

**SmartSchedule: Automated Student Course Planner**

by

**Samerah Al Daher**

**Yasmine Sati**

**Kawthar Abdallah**

**Rein Ghattas**

**Ali Yehya**

**Project Report**

Submitted in Partial Fulfilment of the Requirements for the

Software Engineering Course

Department of Computer Engineering

Faculty of Engineering

ٍ Spring 2024-2025



Contents

[List of Tables 5](#_Toc197086229)

[List of Figures 6](#_Toc197086230)

[Project Description 7](#_Toc197086231)

[Project Overview 7](#_Toc197086232)

[Objectives 7](#_Toc197086233)

[Background 7](#_Toc197086234)

[Literature Review 7](#_Toc197086235)

[Li and Womer (2009): 7](#_Toc197086236)

[Garrido and Onaindia (2010): 8](#_Toc197086237)

[Ajanovski (2013): 8](#_Toc197086238)

[Yang and Xie (2017): 8](#_Toc197086239)

[Hossain et al. (2019): 8](#_Toc197086240)

[Tavakoli et al. (2020): 8](#_Toc197086241)

[Yu Chen et al. (2022): 8](#_Toc197086242)

[Beirut Arab University FYP Project (2023): 8](#_Toc197086243)

[Applications 10](#_Toc197086244)

[Alternative Designs 10](#_Toc197086245)

[Project Planning 11](#_Toc197086246)

[Constraints 11](#_Toc197086247)

[Time-Based Constraints 11](#_Toc197086248)

[Course-Based Constraints: 11](#_Toc197086249)

[Student-Specific Constraints: 11](#_Toc197086250)

[Institutional Constraints: 11](#_Toc197086251)

[Project Issues 11](#_Toc197086252)

[Team Members Tasks 11](#_Toc197086253)

[Ethical Issues 12](#_Toc197086254)

[Software Model Process 12](#_Toc197086255)

[Feasibility Study 13](#_Toc197086256)

[Tools/Technology 13](#_Toc197086257)

[1. Web Development Stack 13](#_Toc197086258)

[2. Automation & Scheduling Algorithms 13](#_Toc197086259)

[3. Cloud Infrastructure & Hosting 13](#_Toc197086260)

[4. APIs & Integrations 14](#_Toc197086261)

[5. Collaboration & Version Control 14](#_Toc197086262)

[Standards 14](#_Toc197086263)

[Milestones 14](#_Toc197086264)

[1. Requirements Gathering 14](#_Toc197086265)

[2. System Design 14](#_Toc197086266)

[3. Development Phase 14](#_Toc197086267)

[4. Testing Phase 14](#_Toc197086268)

[5. Deployment & User Feedback 14](#_Toc197086269)

[6. Maintenance & Updates 14](#_Toc197086270)

[Requirements 15](#_Toc197086271)

[Use Cases 15](#_Toc197086272)

[Use Case 1: User Registration and Authentication 15](#_Toc197086273)

[Use Case 2: Course Selection 15](#_Toc197086274)

[Use Case 3: Schedule Generation 15](#_Toc197086275)

[Use Case 4: Schedule Modification 15](#_Toc197086276)

[Use Case 5: Administrator Management 15](#_Toc197086277)

[Use Case 6: Notification and Reminders 16](#_Toc197086278)

[Diagram 16](#_Toc197086279)

[Functional Requirements 16](#_Toc197086280)

[Data Requirements 17](#_Toc197086281)

[Student Information 17](#_Toc197086282)

[Course Data 17](#_Toc197086283)

[Historical Data 17](#_Toc197086284)

[Non-Functional Requirements 17](#_Toc197086285)

[Performance Requirements 17](#_Toc197086286)

[Dependability Requirements 17](#_Toc197086287)

[Maintainability and Supportability Requirements 17](#_Toc197086288)

[Security Requirements 17](#_Toc197086289)

[Usability and Humanity Requirements 17](#_Toc197086290)

[Look and Feel Requirements 17](#_Toc197086291)

[Operational and Environmental Requirements 17](#_Toc197086292)

[Design 18](#_Toc197086293)

[Design Overview 18](#_Toc197086294)

[Class Diagrams 18](#_Toc197086295)

[Dynamic Model 19](#_Toc197086296)

[Subsystem Decomposition 20](#_Toc197086297)

[Frontend (UI): 20](#_Toc197086298)

[Web Server: 20](#_Toc197086299)

[Scheduler Logic: 20](#_Toc197086300)

[Scraping Engine: 20](#_Toc197086301)

[Database Layer: 20](#_Toc197086302)

[Hardware / software 20](#_Toc197086303)

[Software Components: 20](#_Toc197086304)

[Hardware Setup: 20](#_Toc197086305)

[User Interface 20](#_Toc197086306)

[Login Page: 20](#_Toc197086307)

[Calendar View: 20](#_Toc197086308)

[Test Plans 21](#_Toc197086309)

[Features to be tested 21](#_Toc197086310)

[Pass/Fail Criteria 21](#_Toc197086311)

[Approach 21](#_Toc197086312)

[Suspension and resumption 21](#_Toc197086313)

[Testing materials 21](#_Toc197086314)

[Test cases 22](#_Toc197086315)

[Testing schedule 22](#_Toc197086316)

[Implementation 22](#_Toc197086317)

[Results Evaluation 23](#_Toc197086318)

[Conclusion 23](#_Toc197086319)

[Summary 23](#_Toc197086320)

[Novelty 23](#_Toc197086321)

[Integrity and Values 23](#_Toc197086322)

[Future Work 24](#_Toc197086323)

[References 25](#_Toc197086324)

[Appendix 26](#_Toc197086325)

[Github Link: 26](#_Toc197086326)

# List of Tables

[Table 1:Literature Review Summary 9](#_Toc197083487)

[Table 2 Team Members Tasks 11](#_Toc197083488)

[Table 3 Pass/Fail Criteria Table 21](#_Toc197083489)

[Table 4 Test Cases Table 22](#_Toc197083490)

[Table 5 Testing Schedule 22](#_Toc197083491)

List of Figures

[Figure 1 Use Case Diagram 15](#_Toc197083492)

[Figure 2 Design Overview 17](#_Toc197083493)

[Figure 3 Class Diagram 17](#_Toc197083494)

[Figure 4 Dynamic Model Flowchart 18](#_Toc197083495)

# Project Description

## Project Overview

SmartSchedule is a web-based platform designed to help students manage their courses more smoothly and effectively. The system automates the process of selecting courses for registration based on the student's academic status, preferences, and university course offerings. By integrating Flask (Python) as the backend and MySQL as the database, the platform provides a RESTful API that enables students to access course offerings, track their enrolled classes, and generate optimized schedules effortlessly.

