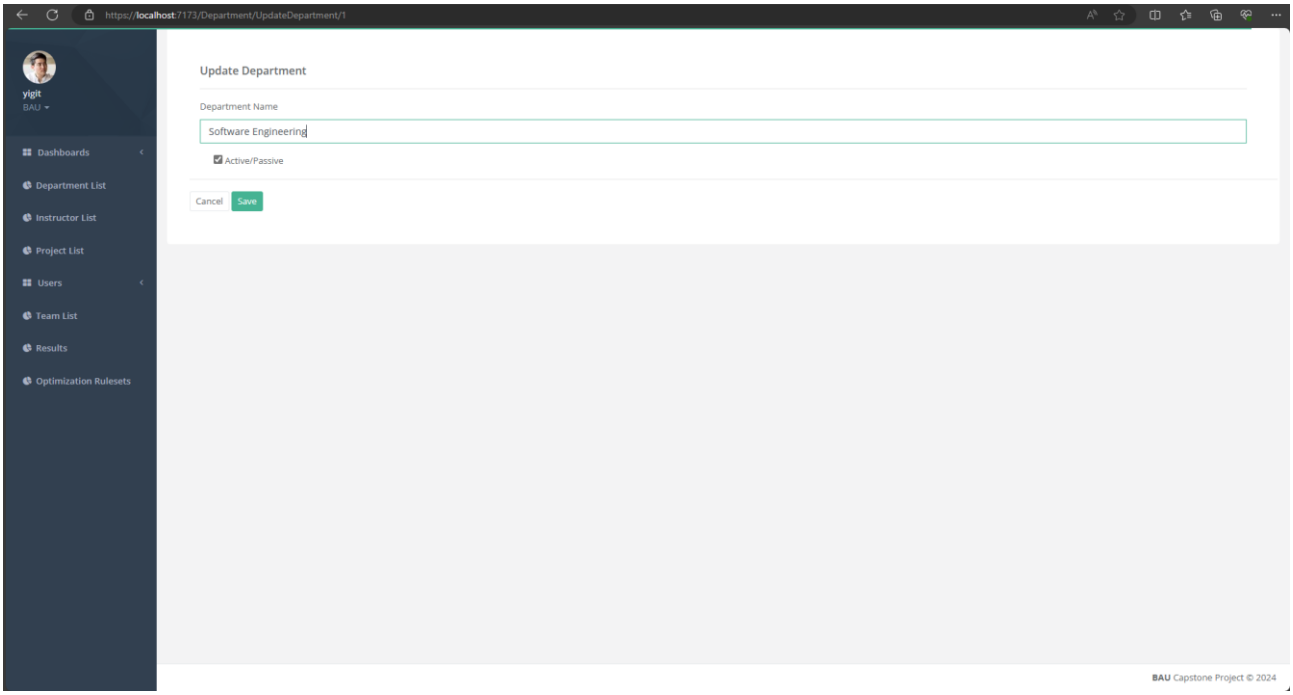


Figure 41. Update Department Page

As shown in Figure 41. This page allows admins to update the existing departments in the system. Selected department's name will be displayed to the user to change.

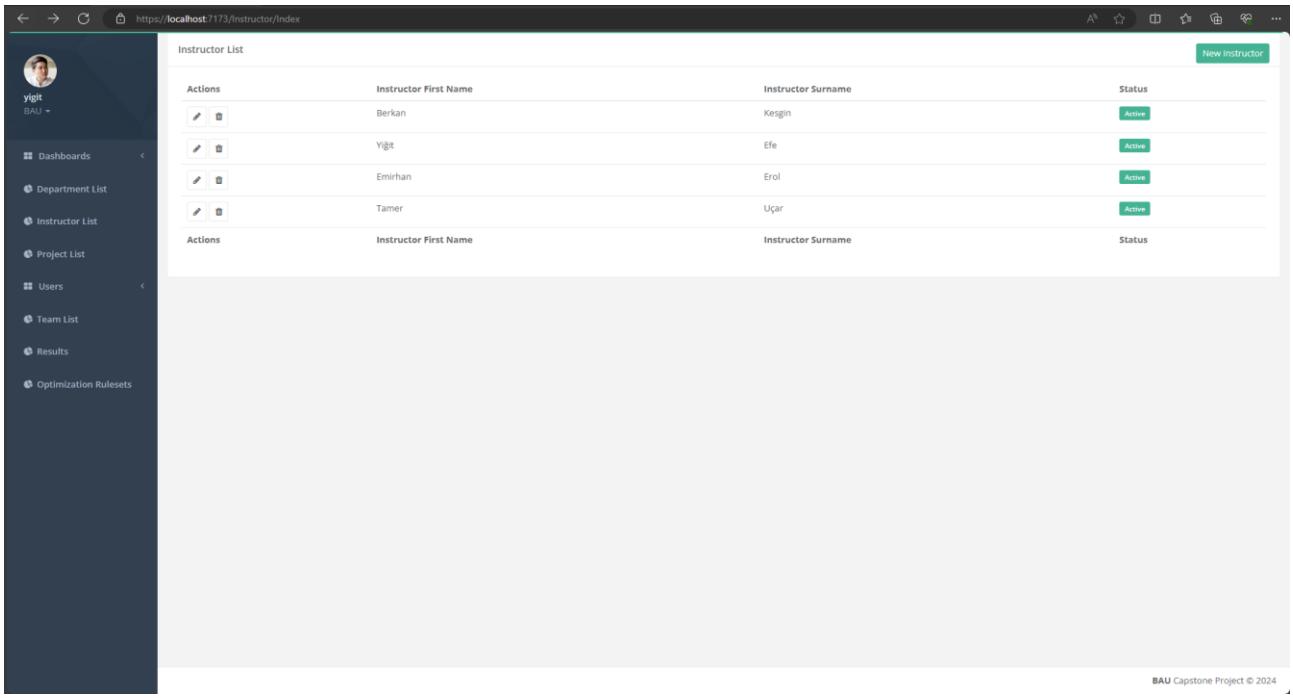


Figure 42. Instructor List Page

As shown in Figure 42. the Instructor List page is implemented for admins with an add Instructor button and a table that shows the registered Instructors with their first name, surname, and action.

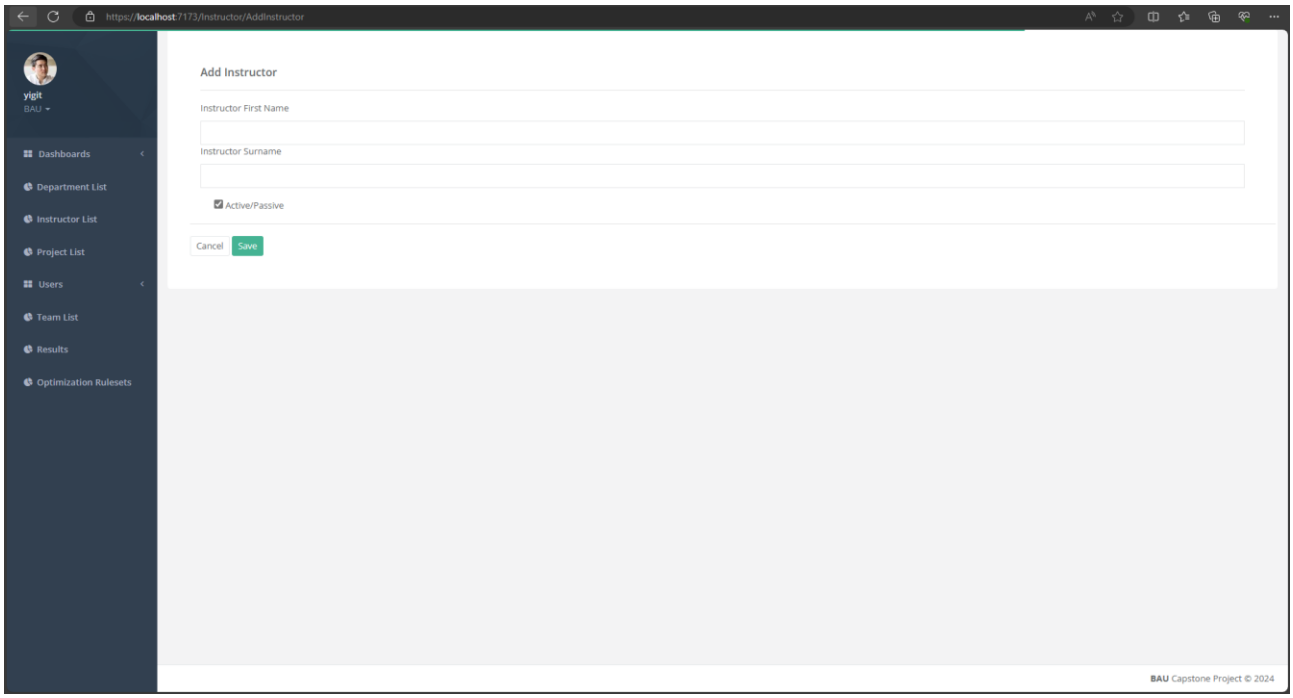


Figure 43. Add Instructor Page

From the page shown in Figure 43. admins can add a new instructor to the system by entering the name and surname of the instructor. Name and Surname fields cannot be empty.

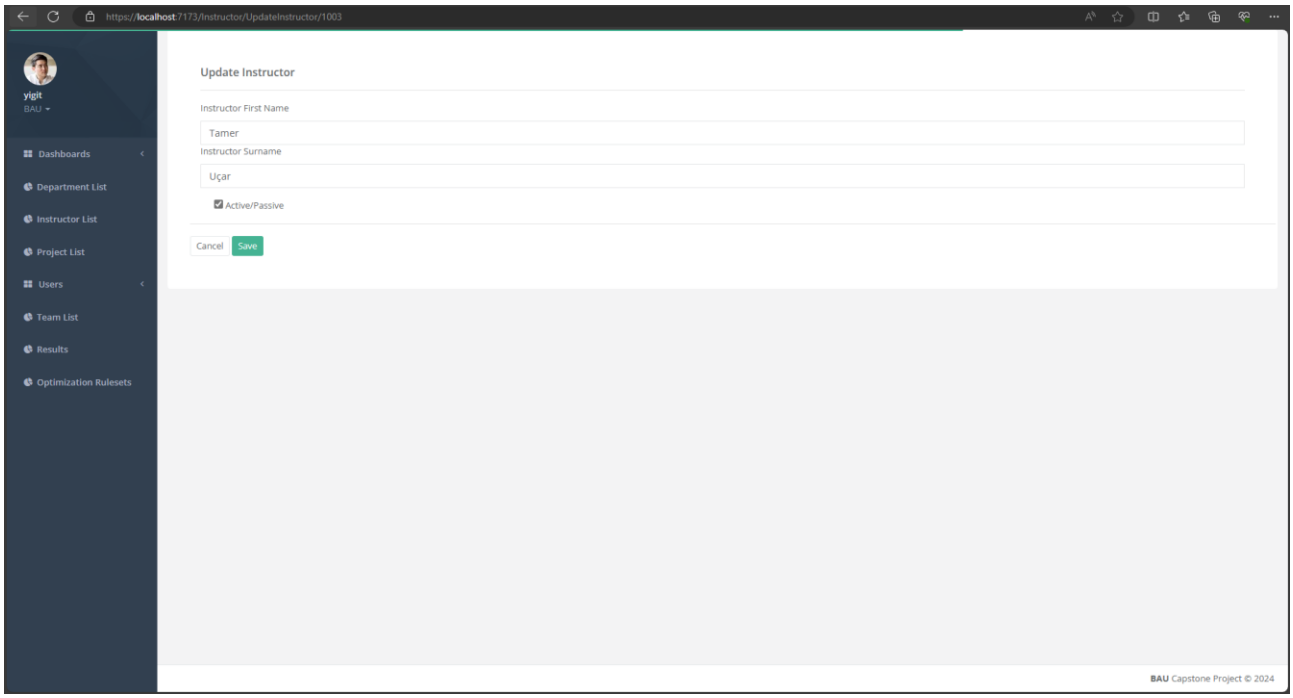


Figure 44. Update Instructor Page

As shown in Figure 44. this page allows admins to update the existing Instructors in the system. Selected Instructor's information are displayed to the user.

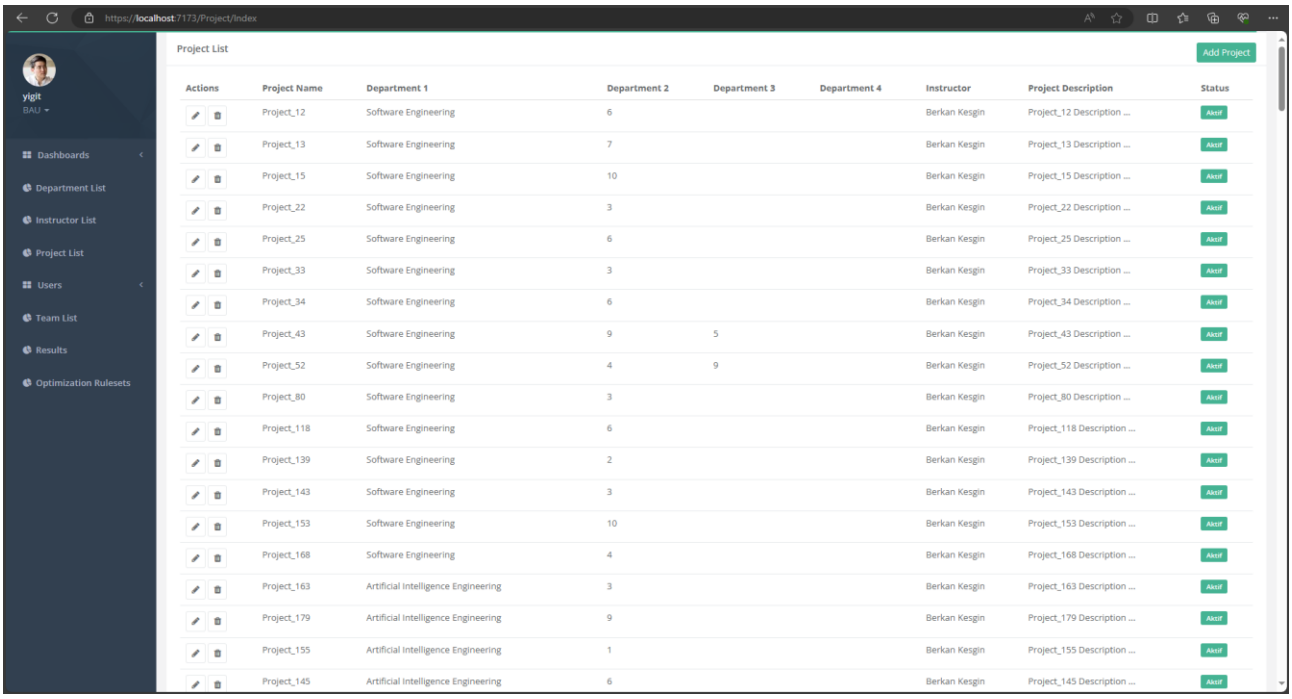


Figure 45. Project List Page

At this page in Figure 45. all projects consisting in the DB can be seen by the admin with their respective names, departments, instructors, descriptions, action and add buttons.

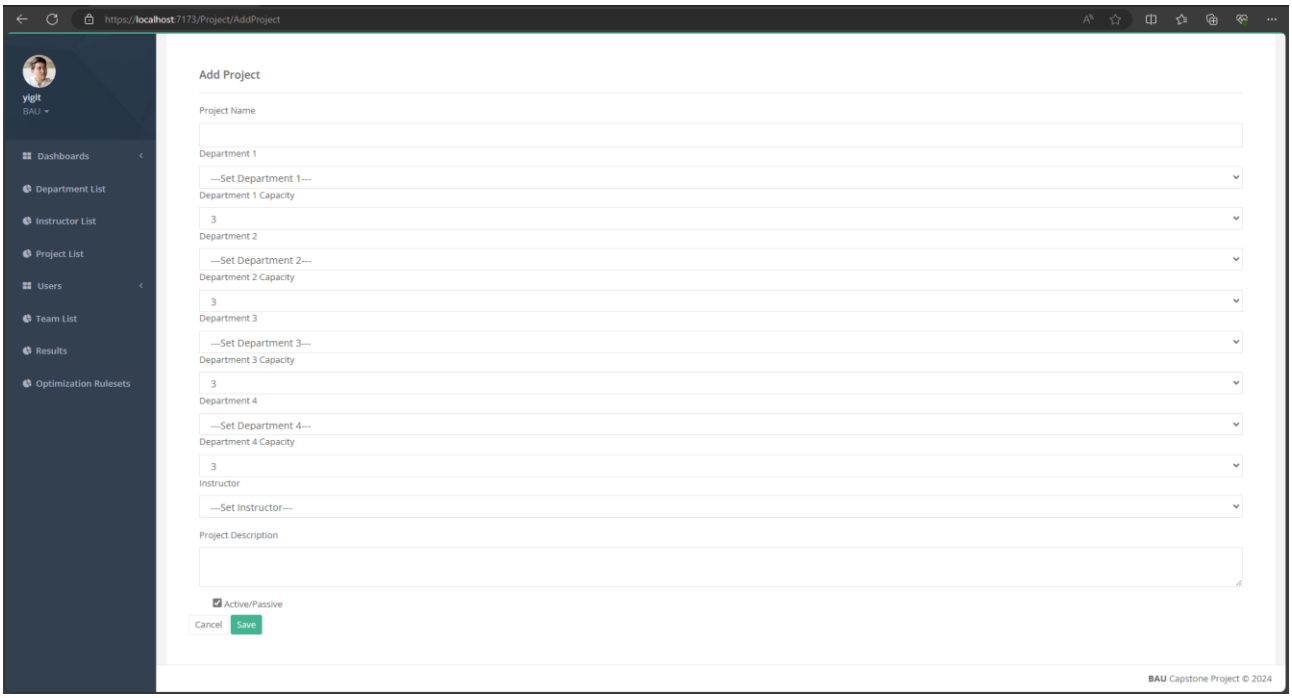


Figure 46. Add Project Page

The Add Project Page in Figure 46. allows admins to create a new project on the system. Admins can set a project name, department IDs and capacities related to the departments, a project instructor and a description.

