Design Document for "Ieu.App v2"

Berat Bora Altaş 20210602004
Mert Koğuş 20210602037
Betül Özsan 20210602050
Tuna Demirci 20220602025
Hulki Enes Uysal 20210602212

1 Introduction

The project is a standalone desktop application that can handle course scheduling and maintaining process. It enables the user to manage classrooms, handle course schedules, inspect weekly schedules, and make further modifications easily. The target of this application is Windows OS and will be developed using TypeScript, React, Tailwind CSS, and Electron. The application will be in English and provide a user-friendly graphical interface for interaction.

These meet the following non-functional requirements.

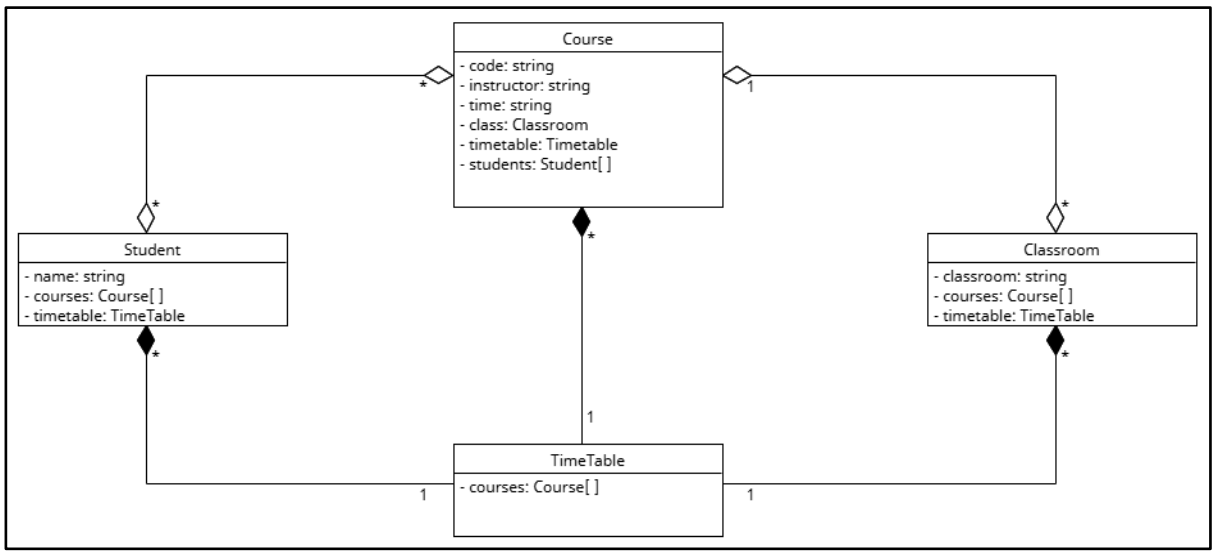
System Non-functional Requirement 1: The system shall be developed using TypeScript, React, Tailwind and Electron 3.5.
~~**System Non-functional Requirement 2**: The program shall be in Turkish.~~
System Non-functional Requirement 2: The program shall be in English.
System Non-functional Requirement 3: The program shall run on Windows operating systems.

2 Software Architecture

The project will follow a modular architecture in React to ensure sustainability, maintainability, and error detection. React's component-based structure allows the development of independent and reusable components, each responsible for certain parts of the UI. The application will work autonomously, reading and processing data from files located in the same directory and displaying the results in the user interface.

2.1 Structural Design

The goal of the app is to fit desired courses into classes and show the user timetables for courses, classes and students. Because we want to create timetables for each of them, we have course, class and student and timetable objects.



As shown in the figure above, a course object holds course code, instructor, course time, class of the course and the students of the course. The class object holds the class name and the courses that will take place in the class. A student object will hold the name of the student and the courses the student is taking. And each of these objects has a timetable unique to them. Timetable objects just hold course objects since they have all the necessary information to create a timetable.

The program doesn't allow the user to create new students or classes, but a new course/meeting can be created and edited.

When a new course is generated from GUI, all its fields are empty. Code, instructor, time should be written by hand via the GUI and the user will add students to the course by the student list if the new course/meeting is not clashing with their own timetables. After that, the user will choose an available classroom with enough capacity from the given list. Then the program will add that course to the students' and the classrooms' course list. Users can also edit the course students and classrooms. Students can be dismissed from the course and new students can be added to the course as mentioned above. The course can be moved to another classroom with enough capacity if that class is available at the desired time.

User Functional Requirement 2: The user shall be able to add new courses**.**

2.2 Behavioural Design

The program uses the given structure from above to store data from parsed input files, stores them in the database and has the functionalities given below to meet the requirements of the project.