## Objectives

1. **Simplify Course Selection:** Allow students to view available courses, including time slots, professors, and prerequisites, in one place, and to have their auto-generated schedule.
2. **Prevent Scheduling Conflicts:** Automatically check for overlapping courses and suggest the best available option.
3. **Accessibility:** Enable students to track their enrolled courses and adjust schedules easily.
4. **Smooth Database Integration:** Store student data, course offerings, and enrollment details in a structured database.
5. **Future Scalability:** Allow expansion for additional features such as a chatbot for advising.

## Background

Selecting courses is a crucial issue in each student's study plan since good management of courses surely ensures a smooth and comfortable learning process. However, managing course registration each semester can be stressful, complicated and time-consuming for students. This issue is caused by the manual course searching, the complexity of different available sections, and the variety of choices for the plan. This project aims to simplify the student course selection process by providing a structured and automated scheduling system with modern web applications and API-driven automation. With this system, students can focus on their education rather than their administrative tasks.

## Literature Review

The following studies and tools provide valuable insights into the design and implementation of course scheduling systems. Each has its strengths and limitations, which have informed the development of SmartSchedule.

### ***Li and Womer (2009):***

This study presents a hybrid MILP/CP Benders Decomposition algorithm for scheduling projects with multi-skilled personnel. The approach effectively separates temporal and logical feasibility, making it suitable for complex planning and scheduling problems. However, it requires expertise in Benders Decomposition and may not be easily adaptable to university course scheduling. [1]

### Garrido and Onaindia (2010):

This work explores the application of AI planning techniques in e-learning environments. The authors propose an integrated planning and scheduling approach that accommodates temporal and resource constraints, making it applicable to real-world scenarios. While the approach is flexible and supports multi-criteria optimization, its practical implementation challenges are not extensively discussed. [2]

### **Ajanovski (2013):**

This study develops a system for creating teacher and teaching schedules based on specific requirements. The system prioritizes teachers based on their teaching load and subject specifications, ensuring efficient scheduling. While the system is effective for teacher scheduling, it lacks detailed information on its functionalities and adaptability to student course scheduling. [3]

### **Yang and Xie (2017):**

 This research proposes a genetic algorithm-based approach for university course scheduling. The algorithm incorporates coevolution to improve performance and efficiently generates high-quality scheduling solutions. The study demonstrates the effectiveness of genetic algorithms in addressing complex scheduling problems, though it may struggle with local optima in some cases. [4]

### Hossain et al. (2019):

 This study employs a particle swarm optimization algorithm to tackle university course scheduling. The algorithm incorporates a forceful swap operation and a repair mechanism to handle constraints effectively. The experimental results demonstrate its efficiency, but further testing is needed to validate its performance in diverse scenarios. [5]

### Tavakoli et al. (2020):

This research proposes a three-stage heuristic algorithm for university course timetabling. The algorithm is applied to the industrial engineering department of a university, achieving a 96% course presentation rate. The approach shows promise but requires further validation in other academic departments. [6]

### Yu Chen et al. (2022):

This research proposes an integer model and a genetic algorithm for university class scheduling. The algorithm aligns satisfaction values with the preferences of professors and students, successfully generating classroom timelines. The study highlights the effectiveness of genetic algorithms in addressing NP-hard scheduling problems. [7]

### Beirut Arab University FYP Project (2023):

This project, developed by Ahmad T. Shaaban, Ahmad S. Abdellatif, Mohammad S. Abu Khurj, and Ali W. Choker, introduces a web-based application for course scheduling at Beirut Arab University. The system uses Next.js, Puppeteer, and DialogFlow to automate course registration, generate optimized schedules, and provide personalized recommendations. The project prioritizes data security, user-friendliness, and scalability, offering a practical solution for BAU students. However, the system's scheduling algorithm may require further optimization to handle complex constraints. [8]

Table 1:Literature Review Summary

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ref | Authors | Description | Advantages | Disadvantages |
| 1 | Li and Womer (2009) | A hybrid MILP/CP Benders Decomposition algorithm for scheduling multi-skilled personnel. | Effective for complex planning and scheduling problems. | Requires expertise in Benders Decomposition |
| 2 | Garrido and Onaindia (2010) | |  | | --- | | AI planning techniques for e-learning, integrating temporal and resource constraints. | |  | | Supports multi-criteria optimization and time/resource constraints | Practical implementation challenges not extensively discussed |
| 3 | Ajanovski (2013) | |  | | --- | | A system for creating teacher and teaching schedules based on specific requirements. | |  | | Facilitates efficient teacher scheduling | Limited information on system functionalities |
| 4 | Yang and Xie (2017) | |  | | --- | | A genetic algorithm-based approach for university course scheduling with coevolution. | |  | | Efficiently generates high-quality scheduling solutions | May struggle with local optima. |
| 5 | Hossain et al. (2019) | A simulated annealing multi-objective algorithm for university course timetabling. | Outperforms traditional genetic algorithms | Limited information on algorithm implementation |
| 6 | Tavakoli et al. (2020) | A three-stage heuristic algorithm for university course timetabling. | Improves course presentation rate | Requires further validation in other academic departments |
| 7 | Yu Chen et al. (2022) | A genetic algorithm for university class scheduling, aligning satisfaction with preferences. | Aligns satisfaction values with preferences | Limited information on algorithm implementation |
| 8 | Shaaban et al. (BAU FYP) | A web-based scheduling system using Next.js, Puppeteer, and DialogFlow for BAU students. | Automates course registration, prioritizes data security and user-friendliness | Scheduling algorithm may require further optimization for complex constraints |

## Applications

The SmartSchedule platform has several practical applications:

1. **University Students**: The target audience of the platform will be university students, who are in need of a course schedule manager.
2. **Academic Advisors:** The platform can be used by advisors to assist students in choosing courses and meeting academic milestones.
3. **University Administrators**: The platform can also assist administrators in overseeing course listings and enrollment statistics more efficiently.
4. **Future Integration**: The platform can be developed further to include features including academic advising chatbots, integration with learning management systems (LMS), and support for several universities.