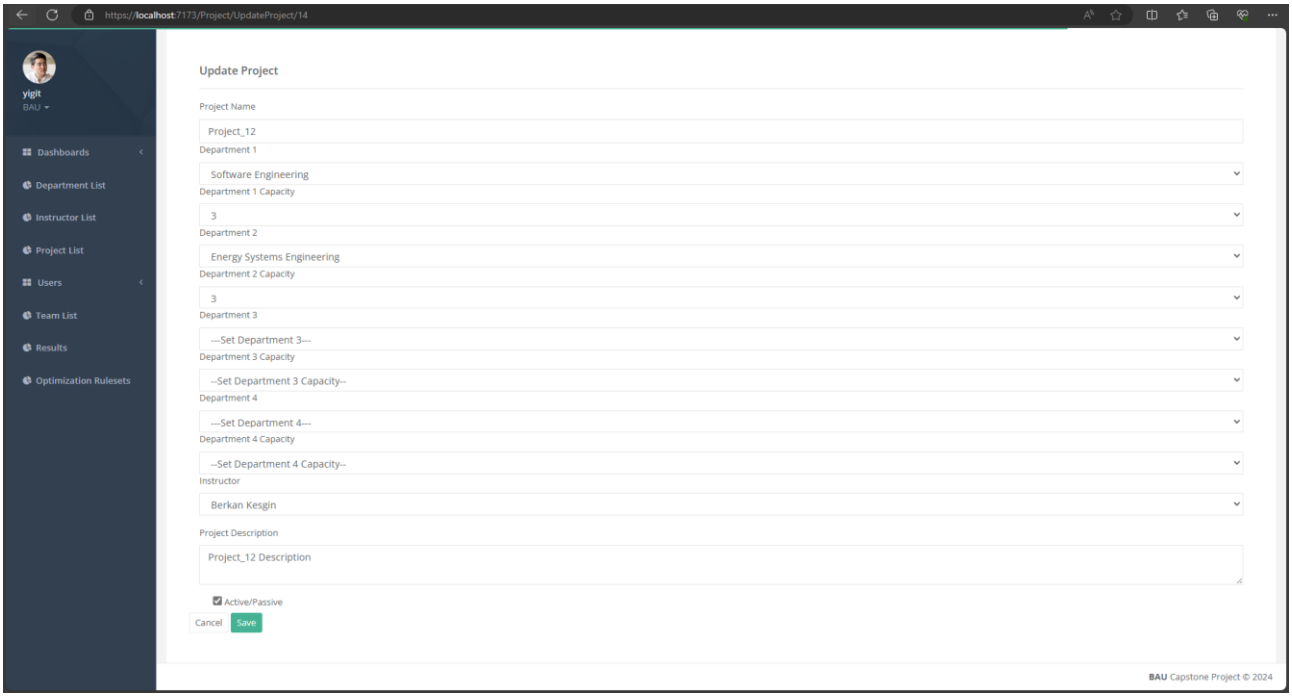


Figure 47. Update Project Page

The page shown in Figure 47. contains active information about the relevant project to be updated, the name of the project, the departments to which the project is affiliated, the student capacity that can be received from those departments, the instructor's name and the project description can be changed by the admin from this page.

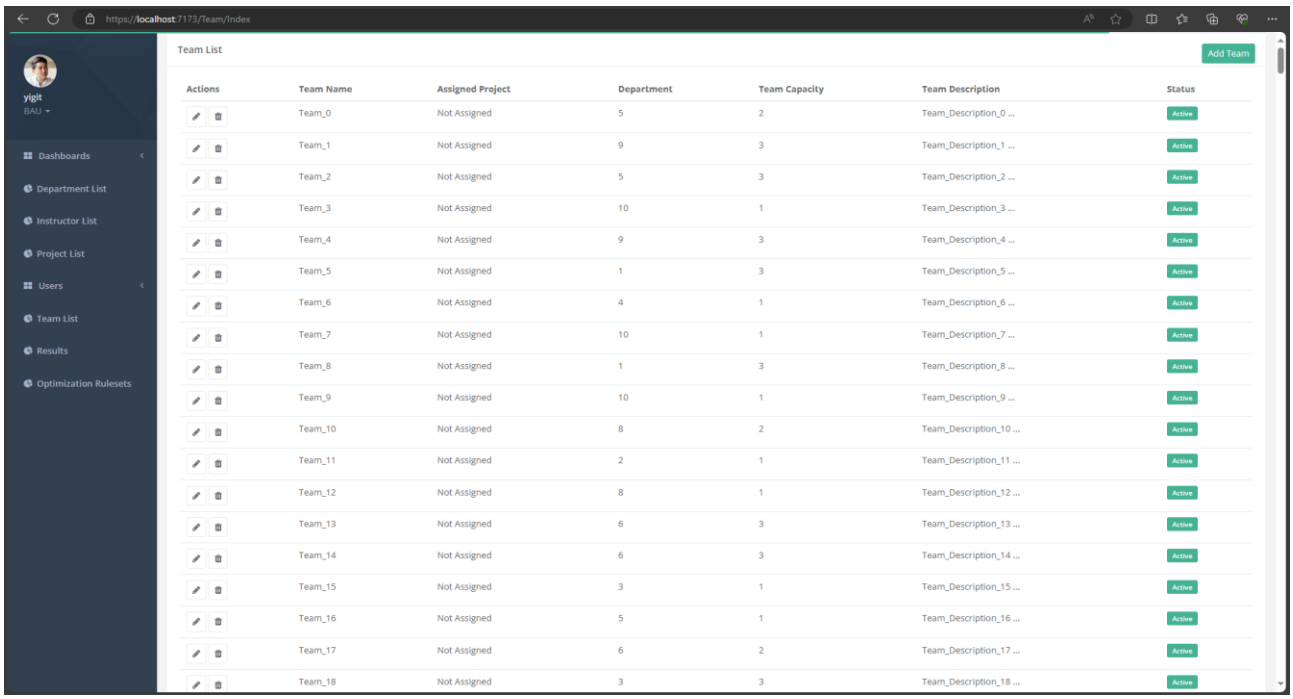


Figure 48. Team List Page

In the Team List Page shown at Figure 48. information about all the registered teams can be found. Team names, assigned projects, department, capacity and instructor is shown on this page.

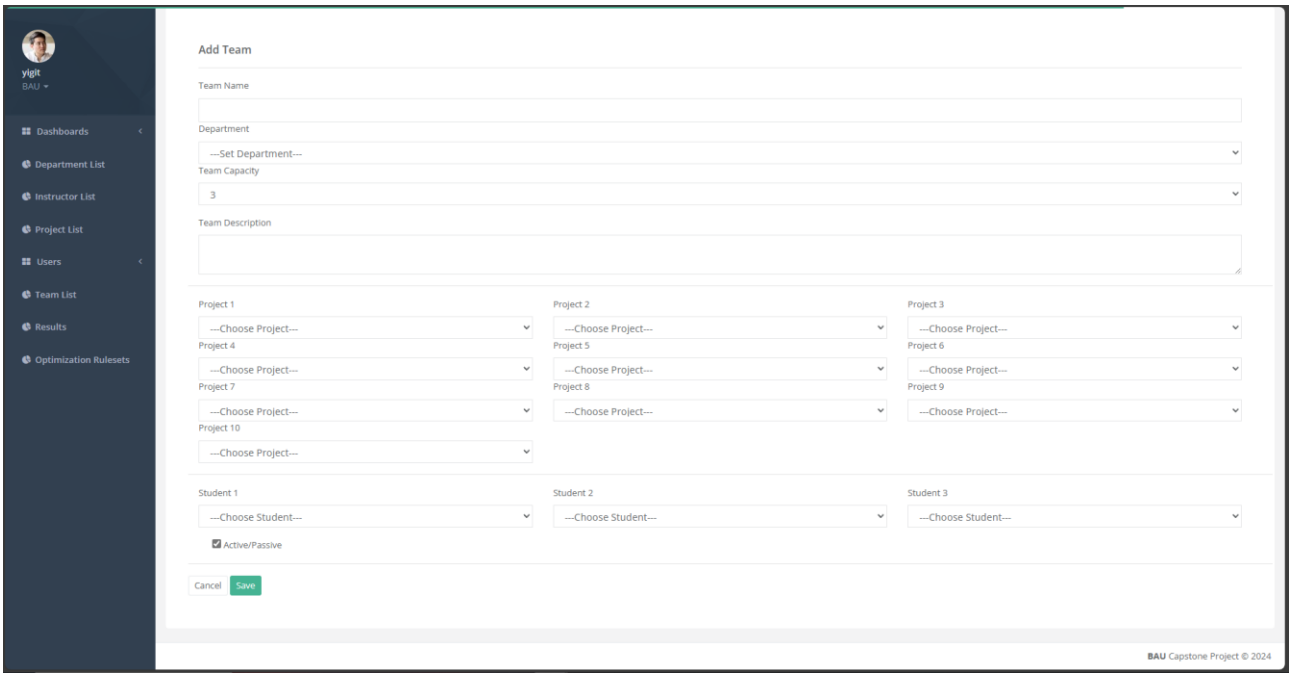


Figure 49. Add Team Page

As shown in Figure 49. admins can create a new team by selecting the name of the new team they want to create, the name of the department it is related to, the maximum capacity of the team up to 3 people, the team description, their project preferences and their own name.

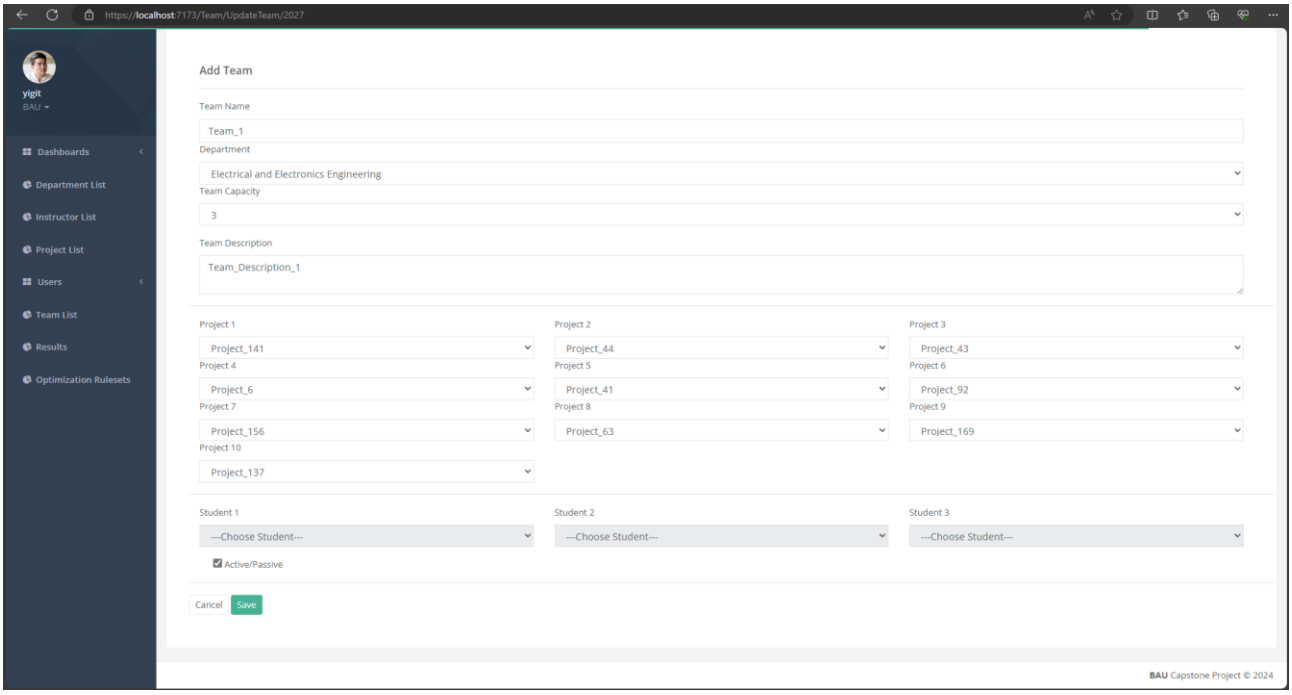


Figure 50. Update Team Page

As shown in figure 50. in order to change the information of an existing team on the update team page, the team's name, relevant department, team capacity, team description and project preferences of the team selected displayed to the admin to change.

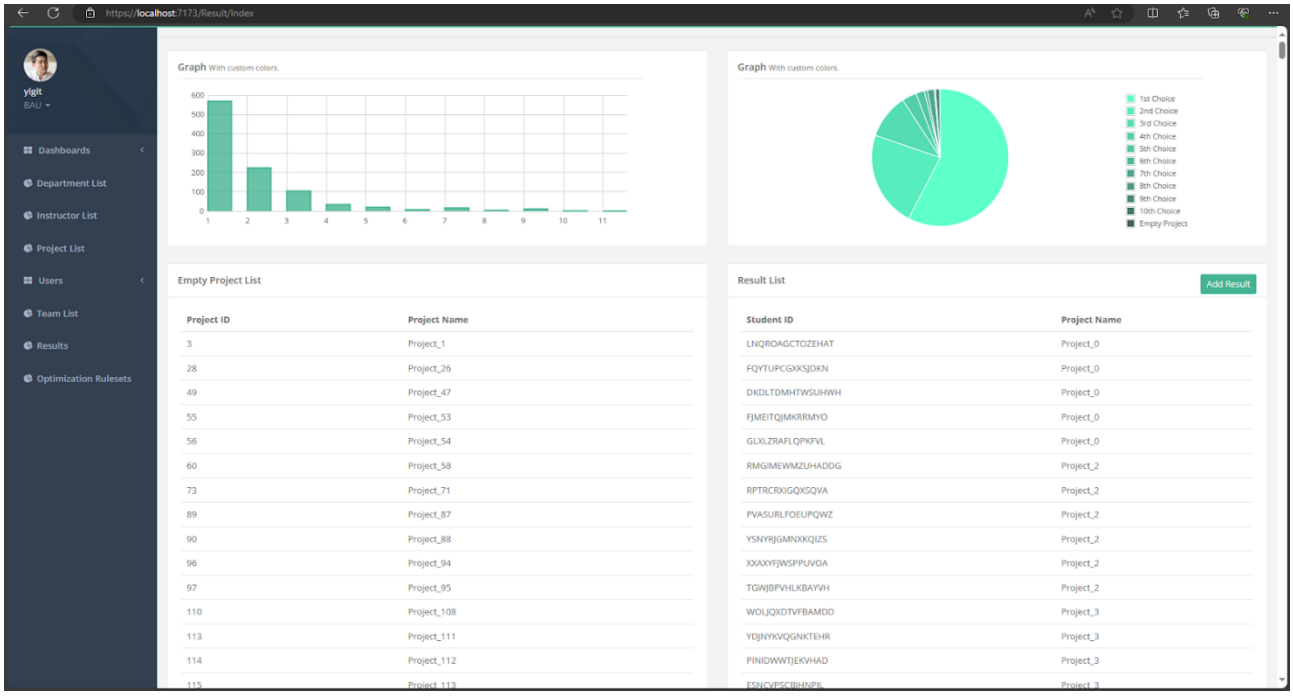


Figure 51. Admin Result Page

Figure 51. shows the Admin Result Page. On the right side of the page, student IDs are lined with their assigned project names. Graphs are added as per request from the Industrial team to visualize the assignment results.

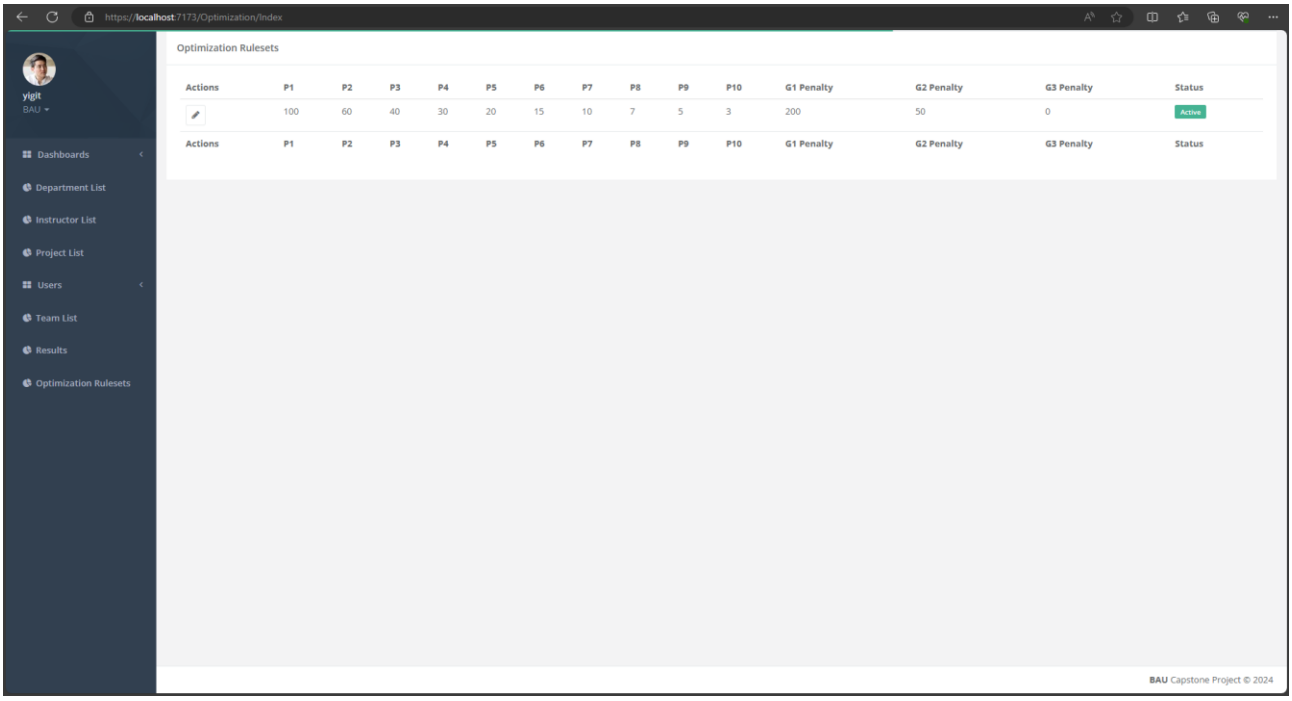


Figure 52. Optimization Rulesets Page

On this page shown on Figure 52., optimization rulesets can be found. Optimization reward and penalty points are shown in the table with edit button.

