# Software Design Document

**DESIGN REPORT FOR SENG-383**

Can Eltayeb 202328202

Cem ÖZAL 202228203

## Project 1: BeePlan - Course Scheduler

### 1. Introduction

#### 1.1 Purpose

This document outlines the design for "BeePlan," a Python-based GUI application. The system's purpose is to generate conflict-free course schedules for a university department, managing complex constraints such as instructor availability, classroom capacity, and specific scheduling rules (e.g., lab hours, exam blocks).

#### 1.2 Scope

The system will:

* Allow an administrator to input data via JSON or CSV files (courses, instructors, rooms, constraints).
* Implement a scheduling algorithm (heuristic or backtracking) to generate a conflict-free timetable.
* Display the generated schedule in a visual weekly timetable grid.
* Provide validation reports listing any conflicts or rule violations that could not be resolved.
* The primary user is a department administrator or scheduler.

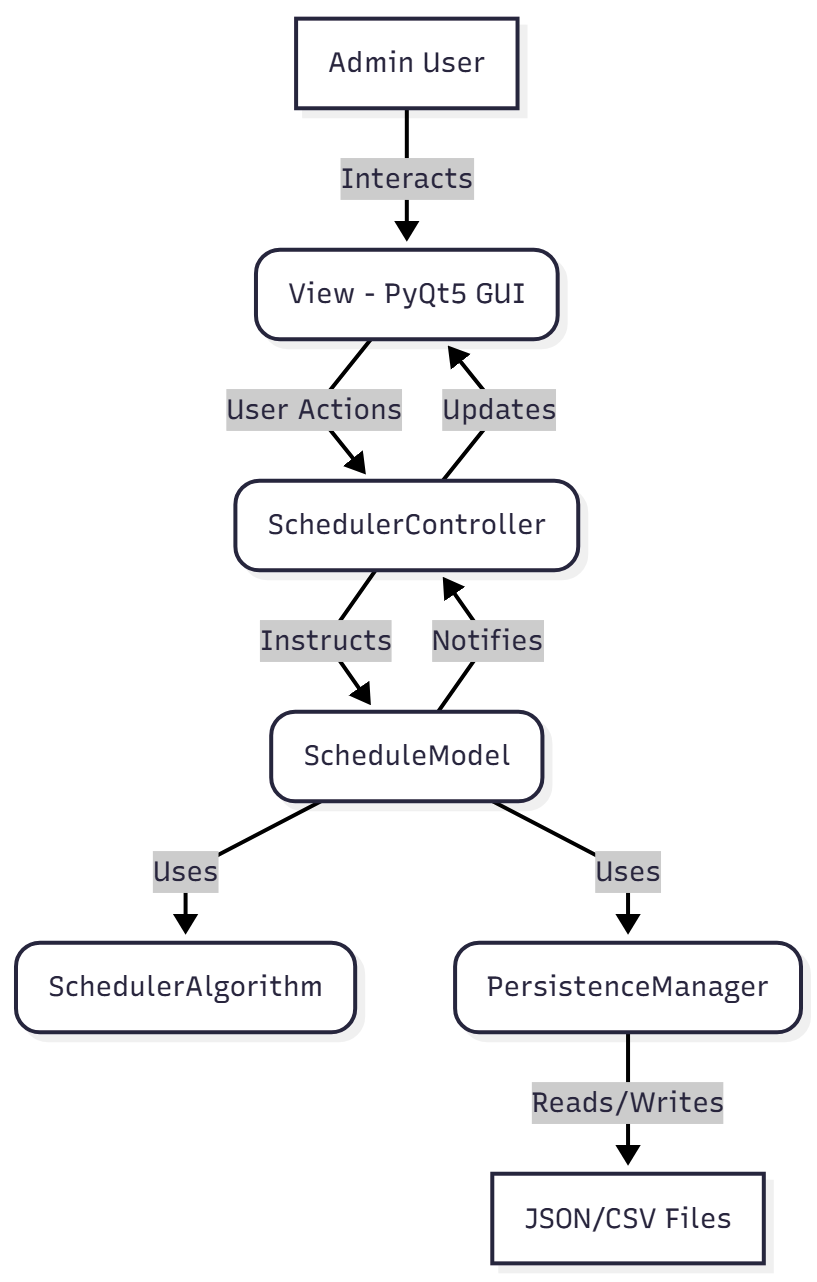
### 2. System Architecture

#### 2.1 Architectural Pattern

A **Model-View-Controller (MVC)** pattern will be used. This pattern is ideal for GUI applications as it separates the data and business logic (Model) from the user interface (View), with a Controller acting as the intermediary.