## Alternative Designs

Several alternative designs were considered during the development of SmartSchedule:

1. **Mobile App:** A mobile application version of SmartSchedule was considered to provide students with on-the-go access to their schedules. However, this was deemed less feasible due to the complexity of integrating with university systems and the need for cross-platform compatibility.
2. **Desktop Application:** A standalone desktop application was also considered, but it was rejected in favor of a web-based platform to ensure broader accessibility and ease of updates.
3. **AI-Powered Scheduling:** An advanced AI-based scheduling system was explored, but it was decided to start with a rule-based algorithm to simplify the initial implementation and ensure reliability.
4. **Integration with Existing Systems:** The option to integrate SmartSchedule with existing university systems (e.g., student portals) was considered, but this was postponed for future development due to the complexity of such integrations.

# Project Planning

## Constraints

The development of effective student scheduling systems necessitates a comprehensive understanding of the various constraints that influence student availability and course placement. These constraints can be broadly categorized into four primary areas: time-based, course-based, student-specific, and institutional.

### Time-Based Constraints

* Class Time Conflicts: A fundamental constraint is the prevention of scheduling two or more classes for a student during overlapping time slots.
* Time Preferences: Students often exhibit preferences for morning or afternoon classes, or for specific days of the week. These preferences should be considered to enhance student satisfaction.
* Breaks and Passing Times: Adequate time must be allocated between classes to allow students to travel between locations and to accommodate lunch breaks and other necessary intervals.
* Extracurricular Activities: Students involved in sports, clubs, or other extracurricular activities have established time commitments that must be integrated into their academic schedule.
* Personal Time: Personal commitments and responsibilities outside of academics must be recognized and factored into the scheduling process.

### Course-Based Constraints:

* Prerequisites: Students must satisfy prerequisite course requirements before enrolling in advanced courses. This ensures proper academic progression.
* Course Availability: Certain courses may be offered only at specific times or semesters, or may have limited enrollment capacities.
* Required Courses: Degree programs often mandate specific courses that students must complete. These requirements must be prioritized.
* Course Load: Institutional policies may limit the number of courses a student can enroll in during a given semester.
* Room and Resource Availability: Specialized courses requiring specific rooms or equipment are subject to the availability of those resources.

### ***Student-Specific Constraints:***

* Individual Needs: Students with disabilities may require specific accommodations, such as accessible classrooms or extended time. Medical conditions may also influence student availability.
* Work Schedules: Students working part-time or full-time have fixed work schedules that must be considered.

### Institutional Constraints:

* Teacher Availability: Teachers have their own scheduling constraints based on their teaching load and other commitments.

## Project Issues

**Software issues**: facing a lot of problems in coding is the viral one and it refers to development and implementation of the web and challenges that can’t be seen only while implementing .Also,there could be issues in data due to sensitive student information and changing requirements which changes in requirments that can lead to huge problems like changing data all over again or working more on database code.In addition to this,is technical challenges which has a difficulty in integrating different systems or comnponents.

## Team Members Tasks

Table 2 Team Members Tasks

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Samerah Al Daher | Ali Yehya | Yasmine Sati | Kawthar Abdallah | Rein Ghattas |
| Manager |  |  |  |  |  |
| Developer |  |  |  |  |  |
| Designer |  |  |  |  |  |
| Frontend Developer |  |  |  |  |  |
| Backend Developer |  |  |  |  |  |
| Database Developer |  |  |  |  |  |
| Testing and Quality Assurance |  |  |  |  |  |
| Security Specialist |  |  |  |  |  |
| Data Analyzer |  |  |  |  |  |
| Data Collection |  |  |  |  |  |

## Ethical Issues

One significant issue we may encounter is the unfair advantage and course registration discrepancies. When implementing the automatic course registration program, ethical concerns arise regarding the potential for creating an unfair advantage and inconsistencies in course registration. Students utilizing the automatic program may have the advantage of completing the registration process faster compared to those who register manually. This could lead to automatic program users securing seats for desired courses before manually registering students have the opportunity to do so. Consequently, students registering manually may face a disadvantage and find it challenging to enroll in their preferred courses. Privacy and data security are also critical ethical considerations. Students may have concerns regarding the privacy and security of their data. They may worry about their data being shared or vulnerable to hacking, as this is a common issue experienced by many companies that handle user data. Ensuring robust security measures, implementing data protection protocols, and providing transparency about how their data is handled and protected can help alleviate these concerns. It is important to address the limitation of our program and manage user expectations. While we strive to deliver the best results, no program is perfect. Occasionally, the system may encounter errors or provide unclear data that may not be optimal. Communicating these limitations upfront and being transparent about the program's capabilities can help manage user expectations and minimize disappointment. System reliability and availability is another ethical concern. Any system has the potential for frequent downtime or technical issues, which can prevent students from registering for courses on time. Ensuring the application's stability, implementing backup plans, and promptly addressing technical issues are crucial to minimize disruptions and provide a reliable registration experience for all students. By acknowledging and addressing these ethical issues, such as unfair advantage, privacy concerns, limitations, and system reliability, we can strive to develop and implement an automatic course registration program that prioritizes fairness, transparency, and the responsible handling of student data. Regular monitoring, feedback collection, and continuous improvement will be essential to ensure the program.

## Software Model Process

For the SmartSchedule project, we will adopt the Agile software development model, specifically the Scrum framework, to ensure flexibility, iterative progress, and continuous feedback. Agile is well-suited for this project because it allows us to adapt to changing requirements, deliver incremental updates, and maintain close collaboration with stakeholders (e.g., students, faculty, and university administrators). The project will be divided into sprints, each lasting 2-3 weeks, with clearly defined goals such as implementing core features (e.g., course scheduling, conflict detection, and AI chatbot integration). Regular sprint reviews and retrospectives will ensure that the team stays on track and continuously improves the development process.