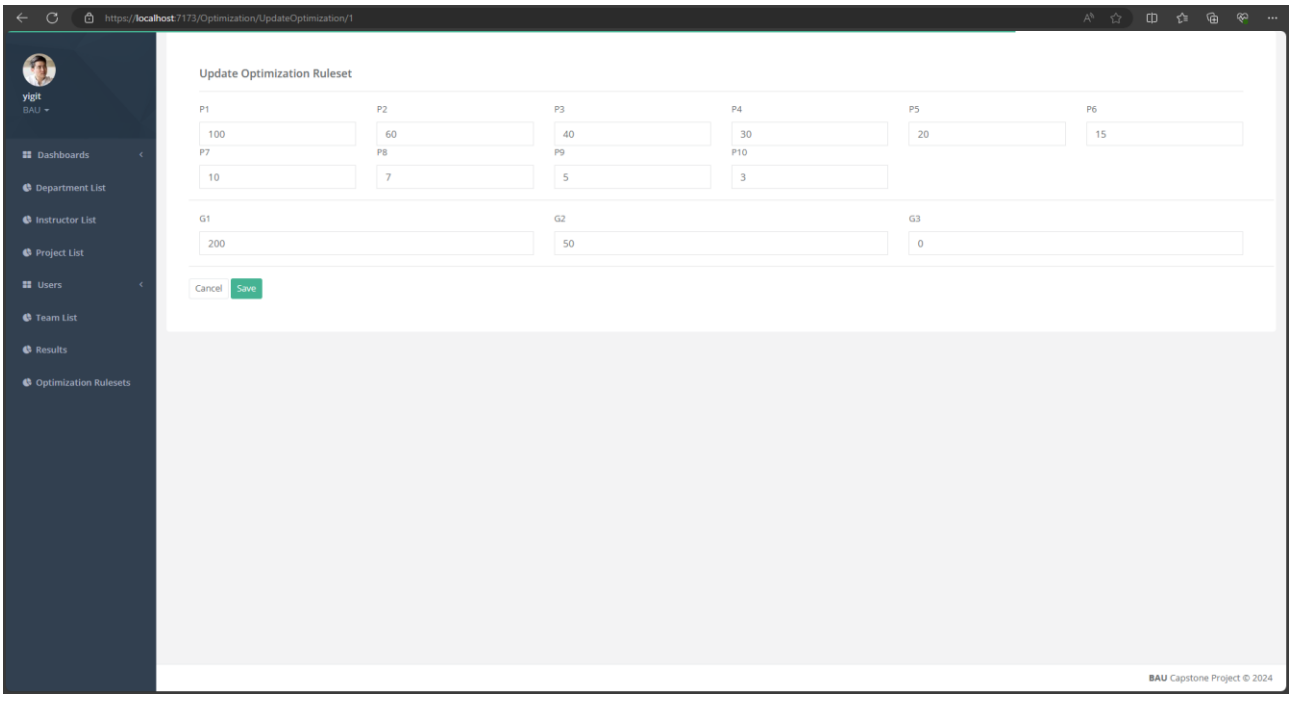


Figure 53. Optimization Ruleset Edit Page

As per request from the industrial engineering team, a page for admins to change Optimization Engine's reward and penalty points implemented to the system. Points can be modified by the admins on the page shown on Figure 53.

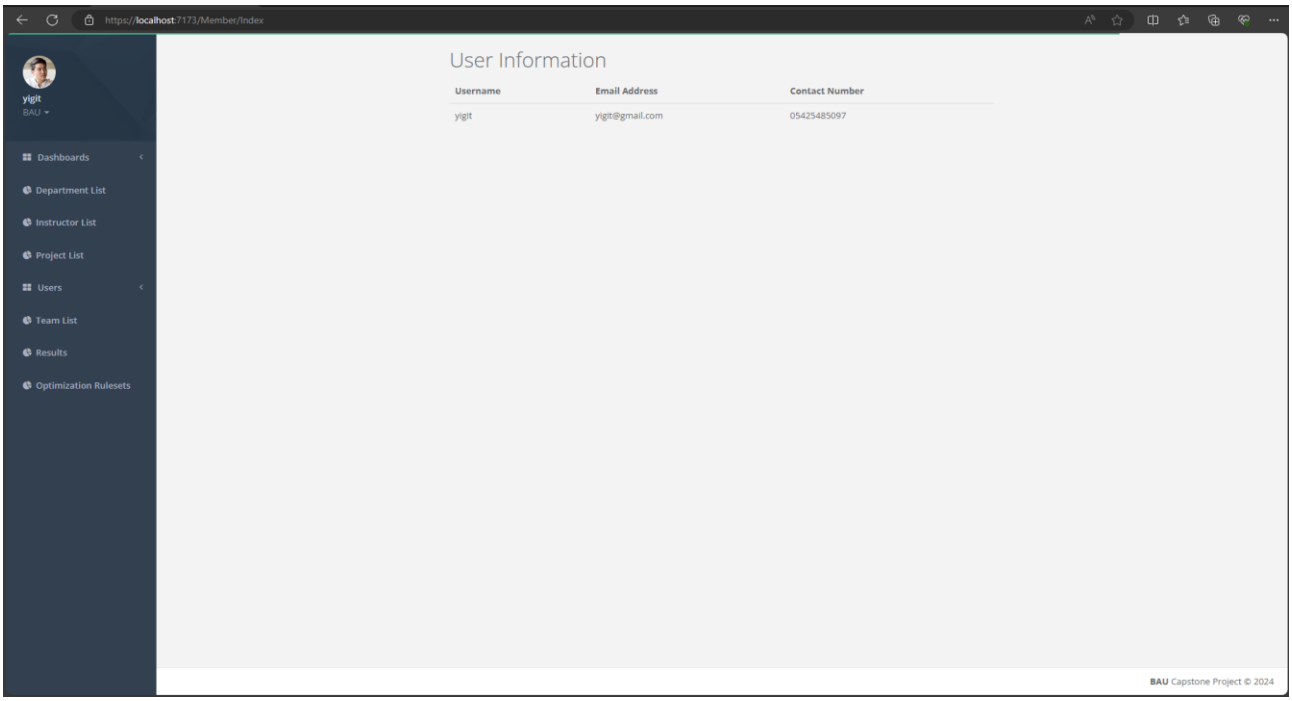


Figure 54. Profile Page

As shown in the figure 54. user information such as username, email address, and contact number related to the user account can be seen in this page.

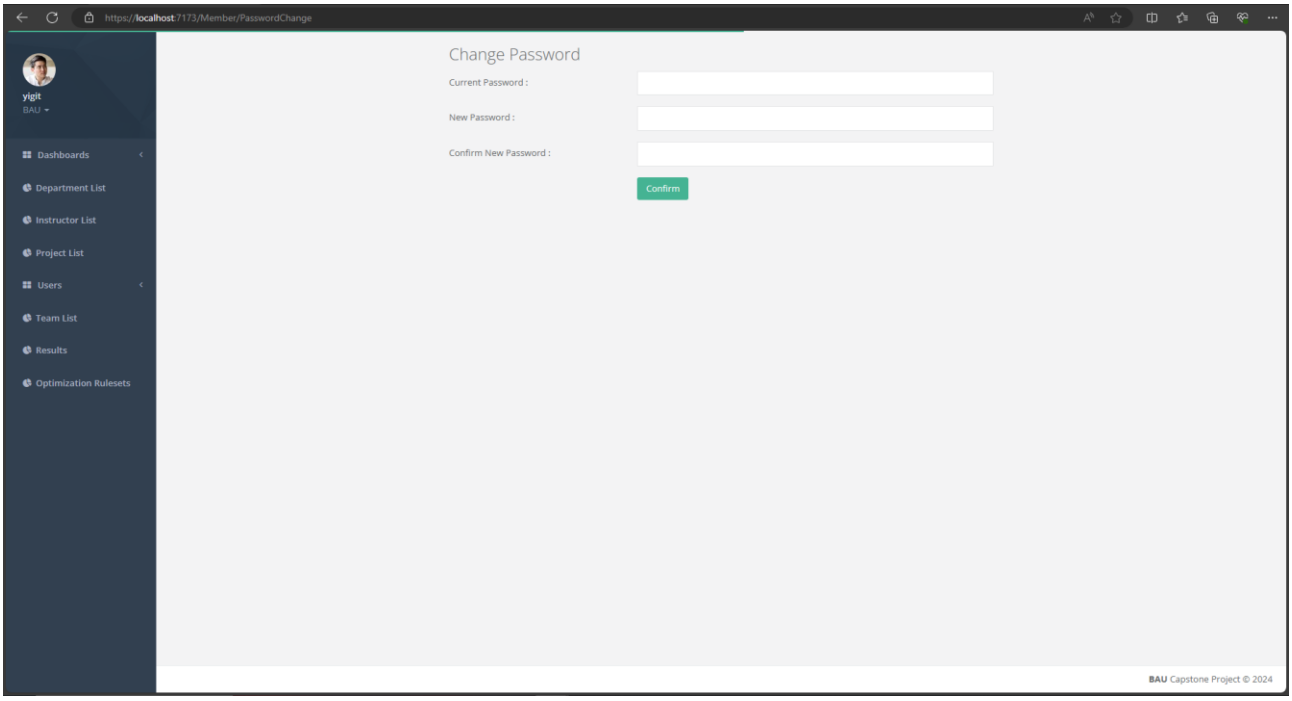


Figure 55. Change Password Page

From the page shown in figure 55. users can change their password by entering their current password and entering the new password they want to use.

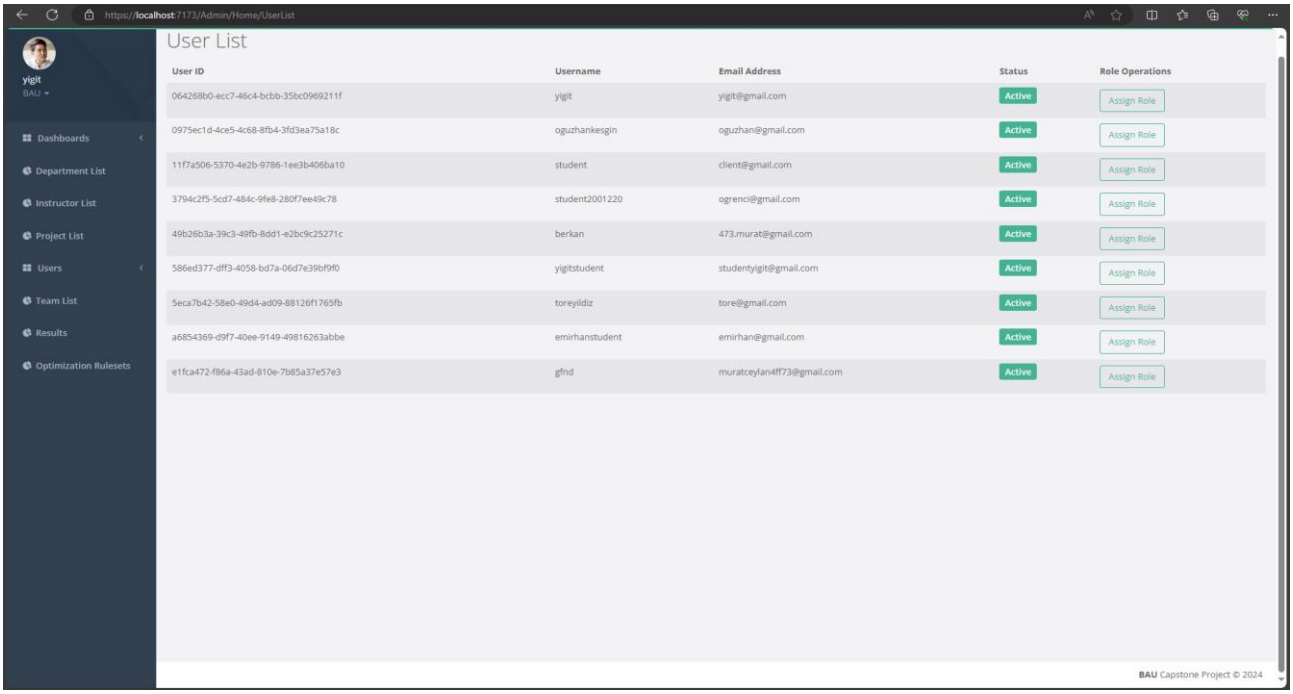


Figure 56. User List Page

On this page as shown in figure 56. User ID, username and e-mail addresses of active users can be displayed and role assignments can be made by admins.

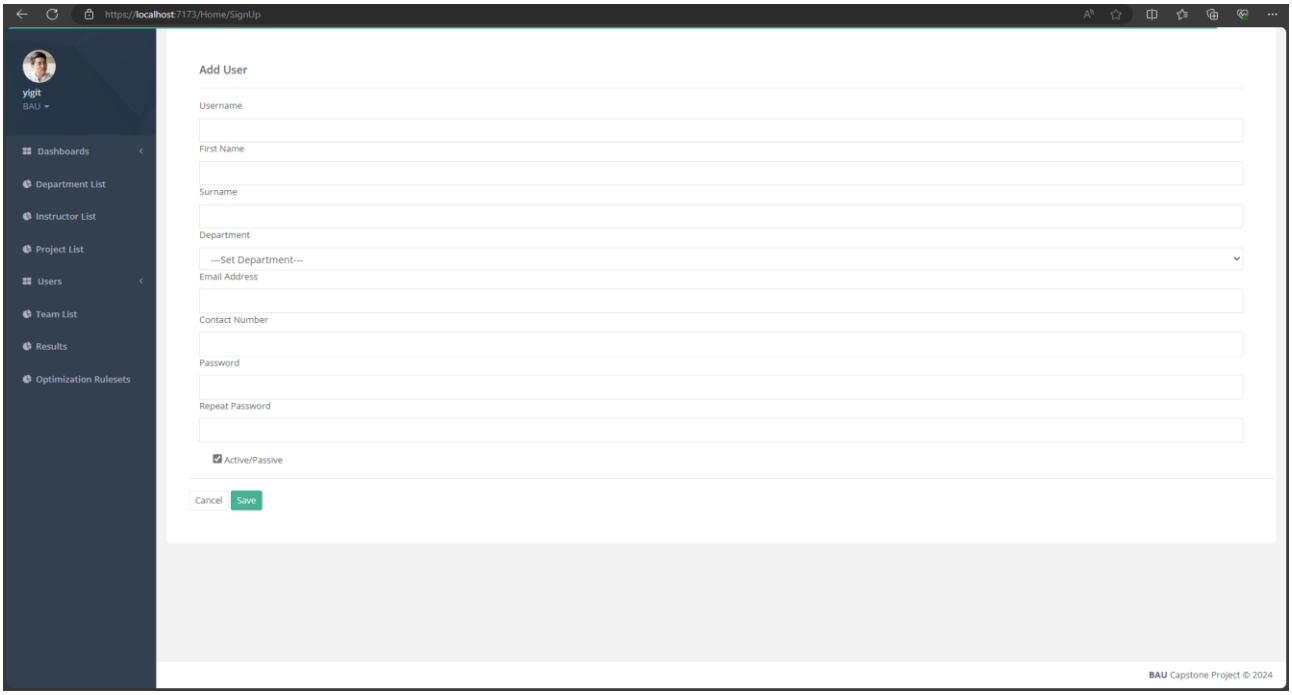


Figure 57. Add User Page

On this page shown in Figure 57. admins can add a new user to the system, they can enter the username, first name, surname, department, email address, contact number, password information of the user to be added and click the save button to save it.

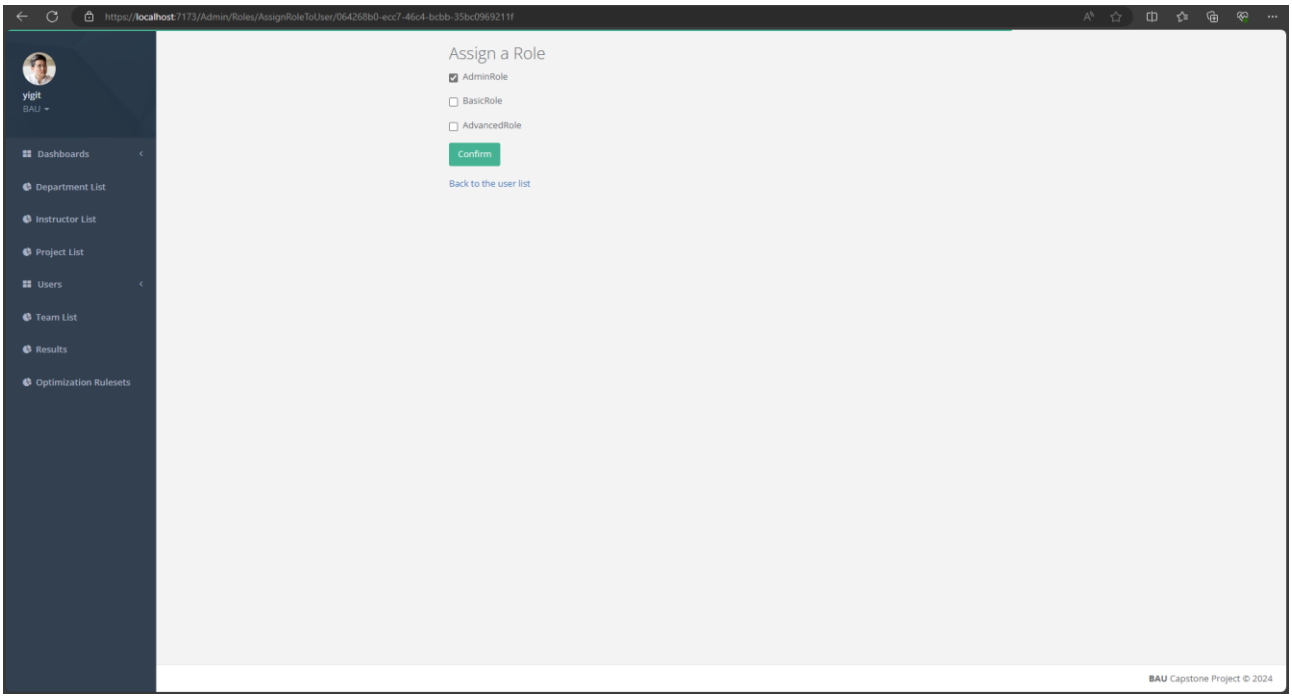


Figure 58. Assign Role Page

On the Assign role page shown in Figure 58. Admin can assign a role to the selected user. AdminRole for the admins, BasicRole for students.

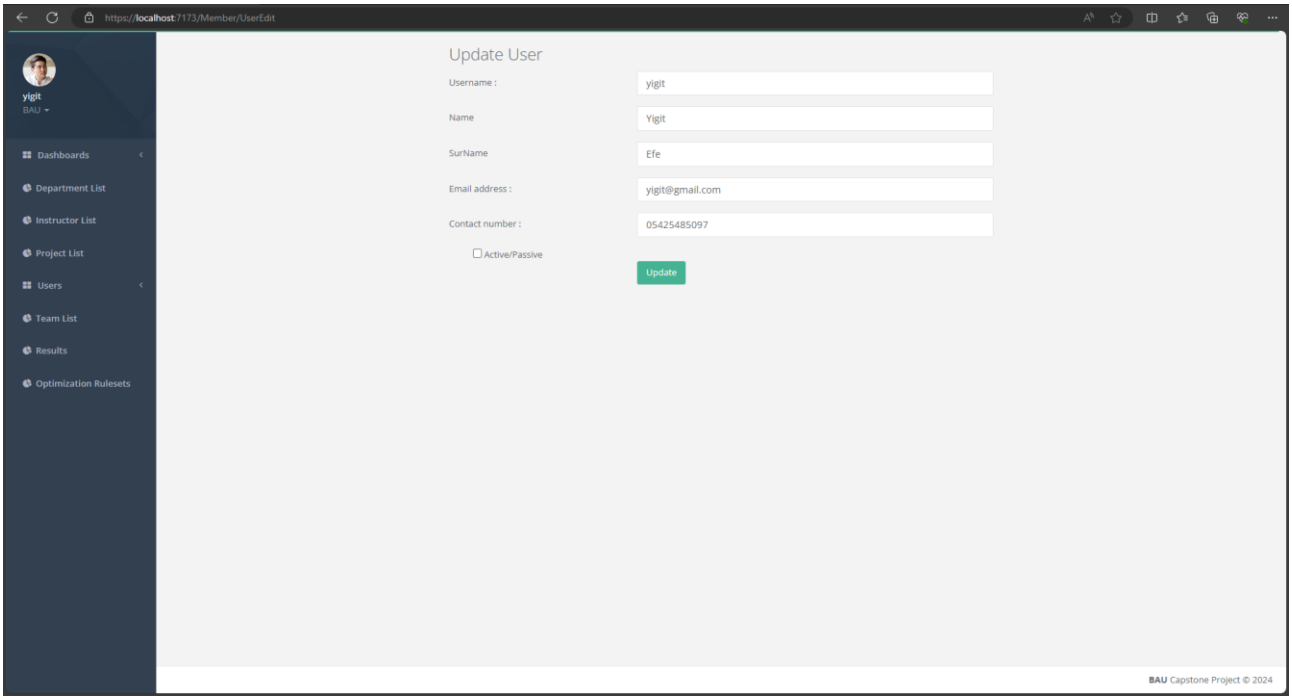


Figure 59. Update User Page

Figure 59. presents a page where users can change their username, name, surname, email and contact number and confirm it by pressing the "update" button.

