metin, yazı tipi, logo, ticari marka içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

Smart Academic Planner

***Group Members***

Ethem Kesim 210302205

Emre Çubuk 220302393

Ozan Umut Güney 200302109

Contents

[1. Introduction 3](#_Toc199800794)

[2. Background 3](#_Toc199800795)

[3. System Architecture & Design 3](#_Toc199800796)

[4. Module Descriptions 3](#_Toc199800797)

[4.1 AppMain.java 3](#_Toc199800798)

[4.2 LocalAIClient.java 3](#_Toc199800799)

[4.3 PromptBuilder.java 3](#_Toc199800800)

[4.4 CourseSelectionController.java 3](#_Toc199800801)

[4.5 ScheduleViewController.java 4](#_Toc199800802)

[4.6 Course.java 4](#_Toc199800803)

[4.7 CourseStorage.java 4](#_Toc199800804)

[4.8 StudentCourseController.java 4](#_Toc199800805)

[4.9 DBHelper.java 4](#_Toc199800806)

[4.10 PrerequisiteChecker.java 4](#_Toc199800807)

[5. Implementation & Code Review 4](#_Toc199800808)

[6. Experimental Results 5](#_Toc199800809)

[7. Challenges Faced 5](#_Toc199800810)

[7.1 AI Model Deployment & Error Handling: 5](#_Toc199800811)

[7.2 AI Integration: 5](#_Toc199800812)

[7.3 Database Technology Selection: 5](#_Toc199800813)

[7.4 Prerequisite & Duplicate Errors: 5](#_Toc199800814)

[8. Conclusion 6](#_Toc199800815)

# 1. Introduction

This report presents the design, implementation, and evaluation of the Smart Academic Planner & Course Clash Detector, a Java-based desktop application that helps university students build conflict-free schedules while ensuring course prerequisite constraints are respected. The planner simplifies academic planning through intuitive GUI interactions, database-backed validations, and optional AI-assisted scheduling logic.

# 2. Background

University students often face difficulties when selecting courses due to time slot overlaps and prerequisite constraints. Manual inspection of multiple course offerings and schedules is time-consuming and error-prone. To mitigate these issues, we developed an automated scheduling system that considers available courses, prerequisite rules, and time constraints. The system is capable of generating an optimized weekly schedule either through a traditional backtracking algorithm or by utilizing a locally hosted AI model via HTTP.

# 3. System Architecture & Design

The application adopts the Model-View-Controller (MVC) architecture. JavaFX is used for the graphical interface, while MySQL serves as the backend database to persist data related to courses, students, and prerequisites. Scheduling logic is decoupled from the UI, allowing both AI-based and algorithmic backtracking approaches to be implemented cleanly.

The architecture is modular and includes core components like course input views, prerequisite validation logic, scheduling UI, and AI integration for advanced functionality. Data flow is strictly managed through in-memory buffers and persistent database calls.

# 4. Module Descriptions

## 4.1 AppMain.java

Initializes and launches the JavaFX application, serving as the main entry point.

## 4.2 LocalAIClient.java

Sends scheduling prompts to a local AI server (such as LM Studio) via HTTP POST and retrieves textual timetable responses.

## 4.3 PromptBuilder.java

Constructs natural language prompts based on selected course data for consumption by large language models.

## 4.4 CourseSelectionController.java

Handles course input logic in the GUI. Manages user inputs for course name, day, time, and invokes prerequisite checks.

## 4.5 ScheduleViewController.java

Displays the final schedule returned by either the AI model or the backtracking algorithm.

## 4.6 Course.java

Defines the data structure for a course, including its name, day, and time range.

## 4.7 CourseStorage.java

Temporarily stores course entries during the planning session to facilitate inter-scene communication.

## 4.8 StudentCourseController.java

Handles student-course relationship data operations in the database such as insertions and lookups.

## 4.9 DBHelper.java

Provides a static method to connect to the MySQL database using JDBC.

## 4.10 PrerequisiteChecker.java

Verifies whether a student has completed all prerequisites for a given course using SQL joins.

# 5. Implementation & Code Review

The application was developed using modular Java code, with each component fulfilling a well-defined responsibility. The implementation emphasizes separation of concerns between UI, logic, and data access. Controllers manage user input and transitions between views. Utility classes handle database connections and validations. Model classes represent core entities such as courses, and temporary storage objects maintain session-level selections.

In the controller layer, CourseSelectionController is responsible for accepting course details and triggering prerequisite checks. ScheduleViewController handles the display logic and acts as a dispatcher between AI-based and backtracking-based schedule generation.

The LocalAIClient and PromptBuilder modules are used for constructing AI prompts and sending them to a local language model endpoint. On the database side, DBHelper and StudentCourseController manage persistence and integrity of student-course relationships. PrerequisiteChecker enforces academic rules by ensuring prerequisite completion before enrollment.

# 6. Experimental Results

The system was tested using various course input combinations to verify schedule correctness and prerequisite validation. In AI mode, the prompt was successfully generated and submitted to a local instance of LM Studio running MythoMax. The model returned readable, conflict-free schedule suggestions, which were parsed and formatted into the GUI.

In backtracking mode, the algorithm iterated through all combinations of available time slots and selected the one without any conflicts. Test cases demonstrated the effectiveness of both approaches in handling overlapping courses and ensuring that no two time intervals clashed.

Additionally, we confirmed that the prerequisite enforcement logic correctly blocked attempts to enroll in advanced courses before satisfying dependencies.

# 7. Challenges Faced

7.1 AI Model Deployment & Error Handling: Integrating the AI model into the local environment required setup and handling of HTTP errors such as 'Connection Refused' when the local server was inactive. To mitigate this, fallback mechanisms were added to allow the application to run in backtracking mode when AI services are unavailable.

7.2 AI Integration:  
Selecting and configuring a suitable local AI model (e.g. MythoMax via LM Studio) was challenging. API issues like connection errors required fallback options and careful testing.

7.3 Database Technology Selection:  
Finding a suitable database solution that matched the project's graph-like requirements (such as Neo4j) proved difficult. Eventually, a relational MySQL schema was used, though it required workarounds to handle complex prerequisite relationships.

7.4 Prerequisite & Duplicate Errors:  
Preventing duplicate course entries and ensuring accurate prerequisite validation required both SQL constraints and extra logic at the application level.

# 8. Conclusion

The Smart Academic Planner successfully enables students to build conflict-free schedules with integrated prerequisite validation. Its modular JavaFX architecture, combined with AI and algorithmic components, provides flexibility and automation for academic planning. This project showcases the application of software engineering principles such as modularity, separation of concerns, and user-centered design. It also demonstrates the power of combining traditional logic-based algorithms with modern AI techniques to solve real-world problems in education.