## 

## Feasibility Study

The purpose of this feasibility study is to assess the technical viability of developing a webbased application for course scheduling and registration at Beirut Arab University (BAU). This study evaluates various aspects, including hosting options, system requirements, technical specifications, cost analysis, potential challenges, and compliance considerations. Hosting Options: Several hosting options have been considered for the web application. Azure, a popular cloud service provider, offers a learning service specifically designed for students. Another option is the Amazon AWS free tier, which is available to all customers. HOSTINGER, a web hosting service, offers affordable plans with additional free months and is suitable for personal web hosting as well as small or medium companies. GitHub provides free hosting with security guarantees and efficient file management capabilities. A comprehensive cost analysis will be conducted to determine the most suitable hosting option in terms of pricing, features, scalability, and security. System Requirements and Technical Specifications: To ensure a successful implementation, it is crucial to define the system requirements and technical specifications of the web application. This includes considering factors such as expected user load, scalability requirements, performance benchmarks, and compatibility with different devices and browsers. By clearly defining these requirements, we can ensure that the web application meets the needs of BAU students and provides a smooth user experience.

Potential Challenges and Mitigation Strategies: The development and deployment of a web application may encounter various challenges. It is important to identify and address these challenges proactively. Potential challenges include integration issues with existing university systems, compatibility issues across different platforms and devices, and the need to adhere to legal and regulatory compliance, such as data protection and privacy laws. Mitigation strategies will be devised to address each challenge, ensuring a seamless development and deployment process. Cost Analysis: A thorough cost analysis will be conducted to evaluate the financial feasibility of the project. This analysis will take into account hosting costs, development and maintenance expenses, licensing fees for required software or frameworks, and any additional costs associated with third-party services or APIs. By analyzing the costs involved, we can ensure the project remains within budget and determine the return on investment for BAU. Compliance Considerations: Compliance with legal and regulatory requirements is crucial for any application handling student data. We will assess the legal and regulatory landscape, including data protection and privacy laws, and ensure that the web application complies with these requirements. Measures such as data encryption, secure storage practices, and obtaining necessary consents from users will be implemented to protect the privacy and security of student data. Project Timeline and Resource Allocation: A detailed project timeline will be developed, outlining the different stages of the project, milestones, and key deliverables. Resource allocation, including human resources, hardware, and software requirements, will also be considered to ensure the project's successful completion within the defined timeline. Collaboration and Stakeholder Engagement: Collaboration with BAU stakeholders, including students, faculty, and administrators, will be essential throughout the project. Regular communication, feedback collection, and stakeholder engagement will help ensure that the web application meets the specific requirements and expectations of BAU. This collaboration will involve conducting user testing, gathering user feedback, and incorporating suggestions and improvements into the development process. Conclusion: This feasibility study provides an overview of the technical aspects and considerations related to the development of a web-based application for course scheduling and registration at BAU. By evaluating hosting options, defining system requirements, addressing potential challenges, conducting a cost analysis, considering compliance requirements, and outlining the project timeline and resource allocation, we aim to develop a comprehensive plan for the successful implementation of the web application. Collaboration with stakeholders and ongoing communication will be crucial for the project's success.

## Tools/Technology

### 1. Web Development Stack

1. Frontend: HTML, CSS, JavaScript
2. Backend: Node.js with Express.js for API development
3. Database: MySQL for structured course data management

### 2. Automation & Scheduling Algorithms

1. Python-based scheduling algorithms to optimize course planning
2. AI/ML integration to suggest the best course schedules based on student preferences and prerequisites

### 3. Cloud Infrastructure & Hosting

Firebase or AWS for authentication and database hosting

Docker for containerized deployment and scalability

### 4. APIs & Integrations

1. University course catalog integration via RESTful APIs
2. Google Calendar API for exporting student schedules

### 5. Collaboration & Version Control

GitHub for code management and version control

Jira or Trello for project tracking and team collaboration

## Standards

## 

To maintain high-quality standards, we will follow IEEE software engineering standards for requirements specification, design, coding, and testing. The project will adhere to MVC (Model-View-Controller) architecture for clean separation of concerns, ensuring scalability and maintainability. Additionally, we will use version control (Git) for collaborative development and CI/CD (Continuous Integration/Continuous Deployment) pipelines to automate testing and deployment. By combining Agile practices with industry standards, we aim to deliver a robust, user-friendly, and scalable solution for course scheduling at Beirut Arab University.

## Milestones

### 1. Requirements Gathering

* Getting user needs (e.g., prerequisites, semester limits, credit hours).\

### 2. System Design

* Designing the database (courses, prerequisites, schedules).
* Planning the algorithms for course selection and scheduling.
* Choosing tools and technologies (Python, web-based UI, database).

### 3. Development Phase

* Basic UI & Input Handling (Creating a form for students to enter courses).
* Database Setup (Implement the course storage system).
* Creating the Algorithms
* User Interface (Allow students to view and modify plans).

### 4. Testing Phase

* Test different student inputs (valid/invalid course selections).

### 5. Deployment & User Feedback

* Deploying on a local or cloud server.
* Getting feedback from students and improving the project.

### 6. Maintenance & Updates

* Fixing bugs and enhance features based on user feedback.

# Requirements

## Use Cases

### ***Use Case 1***: User Registration and Authentication

**Primary Actor:** Student / Admin

**Description:** Allows users to create an account and authenticate themselves before

accessing system features.

**Preconditions:** User has not registered or needs to log in.

**Main Flow:**

1. User accesses the system and selects "Login".

2. For registration, the user provides the required details (name, email, student ID,

password).

3. System verifies and stores the details.

4. For login, the user enters their email and password.

5. The system checks if the information is correct.

**After login:** The user can now use all the features of the system.

### Use Case 2: Course Selection

**Primary Actor:** Student

**Description:** Enables students to select courses based on their curriculum and academic

requirements.

**Preconditions:** Students must be logged in.

**Main Flow:**

1. Student views available courses.

2. Students select preferred courses from the list.

3. System verifies prerequisites and credit limits.

4. Courses are added to the student's provisional schedule.

**Outcome:** Selected courses are saved for scheduling.

### Use Case 3: Schedule Generation

**Primary Actor:** Student

**Description:** Automatically generates an optimized class schedule based on selected

courses, course timings, and availability.