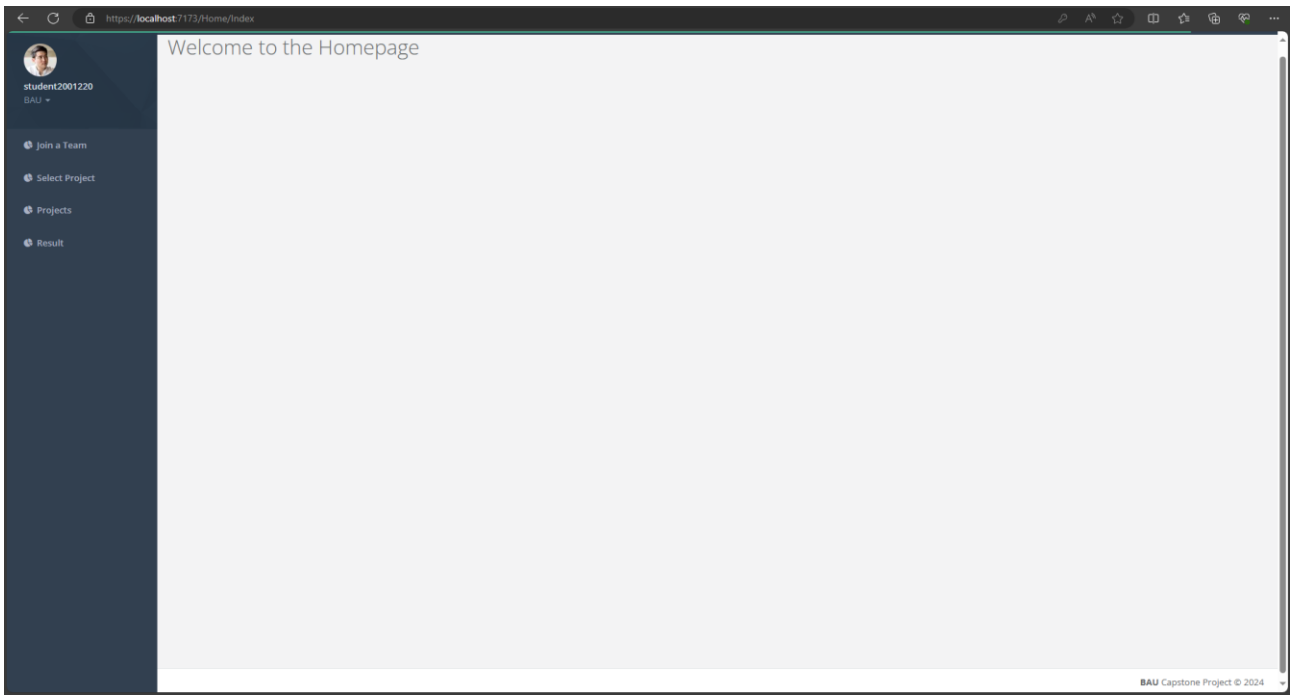


Figure 60. Student Landing Page

After signing in from Figure 37. Login page. If the authenticated user is a student, they will be redirected to the Page shown in Figure 60.

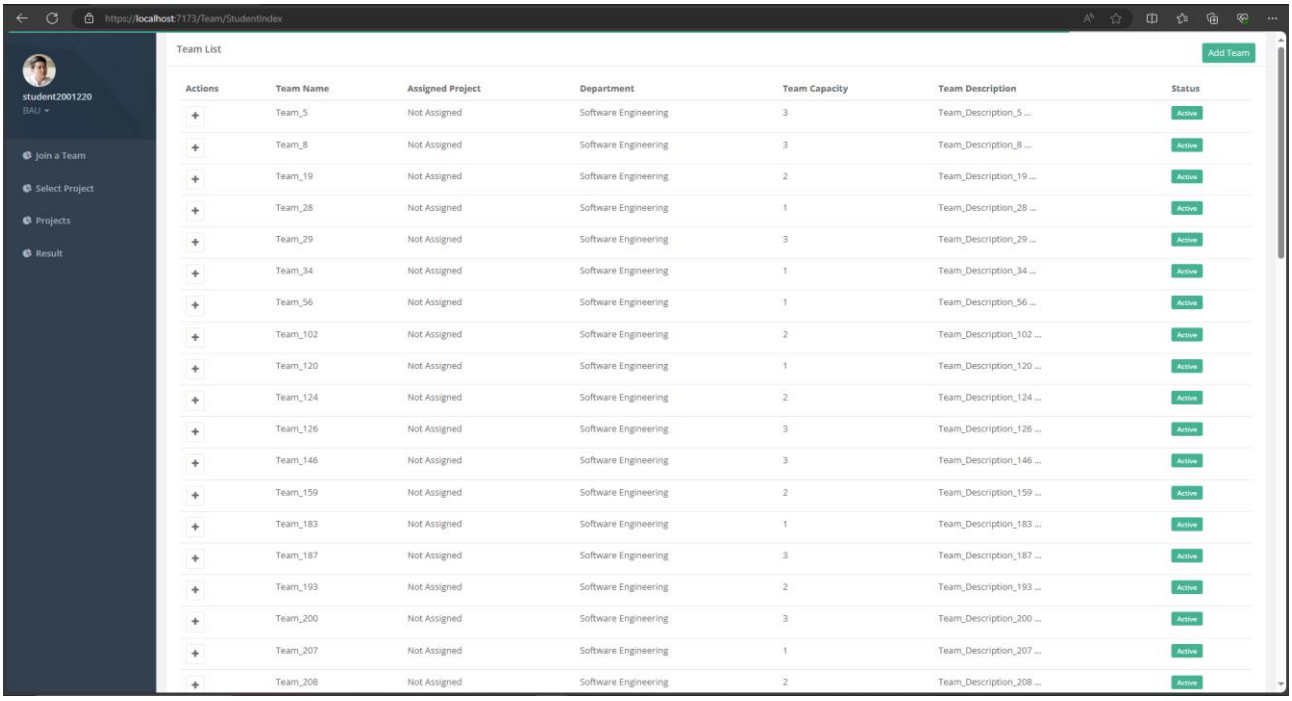


Figure 61. Student Team List Page

As shown in figure 61. students can display the available teams in their departments and also navigate to Join Team page by clicking on the action button. Team name, the project they are assigned to, the department of the team, the capacity of the team, the team description of the teams are displayed. If they wish students can create their own team by clicking the add team button.

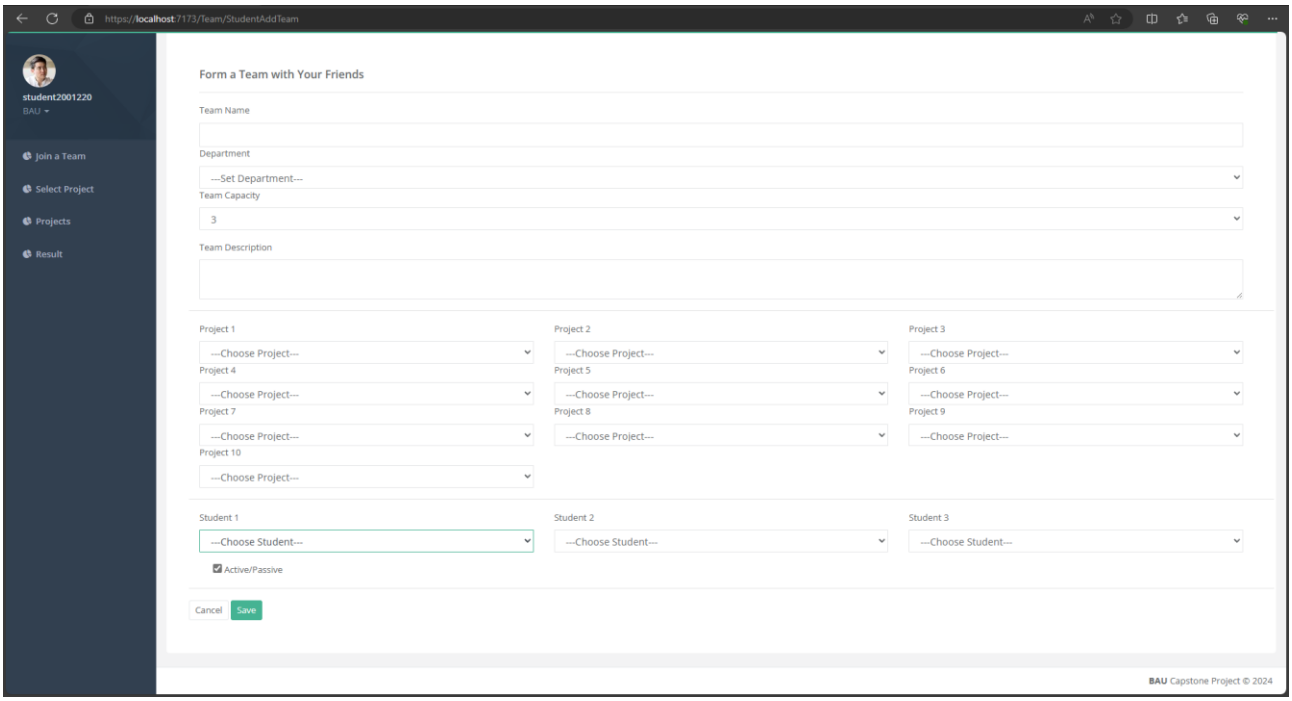


Figure 62. Student Add Team Page

From the page shown in Figure 62. students can create a team by writing their team names, selecting their departments from the dropdown, selecting their team capacity (maximum 3), writing a description of their team, and choosing their own names from the dropdowns where student names are written.

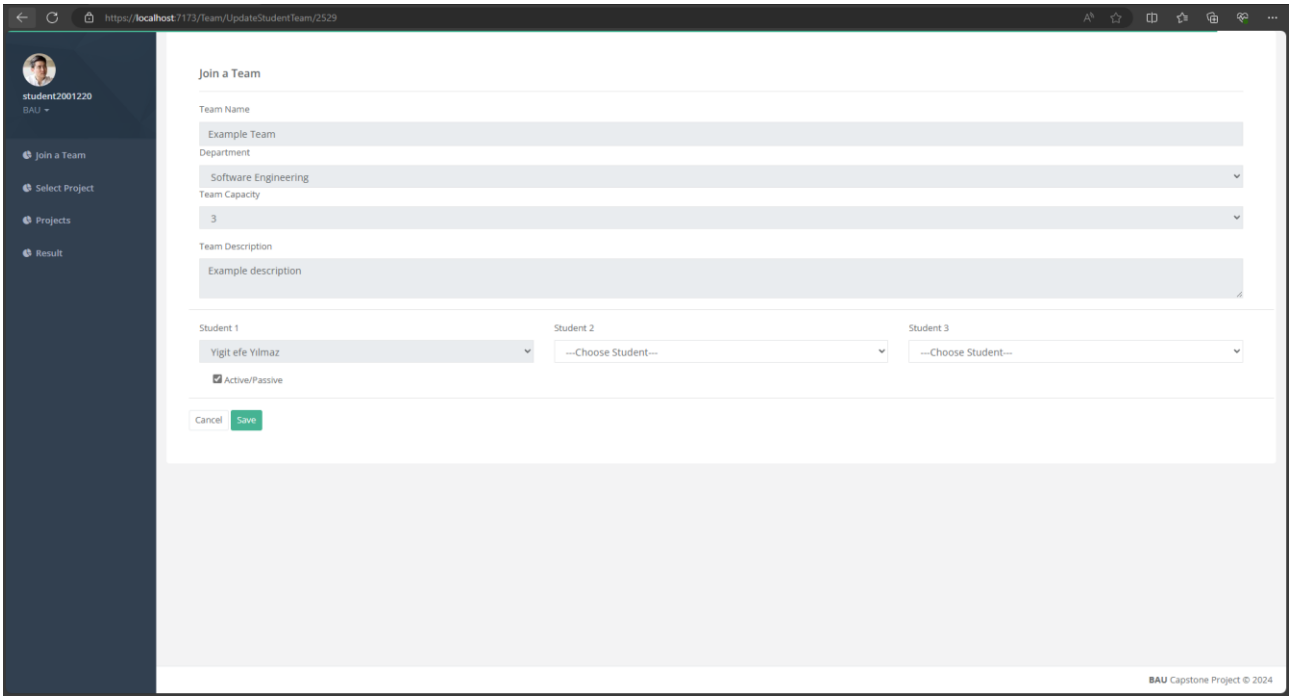


Figure 63. Student Join Team Page

Students can view and select established teams belonging to their department and join one by choosing their names from the empty dropdown list. In this page shown in Figure 63., students cannot

change the team’s attributes, they only can join to the team.

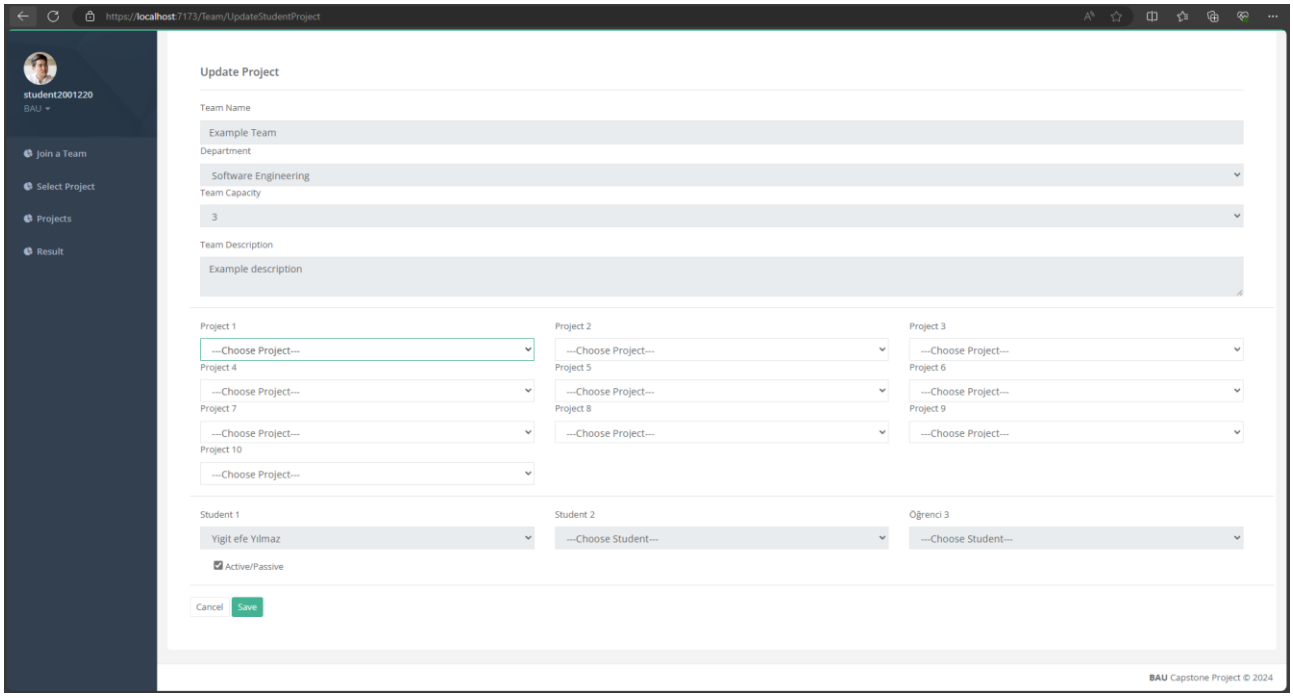


Figure 64. Student Project Selection Page

If authenticated students have already joined a team, they can choose projects on behalf of their team from the dropdowns displayed on the page shown in Figure 64., they can only choose the projects from their own department. If they do not have a team, they will be directed to the team creation page in Figure 62.

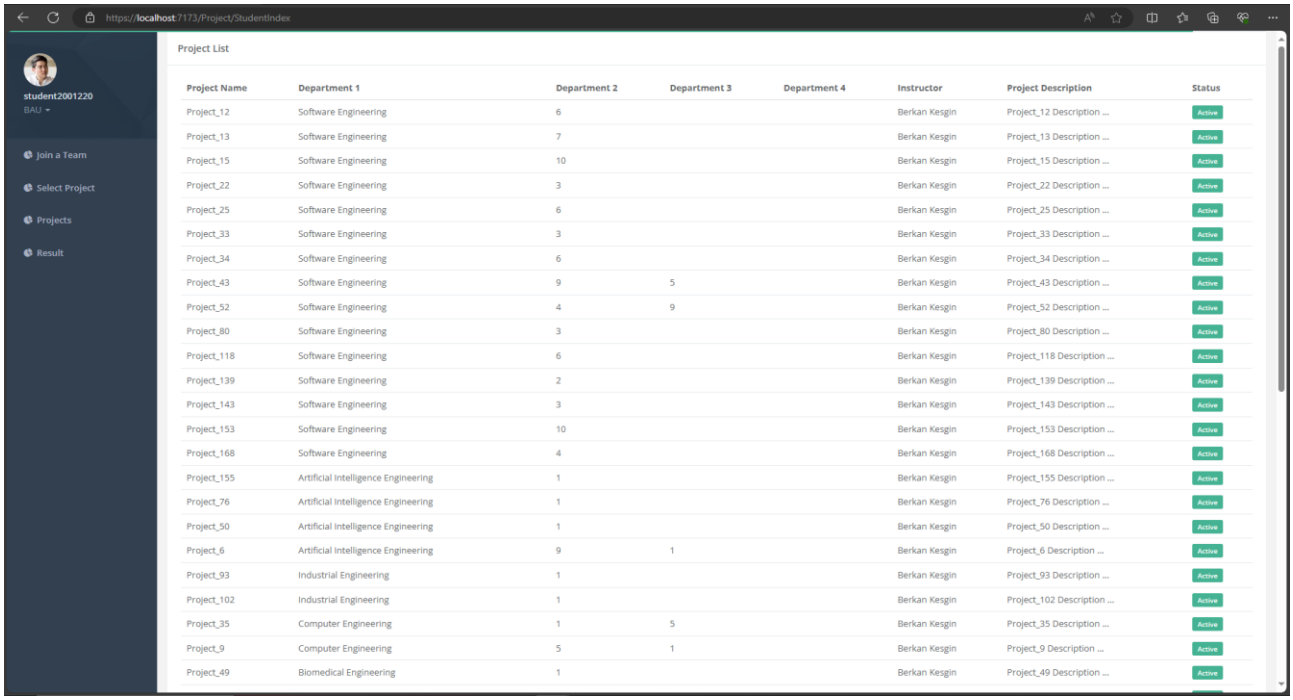


Figure 65. Student Project List Page

From the page shown in Figure 65. authenticated students can display the projects related to their departments. Projects' name, department, instructor name and description are displayed to the students in this page.