Classroom Assignment and Course Distribution

The system is designed in such a way that courses will be distributed among the available classrooms efficiently, using the resources optimally. DistributeCourses() takes in a list of courses, classrooms, and student enrollments and uses an optimization algorithm to assign classrooms considering constraints and features. This function checks for classroom capacity and scheduling conflicts during the process of assignment to meet the requirements set by the system. Capacity validation checks that the number of enrolled students will not exceed the maximum capacity of the classroom, and a scheduling mechanism avoids course conflicts in the same classroom by checking if a time slot is free.

The algorithm gives priority to classrooms that have sizes closest to course enrollment, so it minimizes unused capacity to optimize space utilization. If constraints are violated, other options are considered to get closer to compliance. The solution is scalable; it handles a large number of courses and classrooms by using efficient sorting and filtering mechanisms. This method not only ensures efficient capacity utilization but also maintains schedule integrity, providing a robust framework for classroom assignment. This functionality satisfies Functional System Requirement 2 listed below.

User Functional Requirement 1: The user shall be able to create a suitable timetable for courses

System Functional Requirement 2: The system shall be able to distribute courses among the available classrooms.

Changing a Course's Classroom

The classroom reassignment process makes sure that any change of classroom for a course satisfies capacity and scheduling constraints. In case a user chooses a course for reassignment, the system retrieves its current enrollment and available classrooms and shows only those meeting course requirements. It does not consider rooms with a capacity less than that of the course enrollment, or rooms which conflict with the time slot of the course. If there are no rooms found, it greyed out the "Change Classroom" button and gave a message to the user saying that reassignment was impossible.

The workflow starts when the user selects a course; this triggers the system to filter and list only eligible classrooms. Once a new classroom is selected, the system checks against capacity and scheduling constraints to validate the choice. If valid, it updates the timetable and changes the course assignment to the new classroom. If it does not validate, it will request another classroom selection. In cases when no eligible classrooms exist, the system provides feedback in a way that would not allow for further reassignment attempts. Besides, the system supports the instructor requests for reassignment to higher-capacity classrooms in the case of courses in high demand and thus satisfies the needs of the students effectively. This process maintains the integrity of the scheduling system while accommodating dynamic classroom requirements.

This functionality will meet the User Functional Requirement 3.

User Functional Requirement 3: The user shall be able to change classes of courses.

Editing Courses of Students

The system streamlines student enrollment management by validating classroom capacity and scheduling constraints while dynamically updating course and student schedules. A rerendering mechanism ensures that any changes are immediately reflected in the UI, providing real-time updates and enhancing the user experience. Before enrolling a student, the system checks the classroom's capacity. If there is available capacity, the "Add Student" button is activated; otherwise, it remains inactive unless a student is removed from the course, or the course is reassigned to a classroom with higher capacity.

The system also considers the predefined course schedule, listing only students who are available during the course's time slot to ensure eligibility. Once a student is selected and added to the course, the enrollment updates are automatically propagated. The selected student appears in the course's student list, and the course is added to the student's schedule. The rerendering mechanism updates the UI components dynamically, reflecting state changes such as modified course lists or updated student schedules in real-time. This approach ensures a seamless and intuitive experience for managing student enrollments and course assignments.

This functionality will meet the User Functional Requirement 5.

User Functional Requirement 5: The user shall be able to edit courses of students.

Scheduling Student Meetings

The system also offers a very strong facility in terms of scheduling meetings with students in such a way that there is no conflict. The instructor selects the students who are to attend the meeting. The system then fetches the schedules of the selected students, after which it finds the time slots common to all when the attendees are free.

When common availability is established, the system searches for classrooms that suit the needs of the meeting regarding capacity and resources. It lists available classrooms in front of the instructor, who chooses one of them. Confronted with the confirmation, the system reserves the selected classroom for the specified time and notifies all attendees about the meeting: time, place, and purpose.

It also immediately sends a message to the instructor when no classroom is available at the common times, thus making him/her rethink plans. This smooth procedure ensures proper communication and management of resources without scheduling conflicts. This functionality also contributes to meeting User Functional Requirement 2.

User Functional Requirement 2: The user shall be able to add new courses**.**

Persistent Data System and Data Import

The application will utilize IndexedDB to manage and persist data regarding courses and classroom capacities so that information is still accessible and modifiable between sessions. Once the application starts, the system will invoke the setupDatabase() method, which in turn will automatically read Courses.csv and ClassroomCapacity.csv files from the application path and process their content for storing it into IndexedDB using the saveToDatabase() method. That loads the application with a framed and ready-for-use set of data.