**Preconditions:** Courses must be selected.

**Main Flow:**

1. Student initiates schedule generation.

2. System processes course data, time slots, instructor availability, and potential

conflicts.

3. An optimal schedule is generated and displayed.

**Postconditions:** The student gets a schedule with no time conflicts.

### Use Case 4: Schedule Modification

**Primary Actor:** Student

**Description:** Allows students to manually adjust their schedules or resolve conflicts.

**Preconditions:** A schedule must be generated.

**Main Flow:**

1. Student reviews the generated schedule.

2. Student requests changes (remove or swap a class).

3. System validates the change and updates the schedule accordingly.

**Postconditions:** Student receives an updated version of their schedule.

### Use Case 5: Administrator Management

**Primary Actor:** Admin

**Description:** Lets admins manage courses, class times, and user accounts.

**Preconditions:** Admin must be logged in.

**Main Flow:**

1. Admin accesses the dashboard.

2. Admin adds/updates/removes course information or schedules.

3. Admin manages students.

**Postconditions:** The information is saved and ready for students to use.

### Use Case 6: Notification and Reminders

**Primary Actor:** System

**Description:** Sends updates and reminders to students regarding their schedules,

deadlines, and changes.

**Preconditions**: Student must have a valid notification setup.

**Main Flow:**

1. System monitors changes and important dates.

2. System sends emails or in-app notifications.

**Postconditions**: Students are kept informed of relevant schedule-related events.

### Diagram

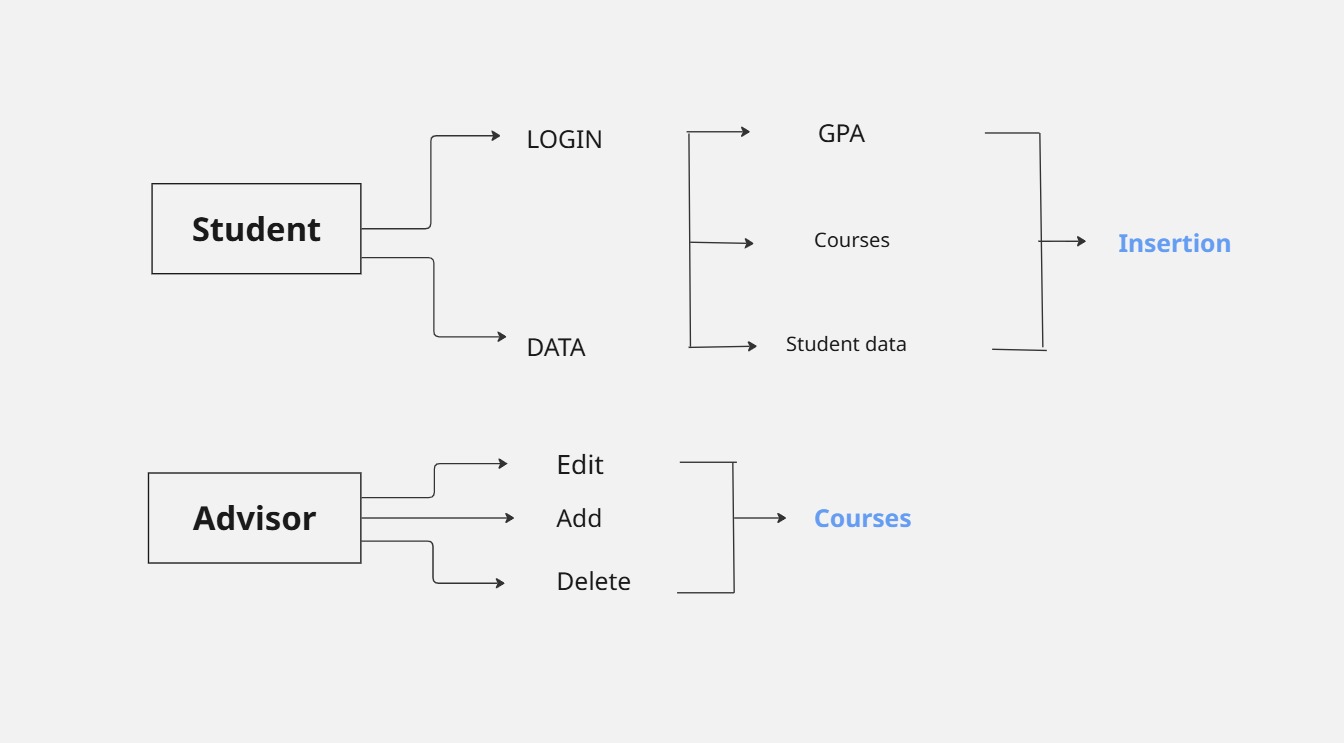


Figure 1 Use Case Diagram

## Functional Requirements

1. Students can log in to the website using their ID and password.
2. The system retrieves and processes the student's data, which is associated with their ID on BAU iConnect X-Files.
3. System gets available courses from course offerings.
4. Based on the gathered information, the system analyzes the prerequisites completed by the student and generates a list of recommended courses to be taken.
5. The student is presented with a dynamic plan on a dedicated page, which showcases the suggested studying plan.
6. In addition to recommending essential courses, the system provides guidance on elective courses that align with the student's academic goals and interests.
7. The system generates a timetable, optimizing the schedule by considering time gaps between courses to provide an efficient and feasible schedule for the student.
8. The system provides the option for automatic registration or allows students to register manually, giving them flexibility and control over the registration process.
9. The data is stored locally so everything is secure and no data leaks will happen.
10. The system uses modern and new technologies which is updated frequently.
11. The system offers great performance.

## Data Requirements

Student Information: The system needs to store and retrieve student information, including

their IDs, names, majors, and academic records. This data is essential for generating

personalized course recommendations and ensuring that prerequisites are met.

Course Data: The system requires comprehensive data on the available courses, including

course codes, names, descriptions, prerequisites, instructors, and class timings. This

information is used to generate course schedules and make appropriate recommendations to

students.