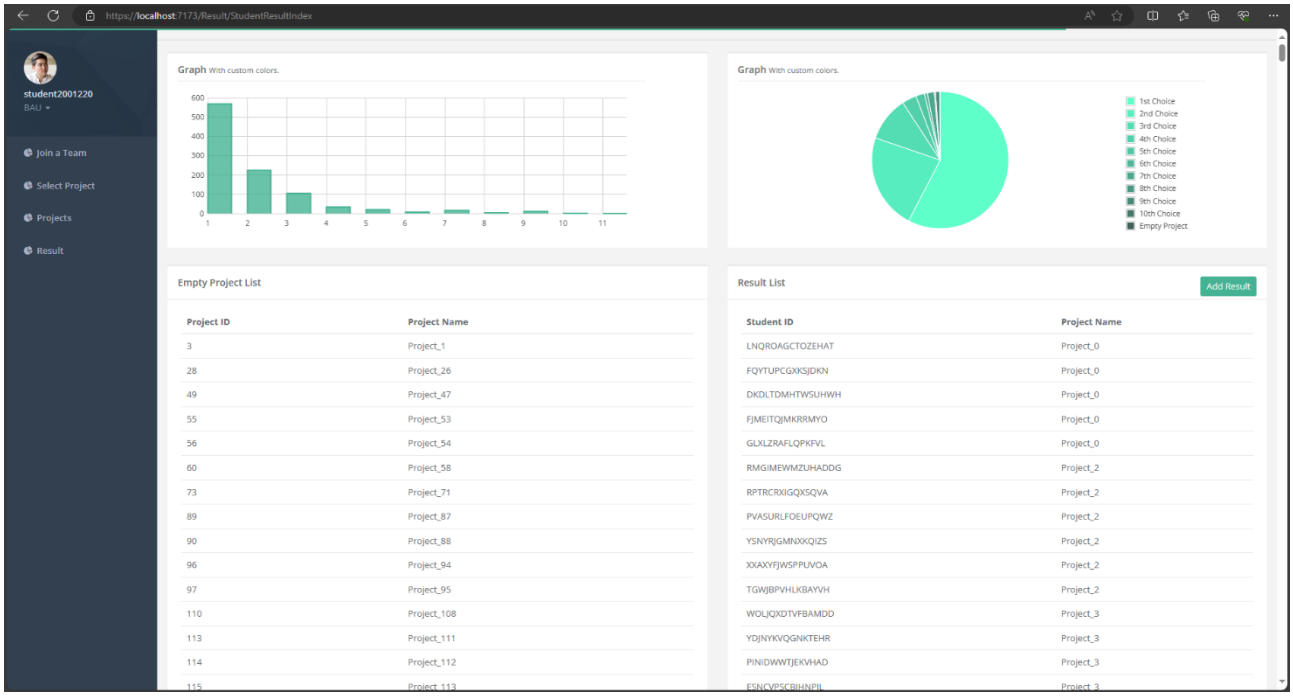


Figure 66. Student Result Page

Project assignment results are displayed to the students on this page shown in Figure 66. Students can see their Student IDs and their assigned project names on the table.

After the completion of the subsystem, to start the integration with the optimization engine, instead of using a local database, we hosted the database so both teams can work on the same dataset and provide live input and output to the both subsystems. Connection string of the hosted database can be found below.

"DevConnection": "Data Source=94.73.145.9;Initial Catalog=u1761450_BTirme;Persist Security Info=True;User ID=u1761450_BTirme;Password=RJg4:=c@xO72u=5; TrustServerCertificate=True;"

3.1.6. Evaluation

Before integrating the Software Engineering subsystem with the Industrial Engineering subsystem, it is important to assess the subsystem's various qualities. There are several topics which the subsystem is evaluated according to.

Unit Testing: Each and every component present in the subsystem must be made sure to function as intended. This can be done by testing each component by itself and running smaller sized simulations. Simple inputs are given to the system to see if the system triggers certain functions as an output.

There are two different units in the software subsystem of this project. The first unit is the DB. DB entities were created on Visual Studio through ASP.NET methods. The testing of this unit was done by having migrations. Doing migrations and creating local DB were achieved successfully by software engineers. During later parts of the project when the DB was hosted online, a single migration was done by a team member.

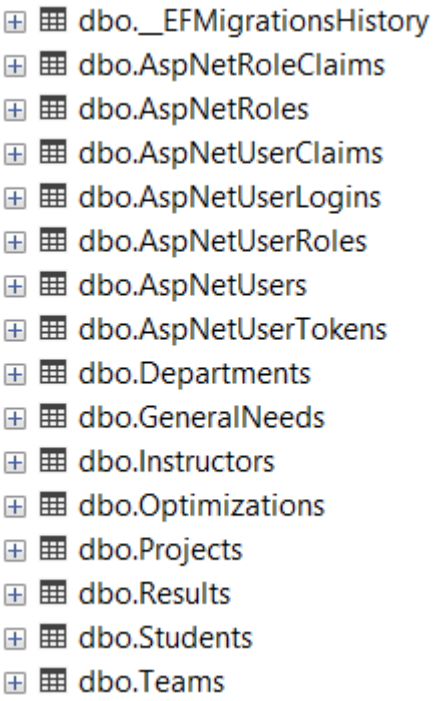


Figure 67. Database tables after a successful migration

It can be seen in Figure x. that database migration was successful and tables were created. Figure 67. was taken after making a migration to the online DB.

The second part of the software subsystem is the frontend side of this project. The main aim of testing this part of the project was to ensure the proper navigation between website views and UIs apart from the DB. The transitions between different website URLs shown in Figures mentioned in 3.1.5 were tested by implementing and checking every navigation button in every page one by one. After the testing period, it was concluded that users are able to navigate between pages based on the boundaries of their roles without any problems.

Integration Testing: Now that the component functionalities are confirmed, the integration of these components is to be provided. The communication between the UI, the website back end and the database must be constructed and tested. Integrating parts into each other piece by piece is a reliable method to test in order to find disconnections between components if any occurs. Possible and impossible inputs are entered to the system to check different behaviors of the system, such as expected errors and data manipulation in the DB, as output.

During and after the integration of the software subsystem within itself, the use-case scenarios mentioned in the 3.1.3 Conceptualization part of the report were applied to the software.

Table 12. Use Case Test Results

Use-case Name	Test Status
Display Results	*Partial Success*
Display Project List	*Success*
Submit Preferences	*Changed/Success*
Login	*Changed/Success*
Add Project	*Success*
Create Team	*Success*

- **Display Results:** Results Page can be successfully reached by both students and instructors as shown in Figure x and Figure x. Instructors can see optimization results any time. However, there is a bug that happens if a student tries to list results without being in a team.
- **Display Project List:** Projects are listed in the Project List Page shown in Figure x.

Students and instructors can easily open this page. One major change made was that while instructors can see every project residing in the DB, students can only see the projects that include their respective departments in any of the 4 project department slots. This was added for better user satisfaction measures.

- **Submit Preferences:** Students are directed to the Select Project Page shown in Figure x. In this page students are able to select 10 projects related to their teams' department. A major change in this page was that teams must make 10 choices. Project selection works successfully and any changes to project choices can be seen in the DB.
- **Login:** Users are welcomed to the website with the Login Page shown in Figure x. The major change to this test was adding the ASP.NET Identity Core library. Identity Core library brought in methods to automatically encrypt user passwords and safely secure user passwords in the database. Even though users are required to take the same actions for logging in, the backend of the software works differently. The system compares the password entered by the user and encrypted password stored in the database if they match each other or not.
- **Add Project:** Instructors can reach the "Add Project Page" shown in Figure x. Created projects can be seen and reached by other pages via pulling the data from the DB. Instructors are not presented with a confirmation message different from the proposal. They are instead redirected to the "Project List Page" seen in Figure x.
- **Create Team:** Students can create teams on their own. The first change made was that students can only include themselves while creating a new team. This is done so that people cannot add others to their teams without their consent. Students must choose a team capacity between 1 and 3 members for better optimization engine functionality.
- **General Changes:** In all pages that require user input, data present on the DB can be chosen in dropdown lists for better usability qualification instead of the user manually entering the corresponding entity's ID.

Usability Testing: Since this application will be used by students more than once, the website UI must be easy-to-learn and easy-to-use. The most optimal way to assure this is to get feedback from fellow university students. Testers should be shown a demonstrative UI and be asked to test how the navigation between pages feel. They should also be requested to fill a survey grading several elements of the UI.

User Acceptance Test could not be achieved. Even though the database is hosted online, users can not access the website without having the source codes of the website and Microsoft Visual Studio installed on their computers.

3.2. Industrial Engineering

3.2.1. Requirements

$$\text{Max } Z = \sum_{i=1}^{S} \sum_{j=1}^{P} Lij * Xij$$

where S is the set of students and L is the set of projects.

L is the preference, 1st choice is 5, 2nd choice is 4 and so on. If the project is not in the list the value will just be 0.

Xij is a binary value and is 1 if student "i" is assigned to the project j and 0 if he is not.

Since the equation is not final, it is currently hard to give an exact performance number for the maximization.

Performance will depend on if a feasible solution exists depending on the inputs. We will likely have around a 500-sample size for our test, for example if 450 of the 450 students put the same project as their first choice, the system will have a very low Z score, even if a solution didn't exist. So instead, we will compare our system to preexisting systems.

First the system will be tested against a very basic system that randomly distributes projects to the students, then it will be the system tested against a FIFO system, will check the 1st preference of the student and assign the project to him if it is available and if not will check the 2nd preference and so on, if none of the student's preferences are available the system will randomly assign him a project.

Lastly, the system will be tested against other systems that were found in the literature review, we will try to replicate the code that is in their article and use the optimization engine that they have on the same list of students as our sample and compare the results, if the system beats the first 2 sets and has a competitive result against the last test, the performance will be sufficient.(The threshold for competitiveness is at least 70% of the Z score for the last test).

For the UI design, the performance will be scored based on real-life surveys conducted on the target group which is 18 to 22-year-old students and university lecturers, they will be given the system UI and tasks, the tasks will be timed. If students can create groups and choose their preferences in less than 3 minutes and instructors can add projects in less than 5 minutes, the UI will be considered successful.

3.2.2. Technologies and methods

For the libraries, we didn't have to consider much as Gurobi was given as the main library that we would be using in the project description, Gurobi is a reputable library that was recommended by various sources so we opted to keep using it instead of looking at alternative libraries.

For the optimization engine methods, we chose to use the Modified LP Method. The main benefit of this method is the fact that it will be simple enough to code while delivering sufficient performance. Simple LP would not give sufficient results and the genetic algorithm would overcomplicate the optimization engine to the point that the performance would suffer and a significant time of our project would have to be used on perfecting it.

For the coding language, the choice was mostly simple as python was the most obvious choice for our system. Although Gurobi is coded in C, using Gurobi in python or java doesn't change its performance by a noticeable margin [6]. So, using C# or C++ would have no benefit for us over using Python and as Python is the language that is used by most of the previous projects that we have seen in literature review, we can leverage that and compare our code to the codes that we have seen in other articles.

3.2.3. Conceptualization

We can refer to table 1 and table 2 in 1.3.2. Concepts part as a reference for the concepts in our subsystem. We considered 4 different optimization methods for the SPA problem and landed on 2 final candidates; Modified LP Method and Other Complex Methods

Other Complex Methods were chosen over Genetic algorithms because of the higher ceiling on these methods. While some of these methods may not generate better results than the Genetic Algorithm, at least one of them will and we can eliminate the other methods in this category. The reason that this was not chosen is mainly the time constraint. According to our Gantt Chart, we have 4 weeks to code the optimization engine after the input methods are done. In this timeframe it seems to be highly unlikely for us to be able to test multiple complex methods and land on the best performing one. This method could have been chosen if the timeframe was double what it was.

The Modified LP Method is the chosen method for our optimization engine. The main justification for this is that it is very familiar to our industrial engineers who took operations research classes. The LP being modified instead of a static basic LP helps us expand the LP into a specified version that we would like to use, we could add elements of other LP methods in our engine so that

it performs up to our requirements. Most of the research done in this area is outside the scope of our school and is different from what our school currently does. For example, our school does not have teacher preferences for the students and teachers are locked into their respective projects and are not allowed to be interchangeable between projects. Another thing is that our projects require members from different departments while most of the research is done with projects on a single department. These differences make our project a unique case that we need a modified model for and this is why we chose the Modified LP method.

For the language of the optimization engine, we have not had to consider many alternatives as one clear choice stood out which was using Python as the main language.

Python is the language that most industrial engineers use in their daily lives as the codes that we need to run are not extremely long and do not require the performance and memory efficiency of C based languages. This project is no different as the optimization engine does not require to be continuously run and does not need extreme amounts of speed. While this is not a competitive advantage since most of the other options also have libraries that will help with compatibility to the database and back-end, python has these libraries as well and with python cross-compatibility will not be an issue. Lastly, our prior knowledge in python will lessen the time cost of doing the optimization engine significantly, the risk of not meeting the 4-week deadline for the optimization engine would have increased significantly if we chose a foreign language that we did not know such as C#. As a result of all these reasons we chose to use Python as our main language for our optimization engine.

3.2.4. Physical architecture

The system will be almost independent from the other systems since it will only be connected to the backend and the database. It will communicate with the backend at the start and at the end of the program. These communications will be brief and will just be the “start” and “stop” commands.

The more crucial connection will be the connection to the database, if the connection gets interrupted at any stage, the program will stop working and will need to be restarted. This is a 2-way communication as the system will start by inputting data from the database and end by outputting data to the database.

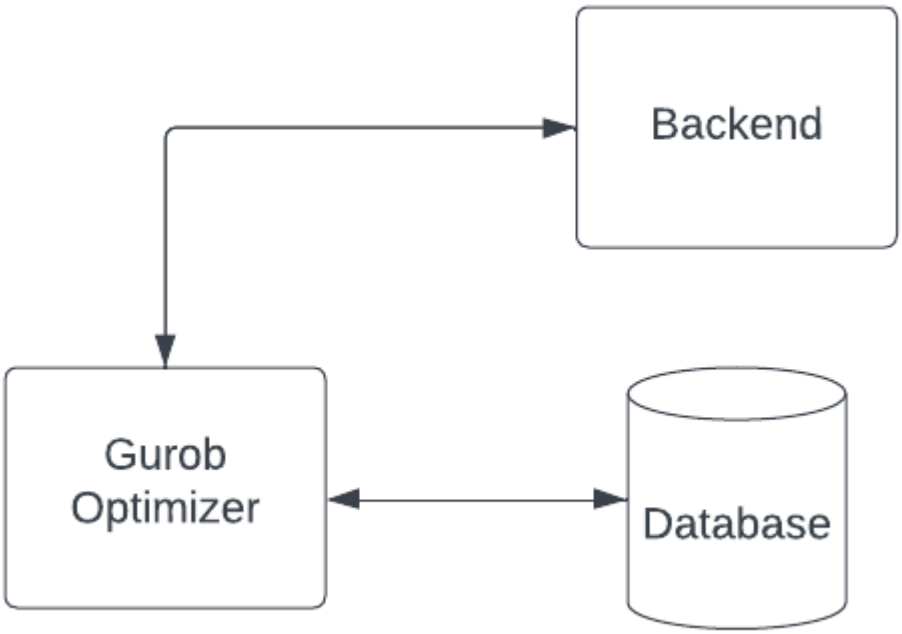


Figure 68. Physical Architecture Communication

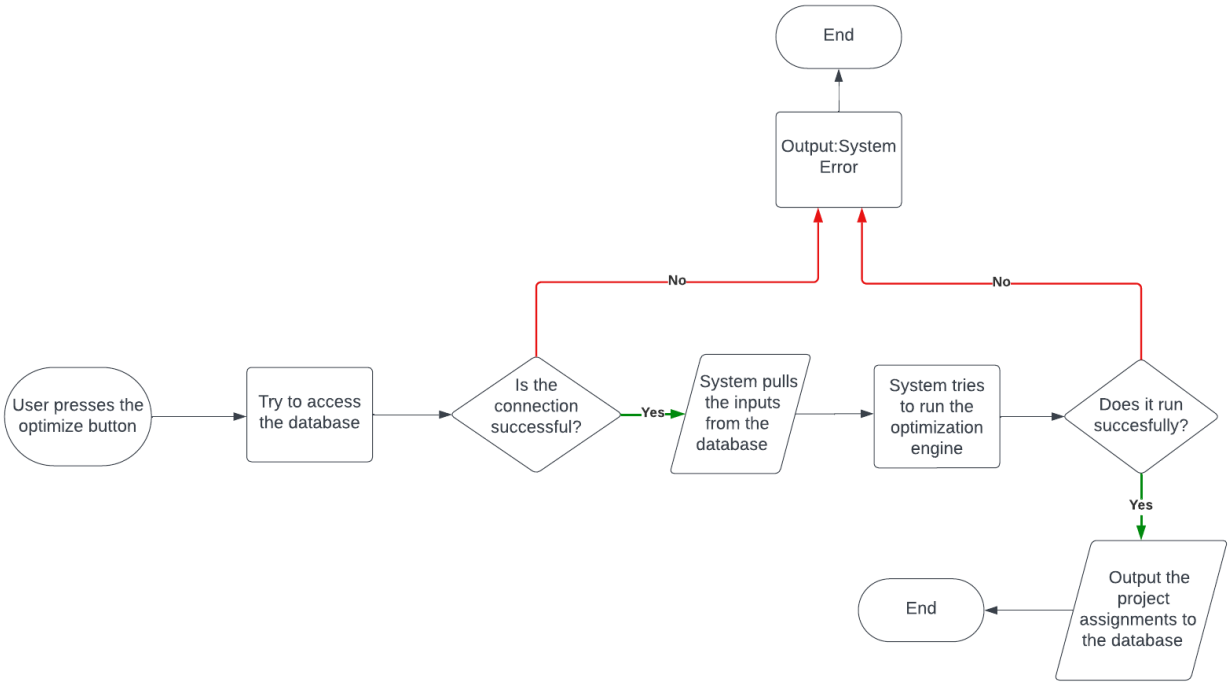


Figure 69. Physical Architecture Process

The Gurobi Python system will have a fairly basic process chart as it does not have many variable processes and the process operation is quite linear.