* **Model:** Contains the core logic. This includes all data entities (Course, Instructor, Room), the data persistence layer (reading/writing JSON/CSV), and the SchedulerAlgorithm itself.
* **View:** The GUI, built with PyQt5. This includes the main window, the TimetableView grid, and the ReportView dialog. It is responsible for displaying data and capturing user input.
* **Controller:** The "glue." It responds to user actions (e.g., "Generate Schedule" button click), instructs the Model to perform operations (e.g., model.run\_scheduler()), and tells the View to update its display with the new data.

#### 2.2 System Architecture Diagram

This diagram shows the high-level interaction between the main components of the application.

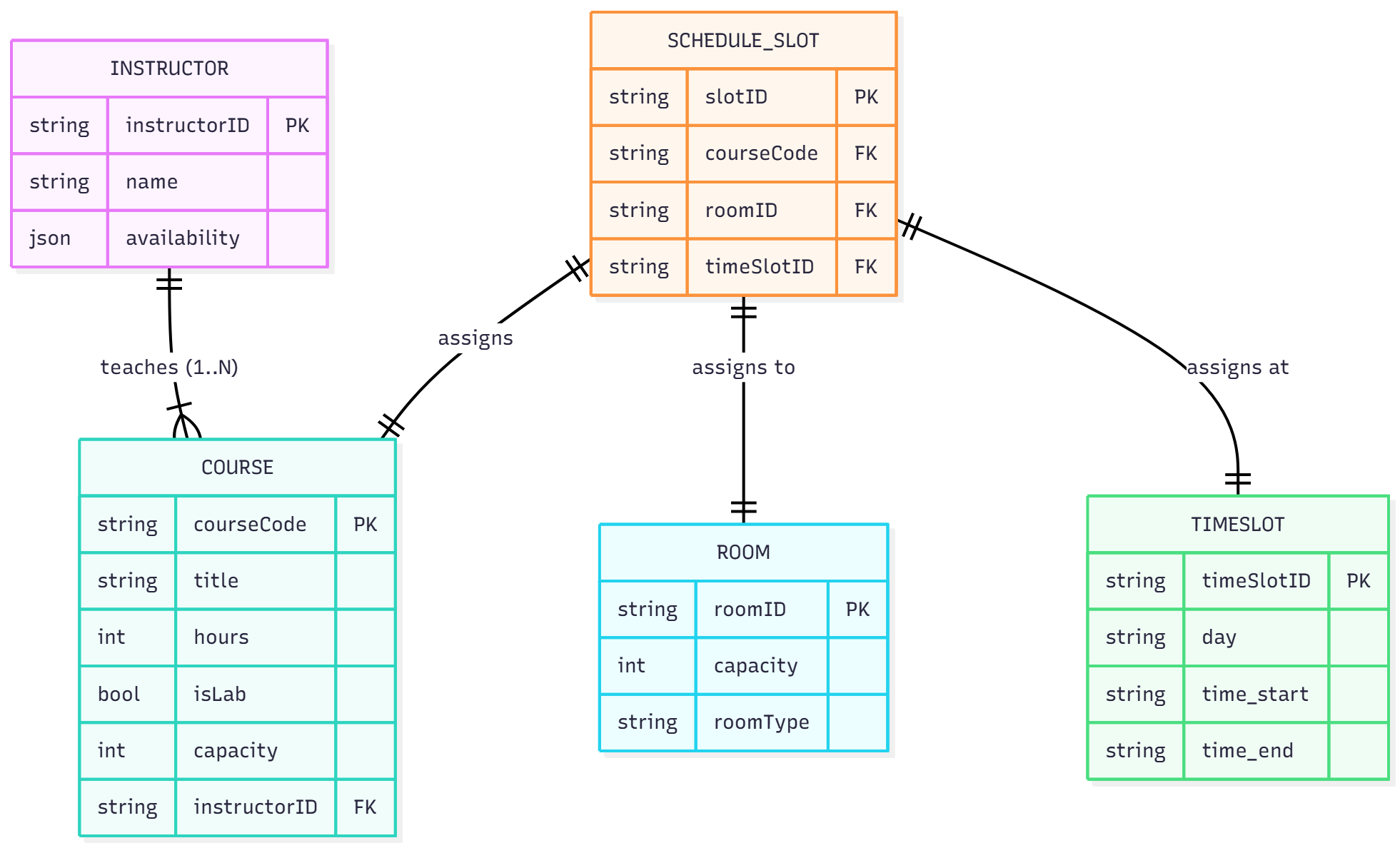
### 3. Data Design

#### 3.1 Data Storage

Data persistence will be file-based, as required. The system will read and write data using JSON, which is structured and easily parsed by Python. The main data entities will be stored in files like courses.json, instructors.json, and rooms.json.