Historical Data: Gathering historical data on past course schedules, enrollment patterns, and

student preferences can provide valuable insights for optimizing future schedules. This data

can help in identifying popular courses, understanding scheduling conflicts, and improving

overall scheduling efficiency.

## Non-Functional Requirements

### Performance Requirements

Performance requirements include the system's ability to generate the student schedule in a short period of time which is set to be accepted at 5 seconds. Also, the system must handle up to 20 concurrent users at the same type to ensure a smooth process for students.

### Dependability Requirements

Dependability in this system is achieved first by being available almost all the time, especially during the registration period. It also must have precise and clean data integration since the course offering information is crucial for smooth and reliable registration.

### Maintainability and Supportability Requirements

The system should follow modular architecture where the logic, web server, and database are separated which makes maintenance and future development easier.

### Security Requirements

Due to the sensitive data we’re dealing with, the system must have a secured login process to access the students' data and generate the proper schedule. Also, students must only be able to view their information and schedules.

### Usability and Humanity Requirements

The object of the project is to provide a smooth registration process for students. So the web app must be simple and used on different devices with a single interaction. It must also show clear messages for errors and issues during schedule generation.

### Look and Feel Requirements

The schedule must be shown in an interactive calendar that includes course information in a professional way. The web pages must follow a unified and modern design.

### Operational and Environmental Requirements

The system must be compatible with different browsers and environments (Chrome, Edge, Safari, etc..)

# Design

## Design Overview

The system has 4 main parts:

* Backend Logic
* Frontend (HTML+CSS+JavaScript (FullCalendar.js)
* Selenium Handler (for Scraping)
* Database (Course Offering, Course Catalog, Student Cache, Study Plan)

The following diagram shows these parts and the flow of the program based on entries and events:

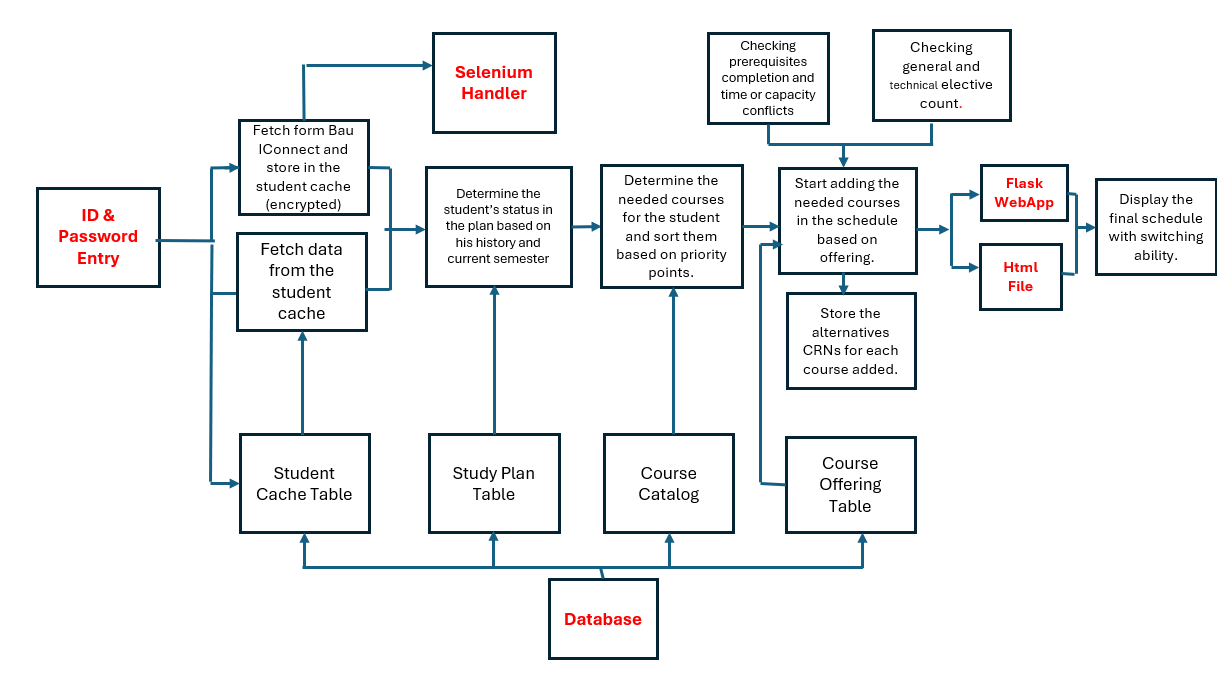


Figure 2 Design Overview

## Class Diagrams

The project logic uses two main classes for students and courses to generate the schedule based on their different characteristics with the generate\_schedule function:

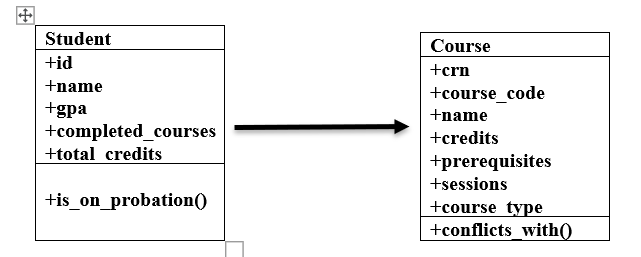


Figure 3 Class Diagram

## Dynamic Model

Start

Credentials found in Student Cache

Enter Credentials

Scrape Student Data

Fetch Student Data from Database

Caching in Database

Loading course offering, course catalog, and study plan.

Yes

No

Courses are selected and prioritized based on student data and study plan

Finding Conflicts

Yes

Move to next section or course

Add CRN

Store alternative CRNS

Display Schedule with switching abilities

End

No

Figure 4 Dynamic Model Flowchart

The dynamic model shows the system's behaviour during the scheduling generation process. This process includes the following main steps:

1. User logs in with student ID and password.
2. If the student data is in the student cache, it fetches data directly. If it's not, it scrapes from the Bau iConnect page and then caches the data.
3. The scheduler loads course offerings, the catalogue, and the study plan.
4. The system, based on students' data, calculates the approximate semester and makes a list of needed courses sorted based on the priority of courses.
5. Based on the course offering, conflicts and credit limits are checked, and a schedule is built.
6. The schedule is displayed in an interactive calendar with section-switching capability.