The program will start when it gets the “optimize” command from the backend of the website, this is the only connection that the program will have with the optimizer. After that, the program will only need to communicate with the database and nothing else, if the connection to the database fails the program will output a system error and stop, if it is successful, it will try to run the optimization engine as the main purpose of the program. If it did not encounter any errors in this step, it would have made all the assignments and matched the students with the projects in an efficient way. If that is the case it will output the data to the database and stop running.

3.2.5. Materialization

Input processes

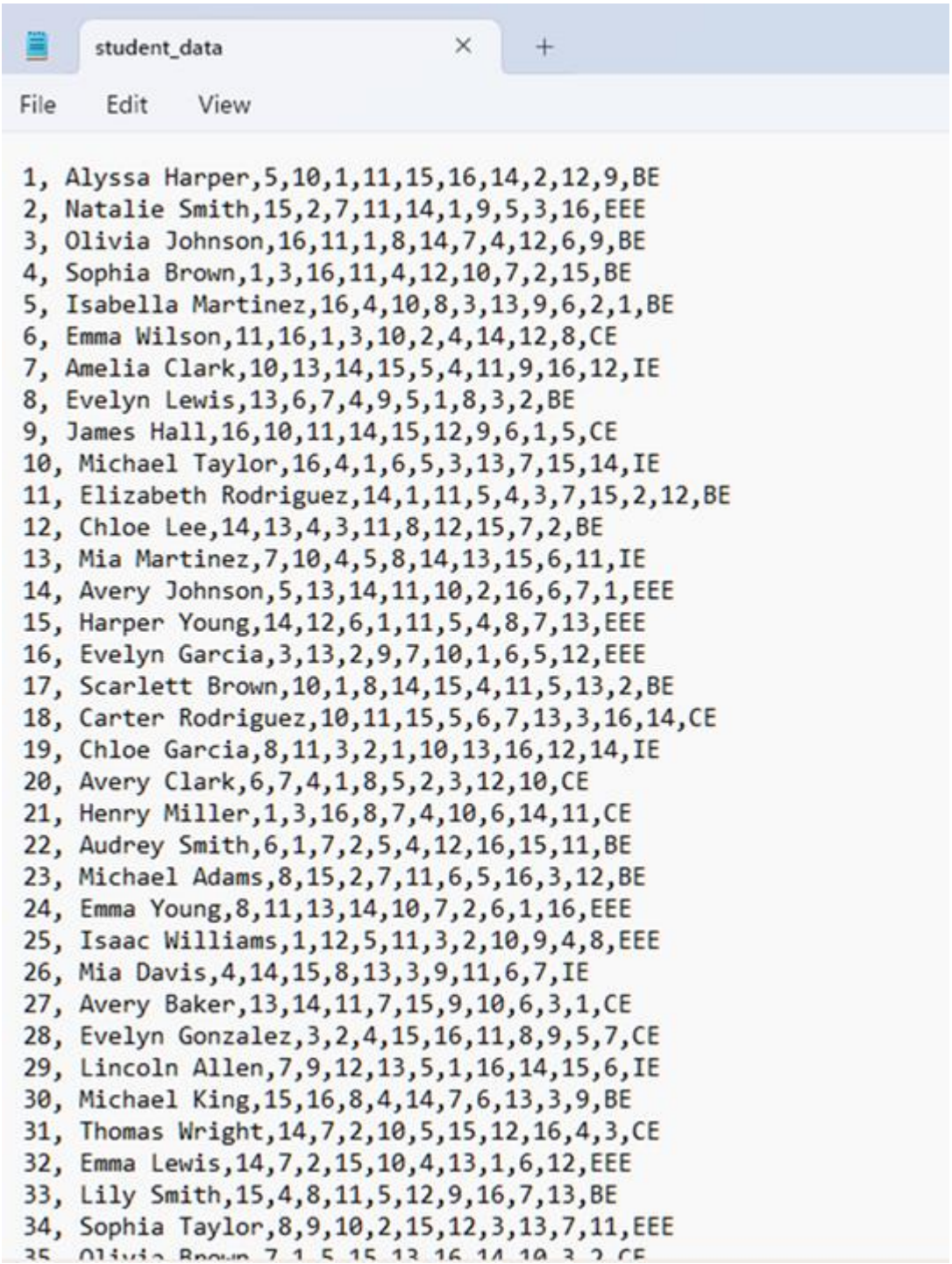


Figure 70. Old Student Input Data

The student data was stored in a ".txt" file as shown, with the format "no, name, preference1, preference2,p3…Department" in later stages a last column that decides the group is planned to be added.

```
1,Crowdwork solutions,BE,4,IE,2,Ms Teacher1,Mr Teacher2
2,Farming job improvement,EEE,3,IE,2,Ms Teacher3,Mr Teacher4
3,Education Access Initiative,CE,1,IE,3,Ms Teacher5,Mr Teacher6
4,Environmental ConIErvation Project,CE,4,IE,4,Ms Teacher7,Mr Teacher8
5,Healthcare Infrastructure Upgrade,CE,3,EEE,5,Ms Teacher9,Mr Teacher10
6,Community Empowerment Program,CE,2,BE,6,Ms Teacher1,Mr Teacher2
7,Water Resource Management Initiative,CE,1,IE,4,Ms Teacher3,Mr Teacher4
8,Social Entrepreneurship Incubator,CE,4,BE,5,Ms Teacher5,Mr Teacher6
9,Technological Innovation Program,CE,2,IE,2,Ms Teacher7,Mr Teacher8
10,Youth Development Scheme,EEE,3,IE,1,Ms Teacher9,Mr Teacher10
11,Artisan Skill Enhancement Project,CE,1,IE,3,Ms Teacher1,Mr Teacher2
12,Gender Equality Advocacy Program,CE,4,IE,2,Ms Teacher3,Mr Teacher4
13,Rural Infrastructure Revitalization,CE,3,IE,2,Ms Teacher5,Mr Teacher6
14,Disaster Preparedness Initiative,EEE,4,IE,4,Ms Teacher7,Mr Teacher8
15,Public Transportation Upgrade,CE,1,BE,6,Ms Teacher9,Mr Teacher10
16,Green Energy Promotion Project,CE,4,IE,3,Ms Teacher1,Mr Teacher2
17,Disaster Preparednes2s Initiative,EEE,4,IE,4,Ms Teacher7,Mr Teacher8
18,Public Transportatio2n Upgrade,CE,1,BE,6,Ms Teacher9,Mr Teacher10
19,Green Energy Promotion2 Project,CE,4,BE,3,Ms Teacher1,Mr Teacher2
20,Disaster Preparednes2s3 Initiative,EEE,4,IE,4,Ms Teacher7,Mr Teacher8
21,Public Transportatio2n3 Upgrade,CE,1,BE,6,Ms Teacher9,Mr Teacher10
22,Green Energy Promotion3 Project,IE,4,BE,3,Ms Teacher1,Mr Teacher2
```

Figure 71. Old Project Input List

The project list is being stored in the format above: "No, name, first department, department limit, second department, department limit, 1st department instructor,2nd department instructor.". It can be noted that there used to be only 2 departments available per project.

```
def read_student_data(file_path):
    student_data = []
    with open(file_path, 'r') as file:
        for line in file:
            No,student_name, p1,p2,p3,p4,p5,p6,p7,p8,p9,p10,dep = line.strip().split(',') #ADD ONE MORE VAR
            student_data.append((No,student_name,p1,p2,p3,p4,p5,p6,p7,p8,p9,p10,dep))
    return student_data

def read_project_data(file_path2):
    project_list=[]
    with open(file_path2,'r') as file1:
        for line in file1:
            No,Project_desc,dep1,dep1_lim,dep2,dep2_lim,instruct1,instruct2 = line.strip().split(',')
            project_list.append((No,Project_desc,dep1,dep1_lim,dep2,dep2_lim,instruct1,instruct2))

    return project_list



file_path = 'student_data.txt'
student_data = read_student_data(file_path)
file_path2="project_list.txt"
project_list=read_project_data(file_path2)
```

Figure 72. Old Input Code

The input from the ".txt" file into python was done by the code figure above. This inputs the student and project lists as a "tuple" list, which is a list that can't be changed. This code also allows for easy addition of inputs without breaking anything.

The main input process was changed from the ".txt" file method to the planned method of inputting straight from the database, this was planned to be a big part of the integration phase in the work structure, fortunately this turned out to be false and the method was significantly easier than anticipated.

```
try:
    conn = pyodbc.connect('''DRIVER={SQL Server}; Server=94.73.145.9;
                          UID=u1761450_BTirme; PWD=RJg4:=c@xO72u=5J; DataBase=u1761450_BTirme''')
    print("Connection successful")
except:
    print("Connection Error")
    exit()
```

Figure 73. Code for DB Connection

The figure above is the start of the integration process where the python program connects to the DB. It will print an error and terminate the program if the connection is unsuccessful.

```
cursor= conn.cursor()

query= "SELECT Team_ID,studentID1,studentID2,studentID3,projectChoice1,projectChoice2,projectChoice3,projectChoice4,projectChoice5,projectChoice6,projectChoice7,projectChoice8,projectChoice9,projectChoice10,Department_ID from dbo.Teams"


cursor.execute(query)
row = cursor.fetchone()
inc=0
inc2=0

student_data=[]
while row:
    inc2+=1
    if( str(row[1])!="None"):
            inc+=1
            student_data.append((inc,str(row[1]),str(row[4]),str(row[5]),str(row[6]),str(row[7]),str(row[8]),str(row[9]),str(row[10]),str(row[11]),str(row[12]),str(row[13]),str(row[14]),str(inc2)))
    if( str(row[2])!="None"):
        inc+=1
        student_data.append((inc,str(row[2]),str(row[4]),str(row[5]),str(row[6]),str(row[7]),str(row[8]),str(row[9]),str(row[10]),str(row[11]),str(row[12]),str(row[13]),str(row[14]),str(inc2)))

    if( str(row[3])!="None"):
        inc+=1
        student_data.append((inc,str(row[3]),str(row[4]),str(row[5]),str(row[6]),str(row[7]),str(row[8]),str(row[9]),str(row[10]),str(row[11]),str(row[12]),str(row[13]),str(row[14]),str(inc2)))

    row = cursor.fetchone()

cursor.close()
```

Figure 74. Code For DB Student Input

The next figure shows the process which takes the "Team" data from the dbo.Teams database and converts it into student data as "Student_Data". This was necessary as the DB stored the data as teams and not students.

It converts the data into format as the ".txt" files above with a "Team" variable at the end.

The format is "Index, Student_No,Pref1,Pref2…. , Department, Team_No" The data is accessed by the using indexes later on so the preservation of the same format is crucial for the program.

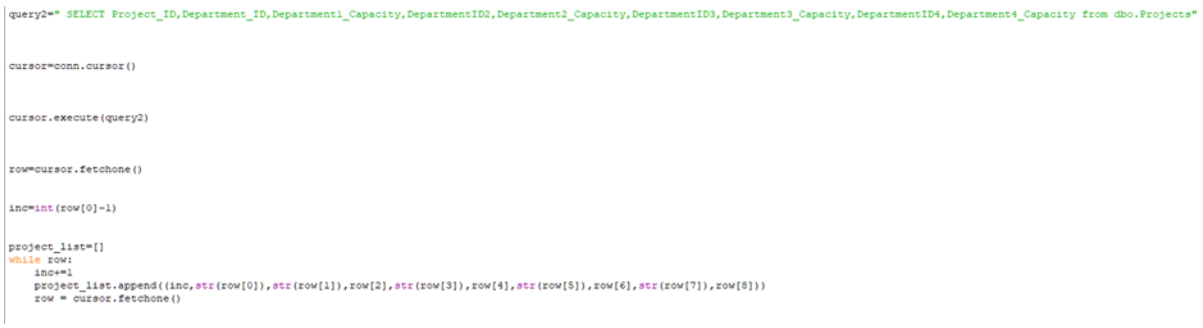


Figure 75. Code for DB Project Input.

The figure above is similar the previous one; it is simpler as conversion wasn't necessary. The system inputs the project data from the dbo.Projects DB as "Project_list". The projects are currently accessed by incrementing indexes in the first row which is a potential risk since the problem could have issues if project data from the DB gets changed. This could potentially be solved by using project IDs in the program.

The format is "Index,Project_ID,Dep1,Dep1Capacity,Dep2….,Dep4Capacity"

```
scount=int(student_data[-1][0])+1
pcount=int(project_list[-1][0])
```

Figure 76. Code for 2 Attributes

The system looks at the indexes of the last student and last project and stores them as scount and pcount to be used in loops as shown in the Figure 77.

```
gp=[]

for i in range(int(student_data[-1][0])):
    gp.append(int(student_data[i][-1]))


gpmax=max(gp)

gsize={}
for i in range(1,gpmax+1):
    gsize[i]=gp.count(i)
```

Figure 77. Code for Team Calculations

Then the system checks the teams of each student and appends the team numbers of each student.

gpmax is calculated to check the number of team so the system knows the amount of teams in the student list. Then the system counts the occurrences of each number in the gp list to calculate the size of each group.

```
sgroup = [[0] * scount for _ in range(scount+1)]

for i in range(1,scount):
    for j in range(1,scount):
        if student_data[i-1][-1]==student_data[j-1][-1] and i!=j and sgroup[j][i]==0:
            sgroup[i][j]=1
        else:
            sgroup[i][j]=0


goodvalues=[]

for i in range(1,scount):
    for j in range(1,scount):
        if sgroup[i][j]==1:
            goodvalues.append((i,j))
```

Figure 78. More Code for Team Calculations

A list called “sgroup” is created which is initially full of 0s then the system loops “scount*scount” times so it compares each student to each student and if their teams numbers are the same and they are not the same student, it assigns “1” to sgroup[i][j] which means student i and j are in the same team it also requires the number to be 0 beforehand so sgroup[i][j] is 1 while sgroup[j][i] is 0 even though they are in the same group. This is to avoid unnecessary overcomplications.

Then these values are added to another list called “goodvalues” which changes the list from being a huge list containing almost all 0s to a small new list which only contains the student that are in the same group in a different format, for example the list is formatted like shown; goodvalues= ([1,3],[6,8]……)

The objective for the calculations above is so that the calculations are done before the gurobi optimizer starts and the optimizer doesn’t have to iterate through as much values.

Gurobi Integration

The objective function is shown below:

Maximize zpsum=z1+z2+z3+z4+z5+z6+z7+z8+z9+z10+p1+p2+p3+p4

Where x[i,j]=1 if "i" student is assigned to the project "j" and x[i,j]=0 otherwise

PV1,PV2=Admin penalty values(200,50 by default)

ZV1,ZV2,ZV3,ZV4,ZV5,ZV6,ZV7,ZV8,ZV9,ZV10=Admin Score values(100,60,40,30,20,15,10,7,5,3 by default)

(y[j]=$\sum_{i=1}^{scount} X[i,j]) for\ each\ j\ value\ in\ pcount$

z[j]=1 if P[j]>0 for each j value in pcount

Subject to:

$(\sum_{j=1}^{pcount} X[i,j]) = 1$ for each i value in scount

$$\left(\sum_{i=1}^{scount} \begin{matrix} x[i,j]\ \ if\ depx[i] = depx[j]) \\ \leq depx\_limit(j)\ for\ each\ j\ value\ in\ pcount \end{matrix}\right.\ for\ x\ in\ dep1, dep2, dep3, dep4$$

Where

$$P1 = \sum_{i=1}^{scount} \square\ -10000\ if\ x[i,j] = 1\ and\ dep[i]!$$
$$= dep[j]\ for\ each\ j\ value\ in\ pcount$$

$$P2 = \sum_{j=1}^{scount} \frac{1}{e^{gsize(i)}} * x[i,j] * x[b,j] * PV1 \; for \; i,b \; in \; goodvalues$$

$$P3 = \sum_{j=1}^{pcount} (Cap(j) - Y[j]) * -PV2$$

$$P4 = \sum_{j=1}^{pcount} Z[j] * -100$$

$Zx = \sum_{i=1}^{scount} X[i,j] * Zvx$ if Spref(i)=Project[j] for each j value in pcount for x=[1,10]

In simpler terms:

Maximize Zpsum=z1+z2….z10+p1+p2+p3+p4

While:

Each student is assigned to only 1 project

Number of students assigned to a project belonging to a specific department can’t exceed department capacity specified in the project data.

Huge penalty point if a student is assigned to a project which doesn’t contain his department

Penalty points if a student is assigned to a different project than his team

Penalty points if a project is partially filled

Z scores are calculated based on the preference of the specified project assigned to the student.

```
m = Model("opt")

x = m.addVars(scount,pcount
              ,vtype=GRB.BINARY,name="x")


y = m.addVars(pcount,vtype=GRB.INTEGER,name="y")

z = m.addVars(pcount,vtype=GRB.BINARY,name="z")
```

Figure 79. Code for Gurobi Variables

These are the 3 decision variables in Figure 79, there are 3 decision variables but only X[i,j] is a real decision variable. The system creates scount*pcount X variables.

X[i,j] where X[i,j]=1 means that the student "i" is assigned to the project "j".

```
m.addConstrs((sum(x[i,j] for i in range(1,int(student_data[-1][-13])+1))<=
            (int(project_list[j-1][3])+int(project_list[j-1][5]))   for j in range(1,int(project_list[-1][-8])+1)) ,name= "c1")
m.addConstrs((sum(x[i,j] for j in range(1,int(project_list[-1][-8])+1))<=1
              for i in range(1,int(student_data[-1][-13])+1)) , name="c2")
m.addConstr((sum(x[i,j] for i in range(1,int(student_data[-1][-13])+1)
                for j in range(1,int(project_list[-1][-8])+1))==(int(student_data[-1][-13])-1)),name="c3")
m.addConstrs((sum(x[i,j] for i in range(1,int(student_data[-1][-13])+1)
                if student_data[i-1][-1]==project_list[j-1][2])<=int(project_list[j-1][3])
                                                for j in range(1,int(project_list[-1][-8])+1)) ,name= "c4")
m.addConstrs((sum(x[i,j] for i in range(1,int(student_data[-1][-13])+1)
                if student_data[i-1][-1]==project_list[j-1][4])<=int(project_list[j-1][5])
                                                for j in range(1,int(project_list[-1][-8])+1)) ,name= "c5")
```

Figure 80. Code for Gurobi Hard Constraints

The constraints made are as follows: c2=limits the program by not allowing it to assign 1 student to multiple projects; c3=forces the program to assign a project to a student so there are no students without projects (the program would not assign a project to a student if it lowered the optimization score) c4,c5= The program checks the department of the student and matches it with the department needed in the project. It also checks the maximum number of students per department from the project list and limits the project allocation accordingly.