The application will automatically call the method checkDatabase() when opening the application to check whether IndexedDB is already filled up with data. If that is the case, it will directly use the app without rereading again from the CSV files; obviously due to efficiency considerations. In this case, the system calls the loadFromCSV() method to reload the information from the Courses.csv and ClassroomCapacity.csv files again to ensure the application would have all information to work with.

Import Feature: The import feature, via the importData() method, will allow the user to load new CSV files into the application. In case of using this feature, the system will clear the existing data in IndexedDB by calling the clearDatabase() method and overwrite it with the new content by reusing the saveToDatabase() method. This ensures that data is always current, consistent, and ready for use.

System Functional Requirement 3: The program shall have a persistent data system.

System Functional Requirement 4: The program shall be able to read data from given input files.

3 Graphical User Interface

Application bar:

- ‘MANUAL’ and ‘IMPORT’ at the top right. Information about the use of the application will be provided. New data entry will be provided with the ‘IMPORT’ button.

Left Box - Search by Course or Lecturer:

- Course codes (e.g. SE115, SE302) and lecturer names are available.
- The times and days of the lectures (e.g. ‘Monday 8:30’) are prominently displayed.
- Next to each course, the duration of the course (e.g. ‘4’, ‘3’) is labelled in a round purple label.
- Adding a new course will be done by pressing the ‘Add’ button.
- Pressing the course code will redirect the user to the relevant page.

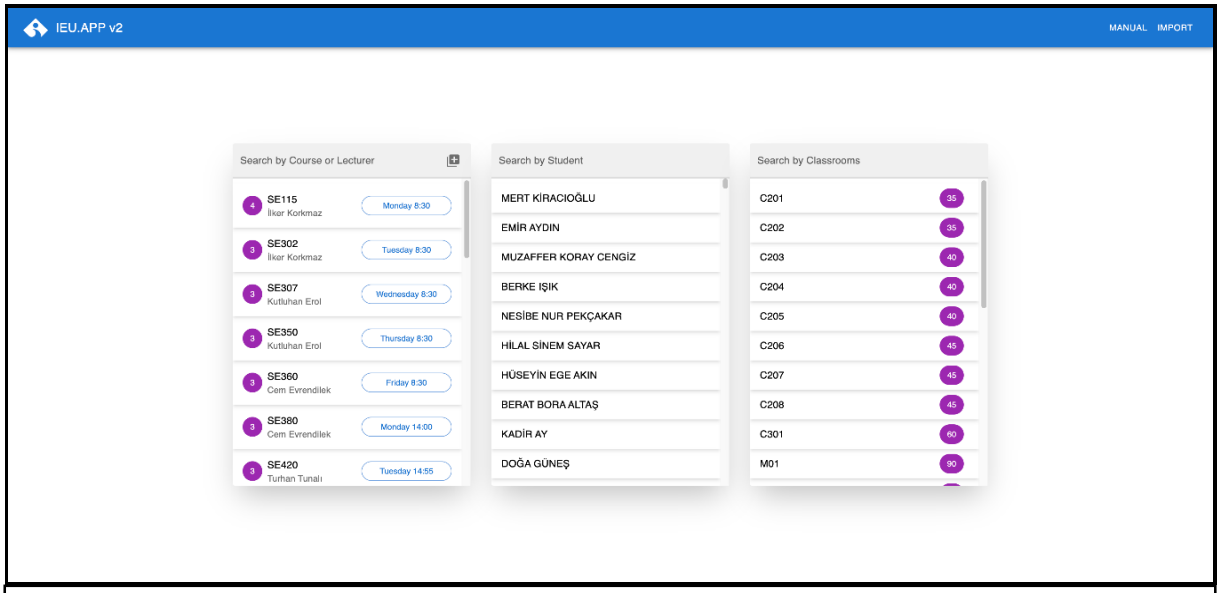
Centre Box - Search by Student:

- There is a list of student names.
- When the student’s name is pressed, it will be directed to the relevant page.

Right Box - Search by Class:

- Class names (e.g. C201, C202) are listed.
- Next to the classes are the class capacities in round purple labels (e.g. ‘35’, ‘40’).
- Upon clicking a class, it will be directed to its’ relevant page.

Upon entering pages of students, classes and courses the program shows the corresponding member’s weekly timetable.



System Functional Requirement 1: The software shall provide a graphical user interface.

User Functional Requirement 4: The user shall be able to view the predefined weekly schedule for any classroom, course, or student, including all relevant details such as times, locations, and participants.

User Non-functional Requirement 1: The user shall access all system functionalities through a detailed and comprehensive user manual.