## Subsystem Decomposition

The system is organized and separated into clear modules to support maintainability and scalability:

### Frontend (UI):

* HTML + FullCalendar.js (displays schedule visually)
* Jinja2 templating for dynamic rendering

### Web Server:

* Flask (webapp.py)
* Handles login, routing, and data exchange with the backend

### Scheduler Logic:

scheduler.py

* Processes academic data, checks rules, and builds the schedule

### Scraping Engine:

selenium\_handler.py

* Automates login and scraping from the iConnect portal

### Database Layer:

MySQL (Hosted on Railway)

* Stores course catalogue, offerings, study plan, and student cache

## Hardware / software

### Software Components:

* Python 3.10
* Flask web framework
* Selenium with ChromeDriver
* MySQL database
* HTML/CSS/JavaScript (FullCalendar)

### Hardware Setup:

* Developer laptop (local development and testing)
* Hosting server (Railway)

This configuration supports modular development and is easily portable to a cloud environment.

## User Interface

The system features a user-friendly and modern web interface:

### Login Page:

* Secure student ID/password input

### Calendar View:

* Displays courses by week using FullCalendar.js
* Shows course info on hover
* Dropdowns for Switching Sections:
* Allows switching between available CRNs for each course
* The design ensures usability, responsiveness, and accessibility across devices.

# Test Plans

## Features to be tested

* Login System: Verifying student ID and password functionality (including from cache and Selenium scraping).
* Schedule Generation Logic: Correct handling of prerequisites, corequisites, and elective limits.
* Conflict Detection: Time overlap prevention and session grouping (lecture/lab).
* Section Switching: Ability to change CRNs without conflict.
* Capacity Check: Courses with 0 capacity should not be added.
* Calendar Display: Correct rendering of the schedule with tooltips and UI elements.
* Generation Time :Time to display the schedule.

## Pass/Fail Criteria

Table 3 Pass/Fail Criteria Table

|  |  |  |
| --- | --- | --- |
| **Test** | **Pass Criteria** | **Fail Criteria** |
| Login | Student logs in, data is retrieved or scraped correctly | Wrong credentials don't trigger error / data not retrieved |
| Schedule Generation | Courses added without violating rules | Conflicts added, wrong credit limits |
| UI Calendar | Courses appear with correct times and tooltips | UI doesn't display events or misplaces them |
| Section Switching | Alternate section appears, no conflict | Conflict not detected / UI breaks |
| Generation Time | Generation in less than 10 seconds. | Generation exceeds 10 seconds. |

## Approach

We use a black-box testing approach focused on:

* Manual testing for user flows
* Debug logging for backend verification
* Edge case validation (maxed electives, no available slots)

## Suspension and resumption

Testing is suspended if:

* Selenium scraping fails due to university website changes.

Testing is resumed once:

* Connection or logic is fixed and verified.

## Testing materials

**Hardware**: PC with Internet access.

**Software**:

* Python 3.13
* MySQL Server (Railway)
* Google Chrome + ChromeDriver
* Flask
* FullCalendar.js
* Selenium

## Test cases

Table 4 Test Cases Table

|  |  |  |
| --- | --- | --- |
| **Test Case** | **Input** | **Expected Result** |
| Valid Login | ID + Password | Student schedule is generated |
| Invalid Login | Wrong password | Error message shown |
| Full Course | Course with 0 capacity | Not added to schedule |
| Elective Limit | Student has max electives | Elective skipped |
| Section Switch | Switch CRN to same time | Conflict alert shown |
| Session Conflict | Add overlapping courses | Second course rejected |
| Generation Time | Request schedule with login | Fast Generation (10 seconds) |