The code for departments 3 and 4 has also been added to the code which are essentially the same constraints as c4 and c5 with just different numbers.

```
m.addConstrs((y[j]==sum(x[i,j] for i in range(1,scount)) for j in range(1,pcount)),name="c8")
m.addConstrs((z[j]==or_(x[i,j] for i in range(1,scount)) for j in range(1,pcount)),name="c9")
```

Figure 81. Code for New Gurobi Hard Constraints

These last 2 constraints are better defined as “Definitions” rather than actual constraints shown in Figure 81. They define the values of Y, Z and they are both tied to X.

Y[j] value is the amount of students assigned to the project J and Z[j] is 1 if at least 1 student is assigned to a project

```
p1=sum(x[i,j]*(-10000 if str((student_data[i-1][-1]))!=str(project_list[j-1][2]) and str((student_data[i-1][-1]))!=str(project_list[j-1][4]) else 0
                                    for j in range(1,int(project_list[-1][-8])+1)
                                        for i in range(1,int(student_data[-1][-13])+1))
```

Figure 82. P1 Penalty Value

A penalty value was made as shown in Figure 82. to discourage the program from assigning a project to a student whose department doesn’t belong. This is added as a big penalty point and not a hard constraint due to the fact that if this penalty is forced to apply there is no feasible current solution and instead of the system giving an error this forces it to still make an assignment that can be improved later on.

```
p2=sum(1/(pow(math.e,gsize.get(int(student_data[i-1][-1]))))*x[i,j]*x[b,j]*Ztable[11] for i,b in goodvalues for j in range(1,pcount))

p3=sum((int(project_list[j-1][3])+int(project_list[j-1][5])+int(project_list[j-1][7])+int(project_list[j-1][9])-y[j])*-Ztable[12] for j in range(1,pcount))

p4=sum(z[j]*-100 for j in range(1,pcount)) ##Counter point so projects don't get penalized for being empty
```

Figure 83. Other Penalty Values

More penalty values added as shown in Figure 83. P2 increases the score based on if the students that are in the same team were assigned to the same team. It also exponentially gets smaller as the team size gets bigger which results in less of a reward when a team with 3 members is placed in a different group rather than a team with 2 sizes. This is a small difference since the current maximum size for teams is 3 but it would still function properly and be much more impactful if the maximum team size increases in the future.

P3,P4 are connected penalty points. P3 penalized the system if a project is partially filled instead of full. This is to prevent the system from making 6 projects with only 1 student instead of filling up 1 project even if the preferences end up being better in the former arrangement. P4 is a

counter penalty which penalizes all the projects with at least 1 student in them. This is to counteract the penalty from P3 when a project is totally empty as an empty project is not a bad thing especially in the case that the projects have much more maximum capacity than the amount of students.

```
z1=sum(x[i,j]*(Ztable[1] if int(student_data[i-1][2])==int(project_list[j-1][0]) else 0)
       for j in range(1,pcount) for i in range(1,scount))
z2=sum(x[i,j]*(Ztable[2] if int(student_data[i-1][3])==int(project_list[j-1][0]) else 0)
       for j in range(1,pcount) for i in range(1,scount))
z3=sum(x[i,j]*(Ztable[3] if int(student_data[i-1][4])==int(project_list[j-1][0]) else 0)
       for j in range(1,pcount) for i in range(1,scount))
z4=sum(x[i,j]*(Ztable[4] if int(student_data[i-1][5])==int(project_list[j-1][0]) else 0)
       for j in range(1,pcount) for i in range(1,scount))
z5=sum(x[i,j]*(Ztable[5] if int(student_data[i-1][6])==int(project_list[j-1][0]) else 0)
       for j in range(1,pcount) for i in range(1,scount))
z6=sum(x[i,j]*(Ztable[6] if int(student_data[i-1][7])==int(project_list[j-1][0]) else 0)
       for j in range(1,pcount) for i in range(1,scount))
z7=sum(x[i,j]*(Ztable[7] if int(student_data[i-1][8])==int(project_list[j-1][0]) else 0)
       for j in range(1,pcount) for i in range(1,scount))
z8=sum(x[i,j]*(Ztable[8] if int(student_data[i-1][9])==int(project_list[j-1][0]) else 0)
       for j in range(1,pcount) for i in range(1,scount))
z9=sum(x[i,j]*(Ztable[9] if int(student_data[i-1][10])==int(project_list[j-1][0]) else 0)
     for j in range(1,pcount) for i in range(1,scount))
z10=sum(x[i,j]*(Ztable[10] if int(student_data[i-1][11])==int(project_list[j-1][0]) else 0)
       for j in range(1,pcount) for i in range(1,scount))
```

Figure 84. Code for Z Scores

This part shown in Figure 84. creates a performance score based on the amount of “first choices, second choices...” fulfilled. The multiplication numbers are imported from the database.

```
zpsum=z1+z2+z3+z4+z5+z6+z7+z8+z9+z10+p1+p2+p3+p4

m.setParam("TimeLimit",60*10)

m.setObjective(zpsum,GRB.MAXIMIZE)

m.optimize()
```

Figure 85. Objective Function Code

The last part shown in Figure 85. is a simple command to start the Gurobi optimization process, the zpsum that needs to be maximized is a sum of all score points and penalty points. The time limit is to prevent the system from running longer than necessary, it is currently unnecessary as the system takes about maximum 2 minutes on the computer that it is being tested on (i5 12400f processor 6 cores 12 threads).

```
implist=[]

for i in range(1,scount):
    for j in range(1,pcount):
        if x[i,j].x==1:
            implist.append((student_data[i-1][1],project_list[j-1][1]))
            break

cursor=conn.cursor()

cursor.execute("DELETE FROM dbo.Results")

cursor.close()




cursor=conn.cursor()


for i in range(len(implist)):
    val=[implist[i][0],implist[i][1],student_data[i][-1]]
    query5="INSERT INTO dbo.Results (Student_ID,Project_ID,Team_ID) VALUES (?,?,?)"
    cursor.execute(query5,val)
```

Figure 86. Output Code

After the optimization is done, the system creates a list that stores the students and their assigned projects using the codes shown in Figure 86. Afterwards, the system empties the dbo.Results database and writes the "student_ID and project_ID" of the assigned students to the database. The team ID value is not necessary as the value is already in the database and the "team_ID value which is being sent to the database is different from the actual "Team_ID" which is used in the database due to the optimizer creating new team_IDs instead of using the ones from the database.

After the upload process is done, the system finishes its job and closes itself.

```
emptypro=[]

for j in range(1,pcount):
    pfull=0
    for i in range(1,scount):
        if x[i,j].x==1:
            pfull=1
            break
    if pfull==0:
        emptypro.append(project_list[j-1][0])
```

Figure 87. Empty Project Calculations

The system additionally determines the empty projects and lists them in a list called “emptypro” as shown in Figure 87. above.

The method for that is determining all projects that have at least 1 member and putting them in a list then, putting all the other projects in a separate list.

3.2.6. Evaluation

We have already mentioned most of the testing methods in the 3.2.1 Requirements part, for the sake of not repeating I will only mention the data collection and analysis in this part.

The system will have a Z value that will need to be maximized to be able to gauge performance. Normally this value does not need to be seen so it will not be in the database, but for the sake of collecting data the system will output the Z value every time it runs and it will be saved in a data folder. Using this method, we will be able to make a change in the optimization engine and we will be able to tell if the change has made a positive impact by looking at the Z value.

The runtime of the system will also be collected each time since if the runtime becomes too large it could create systematic problems.

4. INTEGRATION AND EVALUATION

4.1. Integration

When two subsystems, the Software Engineering subsystem and the Industrial Engineering subsystem, of the project were completed independently, the integration process was executed. A meaningful connection was constructed between separate subsystems. This was the most important part of the project since by successfully achieving the integration of two subsystems, the project resulted in a useful and worthy end product.

As it can be seen in the PN at Figure 4, the general integration consists of the communication between the website backend and the optimization engine and the data manipulation capability of the optimization engine. The prior one is only needed to trigger the optimization engine by the user input from the UI. However, the latter one is the decisive function of this application since the optimization

of project choices will be the main quality of life improvement in the academic environment.

The Integration process started with achieving connectivity between the subsystems through a common hosted database. Firstly, the Software team hosted the database and provided the necessary connection information to the Industrial team. Then in the optimization engine, codes shown in Figure 74. are used to connect the engine to the hosted database. After confirming the established connection, engine code is configured to retrieve the required input data as shown in Figure 75. and Figure 76. Then tested if all the required inputs came to the engine including student, project and points data. Finally using the code shown in Figure 86. optimization engine made able to write the output results to the dedicated table in the database.

```
try:
    conn = pyodbc.connect('''DRIVER={SQL Server}; Server=94.73.145.9;
                          UID=u1761450_BTirme; PWD=RJg4:=c@xO72u=5J; DataBase=u1761450_BTirme''')
    print("Connection successful")
except:
    print("Connection Error")
    exit()
```

The above figure shows the initial connection between the DB and the python code, the program tries to connect to the DB and proceeds if the connection is successful. If the connection fails for any reason, the program will immediately stop and print an error.

```
cursor= conn.cursor()

query= "SELECT Team_ID,studentID1,studentID2,studentID3,projectChoice1,projectChoice2,projectChoice3,projectChoice4,projectChoice5,projectChoice6,projectChoice7,projectChoice8,projectChoice9,projectChoice10,Department_ID from dbo.Teams"


cursor.execute(query)
row = cursor.fetchone()
inc=0
inc2=0

student_data=[]
while row:
    inc2+=1
    if( str(row[1])!="None"):
        inc+=1
        student_data.append((inc,str(row[1]),str(row[4]),str(row[5]),str(row[6]),str(row[7]),str(row[8]),str(row[9]),str(row[10]),str(row[11]),str(row[12]),str(row[13]),str(row[14]),str(inc2)))
    if( str(row[2])!="None"):
        inc+=1
        student_data.append((inc,str(row[2]),str(row[4]),str(row[5]),str(row[6]),str(row[7]),str(row[8]),str(row[9]),str(row[10]),str(row[11]),str(row[12]),str(row[13]),str(row[14]),str(inc2)))

    if( str(row[3])!="None"):
        inc+=1
        student_data.append((inc,str(row[3]),str(row[4]),str(row[5]),str(row[6]),str(row[7]),str(row[8]),str(row[9]),str(row[10]),str(row[11]),str(row[12]),str(row[13]),str(row[14]),str(inc2)))

    row = cursor.fetchone()

cursor.close()
```

This part of the integration shown above, is a bit trickier since both sides use different models of storing student data, the DB stores the student data as teams and not as individual students while the python program only takes students as inputs, therefore the program will convert the team data in the database to student data in the following format: ”Increment

,No,Pref1,Pref2,.....Pref10,Dep,Team No" This format is necessary for the gurobi program to work.

```
query2=" SELECT Project_ID,Department_ID,Department1_Capacity,DepartmentID2,Department2_Capacity,DepartmentID3,Department3_Capacity,DepartmentID4,Department4_Capacity from dbo.Projects"


cursor=conn.cursor()


cursor.execute(query2)


row=cursor.fetchone()


inc=int(row[0]-1)


project_list=[]
while row:
    inc+=1
    project_list.append((inc,str(row[0]),str(row[1]),row[2],str(row[3]),row[4],str(row[5]),row[6],str(row[7]),row[8]))
    row = cursor.fetchone()
```

The project list integration code can be seen in figure[x]. Since the formats of projects were quite similar in both the DB and the python code, integration turned out to be quite simple. The necessary data is taken from the database and turned into a python tuple in the format of "Inc,ProjectID,Dep1,Dep1Capacity,Dep2,Dep2Capacity,.....,Dep4Capacity". Similar to the student data, the indexes of this list are important and can't be changed.

```
implist=[]

for i in range(1,scount):
    for j in range(1,pcount):
        if x[i,j].x==1:
            implist.append((student_data[i-1][1],project_list[j-1][1]))
            break

cursor=conn.cursor()

cursor.execute("DELETE FROM dbo.Results")

cursor.close()




cursor=conn.cursor()


for i in range(len(implist)):
    val=[implist[i][0],implist[i][1],student_data[i][-1]]
    query5="INSERT INTO dbo.Results (Student_ID,Project_ID,Team_ID) VALUES (?,?,?)"
    cursor.execute(query5,val)
```

As it can be seen from the figure, the output process is not done in 1 single loop due to it being efficient and resulting in too may nested loops. instead the system find the X[i,j] values which have been assigned the value 1 by gurobi and stores their “i,j” values in a new list called “implist”. afterwards, the implist values are sent to the “dbo.Results” table in the DB.

```
cursor=conn.cursor()

cursor.execute("DELETE FROM dbo.Graphs")



cursor=conn.cursor()
data5=[z1_value,z2_value,z3_value,z4_value,z5_value,z6_value,z7_value,z8_value,z9_value, z10_value,z11_value,len(emptypro),datetime.now()]
query6="INSERT INTO dbo.Graphs (z1,z2,z3,z4,z5,z6,z7,z8,z9,z10,z11,NullProject,Years) VALUES (?,?,?,?,?,?,?,?,?,?,?,?,?)"
cursor.execute(query6,data5)


cursor.execute("DELETE FROM dbo.Emptyprojects")
```

Figure 88. Code for dbo.Graphs

The system takes the zi_values using the code above shown in Figure 88. which are “the amount of students assigned to their ith preference” the zi values were calculated prior in the program and sends them into the “dbo.Graphs” table. It also sends the amount of empty projects in the system which is being used in the results section.

```
for i in range(len(emptypro)):
    query7="INSERT INTO dbo.Emptyprojects (Project_ID) VALUES(?)"
    cursor.execute(query7,emptypro[i])
```

Figure 89. Code for dbo.Emptyprojects

The system also sends the entire list of empty projects into the database using the code in the Figure 89., with just their “project_ID”s for the system to display the empty projects in the results page.

Unfortunately, we failed to implement the software subsystem to make the optimization engine run from the web application UI using a button. Gurobi library can be implemented on .NET Core Web applications but only with C#. Since the optimization engine was written with Python, we were unable to implement the engine directly inside of the web application. We tried to find solutions to this problem, one of the possible solutions was hosting the optimization engine through a web service then sending an API request from the web application to make it run but doing this would be hard to achieve due to Gurobi library dependencies and academic usage restrictions. To minimize the damage of this failure, we redesigned the optimization engine to do all the input and output processes automatically. By doing that all the user has to do is to double click on the Python file on their computer then the engine will do the rest with given input from the DB.

4.2. Evaluation

After the integration is done, the whole system is tested with a simulation of a sample group that consists of 500 randomly generated teams. The evaluation methods already explained in 3.1.6 and 3.2.6 were executed again with the integrated system to determine the degree of unity between each subsystem. Also, the system evaluated on functional and performance requirements mentioned in 1.2.1 and 1.2.2.

After full completion of the project. Using the codes given in Appendix A, random datasets are created. First, 201 random projects consisting of 1 to 4 departments generated. The projects had 1 to 3 student capacity for each department. The amount of departments per project and student capacity were weighted towards 2 departments and 3 student capacity.

Then randomly 500 teams from 10 departments that totally consisted of 987 students with 1 to 3 members each and randomly selected 10 project choices that matched their departments were generated. With these randomly generated test data, optimization engine put to the test. Optimization engine successfully retrieved the input data, optimized it and wrote the results to the database. Optimized assignment results displayed at the dedicated page in the web application. Results of the first test are shown in Appendix B Test 1. section.

After successfully completing the first test. We made changes on the project choices of the teams to see how the system reacts when some projects are chosen as first choice by multiple teams. We selected 2 projects from each department and rearranged the first project choice of the 100 teams' by these projects. This change resulted in the amount of students who were allocated to their most preferred project to slightly drop as one would expect from a biased sample. This sample was closer to a realistic sample than the previous sample so it was also used in the subsequent tests.

In the Test 3. We changed the G1 penalty score from the web application to see how the system would perform with a lower penalty score for breaking a group, this resulted in around 2% more students being allocated to their most preferred projects, this increase was not sufficient enough to warrant breaking more groups so we reverted this value to our previous G1 value of 200.

In Test 4. We changed P1, P2, P3, P4, P5, P6, P7, P8, P9 and P10 scores to find out if this results in a better distribution of preferred projects for preferences between 2 and 10. To achieve that, All values from P2 to P10 were increased by a significant amount. This resulted in all students being allocated to at least their 10th preference, although the amount of students who weren't assigned to any of their preferred projects were quite low at around 1%. These new values completely eliminated that and only decreased the amount of students who were assigned to their

1st preference by 1%. These results are satisfactory enough for the new P values to be default values.

In conclusion, the integration and subsequent testing of the system demonstrated its effectiveness in optimizing project assignments for students. Through a series of evaluations, including the simulation of a sample group of 500 randomly generated teams and 200 projects, the system successfully met both functional and performance requirements. The initial tests, using random datasets and various configurations, validated the system's capability to handle diverse scenarios and optimize allocations efficiently. Adjustments to project choices and penalty scores further refined the system, ensuring realistic and satisfactory outcomes. Notably, the adjustment of preference scores (P1 to P10) significantly improved the distribution of preferred projects among students, achieving a balanced assignment with minimal instances of unassigned preferences. These comprehensive tests confirm that the system is well-prepared for real-world application, providing reliable and optimized project assignments for students across multiple departments.

5. SUMMARY AND CONCLUSION

In summary, the project was started by identifying the reasons that made the project necessary. In part 1.2, we constituted the basic functional and performance requirements of the system according to the needs of our potential users. In part 1.2.3 we limited the scope of our project by considering the different constraints. In part 1.3 we looked at prior work that was done in this topic and the solutions that they found. We tried to consider how these methods apply to our own project and considered the different concepts that they have used. In part 1.4, we planned how the sub-systems in our system will communicate between each other.

In part 2, the work was broken down into work packages which can be assigned into smaller teams within our group. Afterwards the responsibility matrix was made and the responsible and supporting member for different work packages was decided. In the next part, the work packages were ordered according to their prerequisites. In the Gantt chart the timeline of these work packages were planned and set. In part 2.6, the risk assessments were made and the proper plans of actions were prepared.

In part 3, respective requirements for both subsystems were made. The technology and methods were decided. In part 3.x.3, different concepts that were considered in part 1.3.2 were further looked into and in the end the choices that fit our system the best were chosen. In part 3.x.4, The dependencies and communications between various users and sub-systems in our system were mentioned. In part 3.x.5, the implementation processes of both subsystems were covered in detail. In the Evaluation parts, testing validated component functionality and system coherence, enabling successful use-case scenarios and UI usability checks. Performance metrics tracked optimization changes, though formal user acceptance testing faced access constraints. Overall, these efforts aligned the system with project goals and user needs.

In part 4, the project integrated Software Engineering and Industrial Engineering subsystems via a hosted database, overcoming challenges in integrating the optimization engine for automated processes, suggesting potential for future improvements. In part 4.2, after integration was done, the system underwent thorough testing with simulated data. The optimization engine successfully assigned projects to students based on preferences, with detailed results provided in Appendices A and B for further analysis.

In conclusion, this project not only meets the current needs of academic project management but also provides a scalable and adaptable solution for future developments. The use of .NET Core Identity for secure authentication and the Gurobi library for optimization reflects the integration of modern technologies to deliver a high performing and user centric application. Both software engineers and industrial engineers involved in this project benefitted from enhanced learning opportunities,

including gaining deeper insights into .NET technologies and the Gurobi optimization library, respectively. The project shows how well software engineering and industrial optimization techniques can be used in a school setting.

ACKNOWLEDGEMENTS

We wish to thank our advisers…

Prof. Sabri Tankut Atan

Assist. Prof. Tamer Uçar

… for leading us during this project.

REFERENCES

1. Paul R. Harper, Valter de Senna, Israel T. Vieira, Arjan K. Shahani,A genetic algorithm for the project assignment problem,Computers & Operations Research,Volume 32, Issue 5,2005,1255-1265.
2. David F. Manlove, Gregg O'Malley,Student-Project Allocation with preferences over Projects,Journal of Discrete Algorithms,Volume 6, Issue 4,2008,553-560.
3. David J. Abraham, Robert W. Irving, David F. Manlove,Two algorithms for the Student-Project Allocation problem,Journal of Discrete Algorithms,Volume 5, Issue 1,2007,73-90.
4. F.Cooper, D.Manlove, A3/2-approximation algorithm for the student-project allocation problem, in:LIPIcs-Leibniz International Proceedings in Informatics,vol.103,Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik,2018.
5. T. Kulasinghe, “How to connect to a database in ASP.NET core using entity Framework Core,” Medium, https://medium.com/@kulasinghet/how-to-connect-to-a-database-in-asp-net-core-using-entity-framework-core-a11b291d0e0d (accessed Dec. 29, 2023).

APPENDIX A.

Code for randomly generating projects.

```
Random rastgele = new Random();

 string name = "Project_";

 for (int i = 0; i < 200; i++)

 {

     string userName = User.Identity.Name;

     var hasUser = await _UserManager.FindByNameAsync(userName);

     int[] Department_IDRandom = Enumerable.Range(1, 10).ToArray();

     int[] Department_Capasite = Enumerable.Range(1, 4).ToArray();

     int[] randomNumbers = Department_IDRandom.OrderBy(x =>
rastgele.Next()).Take(4).ToArray();

     /*

      100 - 90 hepsi dolu olacak

      */

     p.Project_Name = name + i;

     p.Department_ID = randomNumbers[0];

     p.Department1_Capacity = 3;

     int randomSayi = rastgele.Next(1, 100);

     if (randomSayi >= 99)

     {

         p.DepartmentID2 = randomNumbers[1];

         p.DepartmentID3 = randomNumbers[2];

         p.DepartmentID4 = randomNumbers[3];

         p.Department2_Capacity = 3;

         p.Department3_Capacity = 3;

         p.Department4_Capacity = 3;
```

```
    }
    else if (randomSayi == 98)
    {
        p.DepartmentID2 = 0;
        p.DepartmentID3 = 0;
        p.DepartmentID4 = 0;
        p.Department2_Capacity = 0;
        p.Department3_Capacity = 0;
        p.Department4_Capacity = 0;
    }
    else if (randomSayi >= 90 && randomSayi <98)
    {
        p.DepartmentID2 = randomNumbers[1];
        p.DepartmentID3 = randomNumbers[2];
        p.DepartmentID4 = 0;
        p.Department2_Capacity = 3;
        p.Department3_Capacity = 3;
        p.Department4_Capacity = 0;
    }
    else
    { p.DepartmentID2 = randomNumbers[1]; p.DepartmentID3 = 0;p.DepartmentID4 =
0; p.Department2_Capacity = 3; p.Department3_Capacity = 0;
p.Department4_Capacity = 0;
    }
    p.Instructor_ID = 1;
    p.Project_Description = name + i + " Description";
    p.Create_ID = hasUser.Id;
    p.Create_Date = DateTime.Now;
```

```
    p.Update_ID = null;
    p.Update_Date = DateTime.Now;
    p.IsDelete = false;
    p.IsDelete_ID = null;
    p.IsDelete_Date = DateTime.Now;
    p.IsActive = true;
    _projectService.TAdd(p);
    p.Project_ID = 0;
```

Code for randomly generating teams.

```
try
 {
    Random rastgele = new Random();
    string userName = User.Identity.Name;
    var hasUser = await _UserManager.FindByNameAsync(userName);
    var userID = hasUser.Id;
    var user = await _UserManager.FindByIdAsync(userID);
    var roles = await _UserManager.GetRolesAsync(user);
    var getRole = roles.FirstOrDefault();

    TeamValidator bcv = new TeamValidator();
    ValidationResult result = bcv.Validate(p);
    if (result.IsValid)
    {
        string name = "A";
        for (int i = 0; i < 100; i++)
        {
            int numbers = rastgele.Next(1, 11);
     //       int[] randomNumbers = numbers.OrderBy(x =>
rastgele.Next()).Take(10).ToArray();
            int Department_ID = rastgele.Next(1, 8);
            int Team_Capacity = rastgele.Next(1, 4);
```

```
            int uzunluk = 15;
            string rasgeleString1 = "";
            string rasgeleString2 = "";
            string rasgeleString3 = "";
            for (int j = 0; j < uzunluk; j++)
            {
                rasgeleString1 += (char)rastgele.Next(65, 91);
                rasgeleString2 += (char)rastgele.Next(65, 91);
                rasgeleString3 += (char)rastgele.Next(65, 91);
            }
            var randomNumbers1 = _projectService.TGetList().Where(x =>
x.IsActive == true).Where(x => x.Department_ID == numbers || x.DepartmentID2 ==
numbers || x.DepartmentID3 == numbers || x.DepartmentID4 == numbers).ToList();
            int[] selectProID = randomNumbers1.Select(x =>
x.Project_ID).ToArray();
            int[] selectProIDSelectID = selectProID.OrderBy(x =>
rastgele.Next()).Take(10).ToArray();

            p.Team_Name = name + i;
            p.AssignedProjectID = 0;
            p.Department_ID = numbers;
            p.Team_Capacity = Team_Capacity;
            p.Team_Description = "Denmee";

            for (int e = 0; e < 11; e++)
            {
                p.projectChoice1 = selectProIDSelectID[0];
                p.projectChoice2 = selectProIDSelectID[1];
                p.projectChoice3 = selectProIDSelectID[2];
                p.projectChoice4 = selectProIDSelectID[3];
                p.projectChoice5 = selectProIDSelectID[4];
                p.projectChoice6 = selectProIDSelectID[5];
                p.projectChoice7 = selectProIDSelectID[6];
                p.projectChoice8 = selectProIDSelectID[7];
                p.projectChoice9 = selectProIDSelectID[8];
                p.projectChoice10 = selectProIDSelectID[9];
            }
```

```
        if (Team_Capacity == 3)
        {
            p.studentID1 = rasgeleString1;
            p.studentID2 = rasgeleString2;
            p.studentID3 = rasgeleString3;
        }
        else if (Team_Capacity == 2)
        {
            p.studentID1 = rasgeleString1;
            p.studentID2 = rasgeleString2;
            p.studentID3 = null;
        }
        else
        {
            p.studentID1 = rasgeleString1;
            p.studentID2 = null;
            p.studentID3 = null;
        }

        p.Create_ID = hasUser.Id;
        p.Create_Date = DateTime.Now;
        p.Update_ID = null;
        p.Update_Date = DateTime.Now;
        p.IsDelete = false;
        p.IsDelete_ID = null;
        p.IsDelete_Date = DateTime.Now;
        p.IsActive = true;
        _teamService.TAdd(p);
        p.Team_ID = 0;
    }
}
else
{
    foreach (var item in result.Errors)
    {
        ModelState.AddModelError(item.PropertyName, item.ErrorMessage); } }
return View();
```

APPENDIX B.

Test 1:

Number of students assigned to P1=652 Percentage=66%

Number of students assigned to P2=176 Percentage=17%

Number of students assigned to P3=89 Percentage=9%

Number of students assigned to P4=24 Percentage=2%

Number of students assigned to P5=7 Percentage=0%

Number of students assigned to P6=2 Percentage=0%

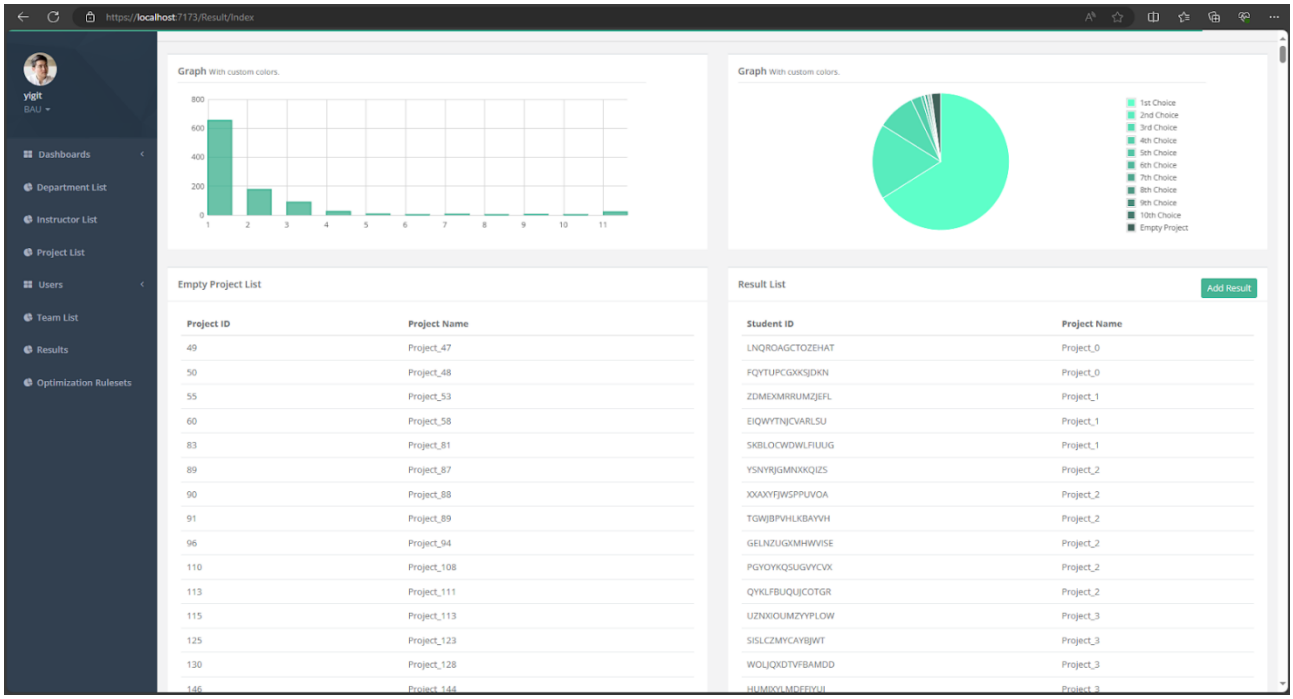
Number of students assigned to P7=6 Percentage=0%

Number of students assigned to P8=3 Percentage=0%

Number of students assigned to P9=4 Percentage=0%

Number of students assigned to P10=2 Percentage=0%

Number of students assigned to non-preferred projects are=22 Percentage=2%



Test 2:

Number of students assigned to P1=580 Percentage=58%

Number of students assigned to P2=226 Percentage=22%

Number of students assigned to P3=89 Percentage=9%

Number of students assigned to P4=38 Percentage=3%

Number of students assigned to P5=18 Percentage=1%

Number of students assigned to P6=4 Percentage=0%

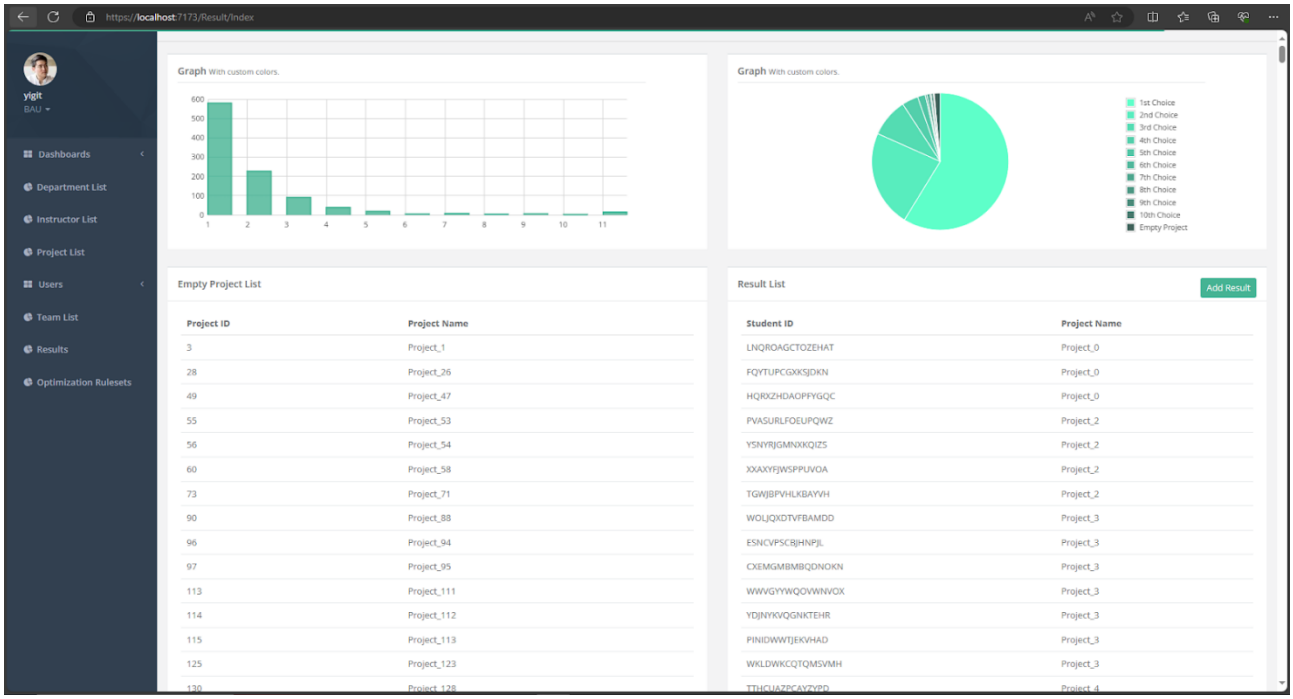
Number of students assigned to P7=7 Percentage=0%

Number of students assigned to P8=3 Percentage=0%

Number of students assigned to P9=5 Percentage=0%

Number of students assigned to P10=2 Percentage=0%

Number of students assigned to non-preferred projects are=13 Percentage=1%



Test 3:

Number of students assigned to P1=586 Percentage=59%

Number of students assigned to P2=223 Percentage=22%

Number of students assigned to P3=92 Percentage=9%

Number of students assigned to P4=36 Percentage=3%

Number of students assigned to P5=13 Percentage=1%

Number of students assigned to P6=4 Percentage=0%

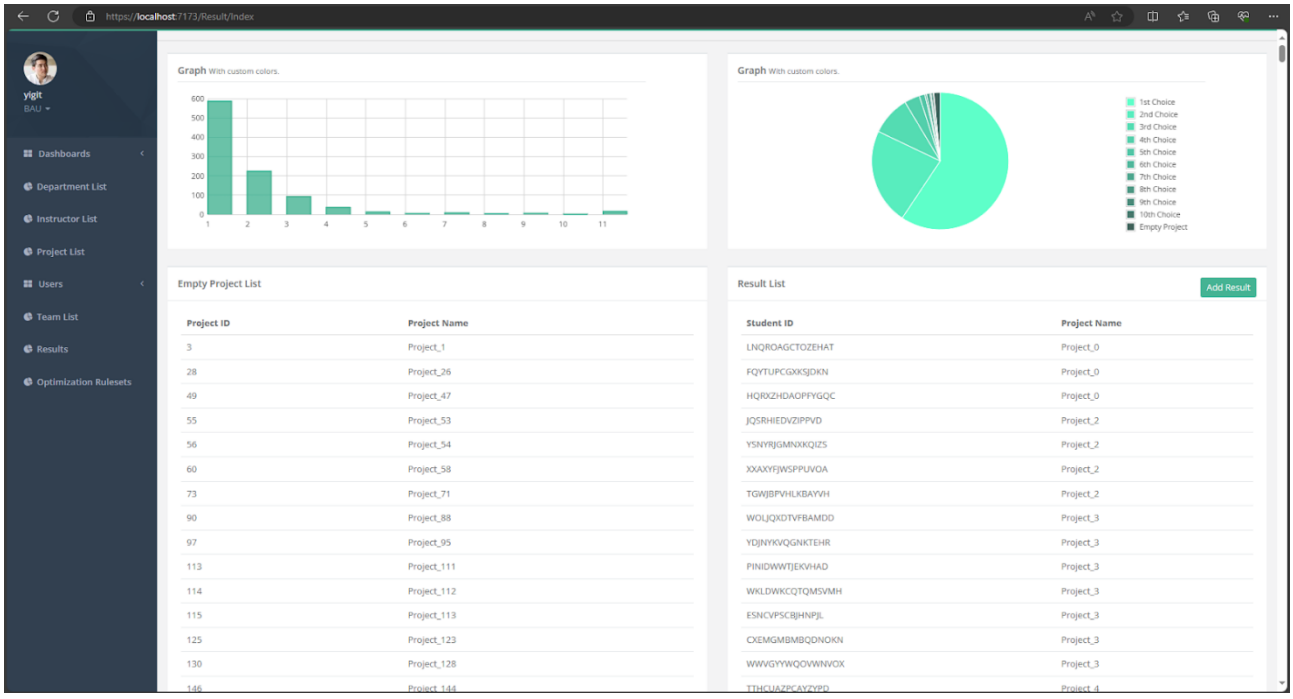
Number of students assigned to P7=8 Percentage=0%

Number of students assigned to P8=3 Percentage=0%

Number of students assigned to P9=5 Percentage=0%

Number of students assigned to P10=1 Percentage=0%

Number of students assigned to non-preferred projects are=14 Percentage=1%



Test 4:

Number of students assigned to P1=569 Percentage=57%

Number of students assigned to P2=223 Percentage=22%

Number of students assigned to P3=104 Percentage=10%

Number of students assigned to P4=34 Percentage=3%

Number of students assigned to P5=20 Percentage=2%

Number of students assigned to P6=7 Percentage=0%

Number of students assigned to P7=16 Percentage=1%

Number of students assigned to P8=3 Percentage=0%

Number of students assigned to P9=10 Percentage=1%

Number of students assigned to P10=1 Percentage=0%

Number of students assigned to non-preferred projects are=0 Percentage=0%