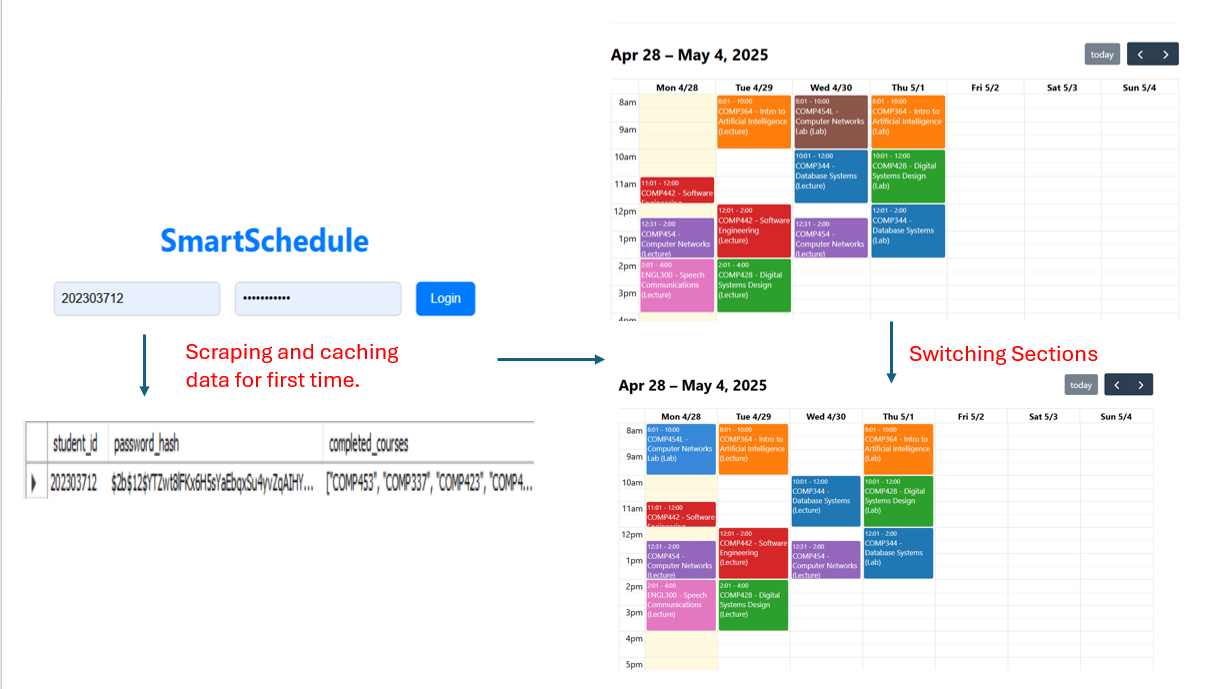
## Testing schedule

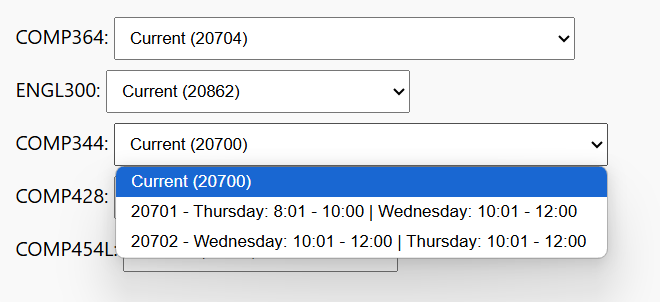
Table 5 Testing Schedule

|  |  |
| --- | --- |
| Date | Activity |
| April 12 | Testing Basic Algorithm |
| April 15 | Scraping Student Data |
| April 15 | UI rendering and calendar verification |
| April 20 | Conflict switching, elective limits tested |
| April 26 | Final validation and performance test |

# Implementation

The system generates a personalized, conflict-free schedule for each student. It displays the output as an interactive calendar with course details and allows switching between sections. The output also ensures prerequisites, credit limits, and capacity rules are respected. The schedule reflects real-time availability and is optimized for usability.



****

# Results Evaluation

The system successfully fetches student data and accordingly generates a schedule with no time or capacity conflicts and with courses prioritised according to their academic status and completed courses.The schedules perfectly aligns with the academic advisors recommended schedules and with the study plan. The interface is smooth and user-friendly, allowing switching sections easily. However, the loading time in scraping wasn’t in the accepted range, and the problem was addressed by caching students' data for future logins.

# Conclusion

## Summary

SmartSchedule is an intelligent scheduling system designed to automate and optimise the course registration process for university students. By integrating Selenium-based scraping from the BAU iConnect portal, a Mysql-backed course catalog and offering system, and a dynamic frontend built with Flask and FullCalendar.js, SmartSchedule generates conflict-free, credit-balanced, and prerequisite-aware schedules. It also allows students to switch between course sections and view their schedules interactively.

## Novelty

This project introduces a hybrid dynamic scheduling mechanism that:

* Fetches real student data in real-time using automated scraping.
* Prioritises core and major courses while respecting elective limits.
* Uses a visual, switchable interface rather than static or manual forms.
* Implements smart decision-making based on both academic rules and live course capacity.

Unlike traditional static planners, SmartSchedule reacts to changing conditions, such as time conflicts, seat availability, and student progress.

## Integrity and Values

SmartSchedule maintains strict data integrity by:

* Encrypting student credentials before storage.
* Supporting academic fairness by applying the same rules to all students.

It follows ethical standards in software development and academic integrity, ensuring reliability and trustworthiness.

## Future Work

The system can be improved and extended in the following ways:

* Deployment on a public server with secure login and HTTPS support.
* Admin dashboard to manage course offerings and update catalogues dynamically.
* Integration with university registration systems to allow one-click enrollment.
* AI-based suggestion engine that factors in course difficulty, instructor rating, or student preferences.

# References

**[1]** [Li, H., & Womer, K. (2009). Scheduling projects with multi-skilled personnel by a hybrid MILP/CP benders decomposition algorithm. *Journal of Scheduling, 12*(3), 281-298. doi:10.1007/s10951-008-0079-3](https://doi.org/10.1007/s10951-008-0079-3)

**[2]**  [Garrido, A., & Onaindia, E. (2010). On the application of planning and scheduling techniques to e-learning. In *Proc. 23rd Int. Conference on Industrial, Engineering & Other Applications of Applied Intelligent Systems (IEA-AIE 2010) - LNAI 6096* (pp. 244-253). Springer.](https://link.springer.com/chapter/10.1007/978-3-642-13025-0_26)

[3] Reference: Ajanovski, V. V. (2013). Integration of a course enrolment and class timetable scheduling in a student information system. International Journal of Database Management Systems, 5(3), 85-95.

**[4]** [Yang, L., & Xie, C. (2017). Research on model of course scheduling system for ideological and political teaching in colleges and universities based on particle swarm optimization. *Revista Facultad de Ingeniería, 32*(17), 657-663.](https://revistas.unal.edu.co/index.php/refame/article/view/64789)

**[5]** [Hossain, S. I., Akhand, M. A. H., Shuvo, M. I. R., Siddique, N., & Adeli, H. (2019). Optimization of university course scheduling problem using particle swarm optimization with selective search. *Expert Systems with Applications, 127*, 9-24. doi:10.1016/j.eswa.2019.02.026](https://doi.org/10.1016/j.eswa.2019.02.026)

**[6]** [Tavakoli, M. M., Shirouyehzad, H., Lotfi, F. H., & Najafi, S. E. (2020). Proposing a novel heuristic algorithm for university course timetabling problem with the quality of courses rendered approach; a case study. *Alexandria Engineering Journal, 59*(5), 3355-3367. doi:10.1016/j.aej.2020.05.004](https://doi.org/10.1016/j.aej.2020.05.004)

**[7]** [Chen, Y., Ghasemi, P., Bayanati, M., Ebrahimi, M., & Khalijian, S. (2022). A novel optimization approach for educational class scheduling with considering the students and teachers' preferences. *Discrete Dynamics in Nature and Society, 2022*, 5505631. doi:10.1155/2022/5505631](https://doi.org/10.1155/2022/5505631)

**[8]** [Beirut Arab University FYP Project (2023). Streamlining Course Scheduling: Revolutionizing Timetabling for BAU Students.](https://baudom-my.sharepoint.com/:b:/g/personal/ahy641_student_bau_edu_lb/EZrAlBDkQHpPut1iwd3Y5k4BysR52LBJof2FNAJeB5l3RA?e=oAhgQL)

# Appendix

## Github Link:

<https://github.com/ceprogrambau/SoftwareEngineering/tree/main/Group%201>