#### 3.2 Entity-Relationship (ER) Diagram

This ERD shows the logical relationships between the core data entities.



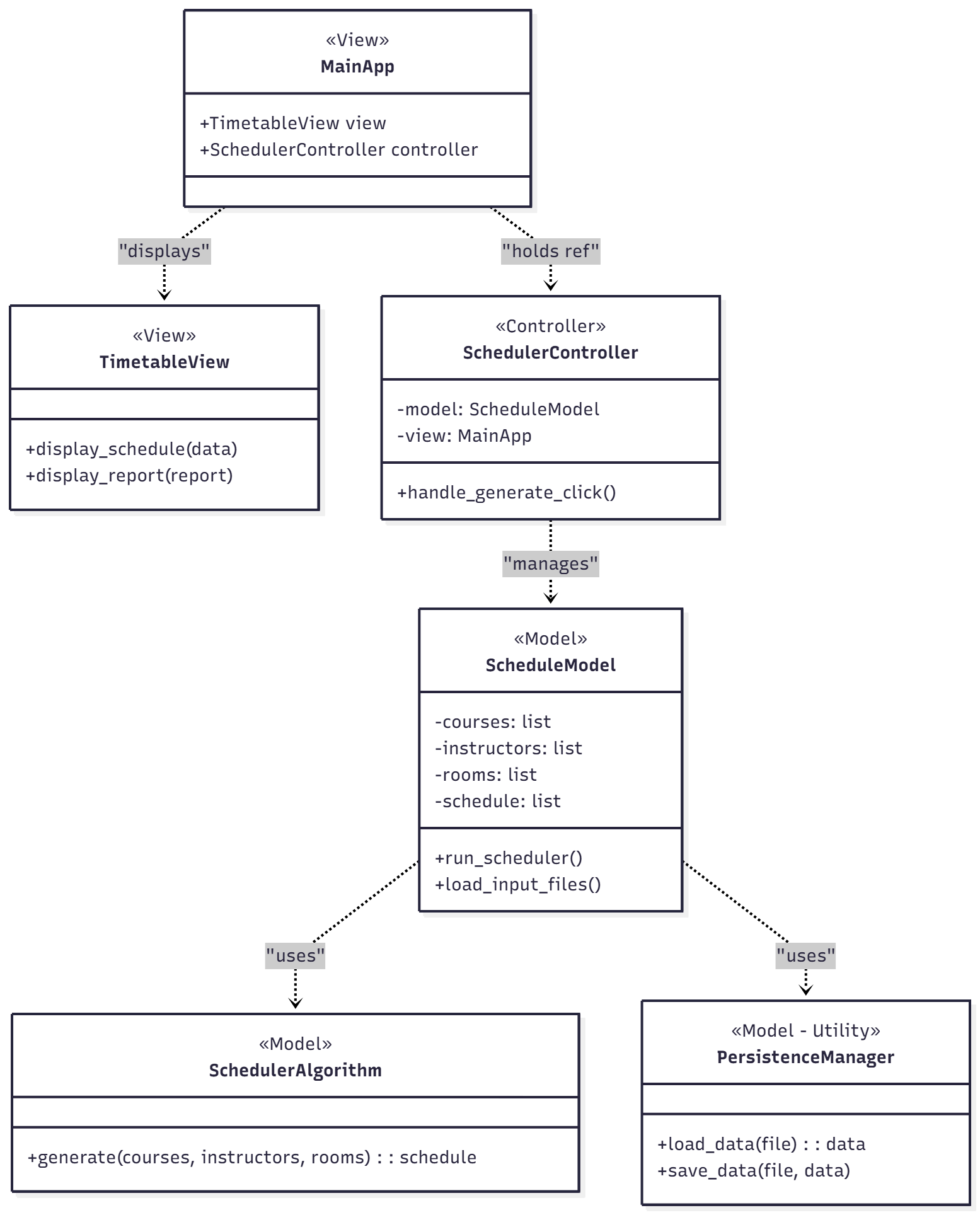
**ERD Explanation:**

* **Instructor:** Has attributes like instructorID, name, and availability (a structured object or JSON string). An Instructor can teach one or many Courses.
* **Course:** The central entity. It has attributes like courseCode, title, hours, isLab (boolean), capacity, and requiresRoomType. It is taught by one Instructor.
* **ScheduleSlot:** Represents a generated assignment. It links a Course to a specific Room and TimeSlot.
* **Room:** Has attributes roomID, capacity, and roomType (e.g., "Lab," "Lecture").
* **TimeSlot:** Represents a block in the schedule (e.g., "Monday 09:00-10:00").

### 4. Class & Component Design

#### 4.1 UML Class Diagram

This diagram details the primary classes for the application, their attributes, methods, and relationships, following the MVC pattern.



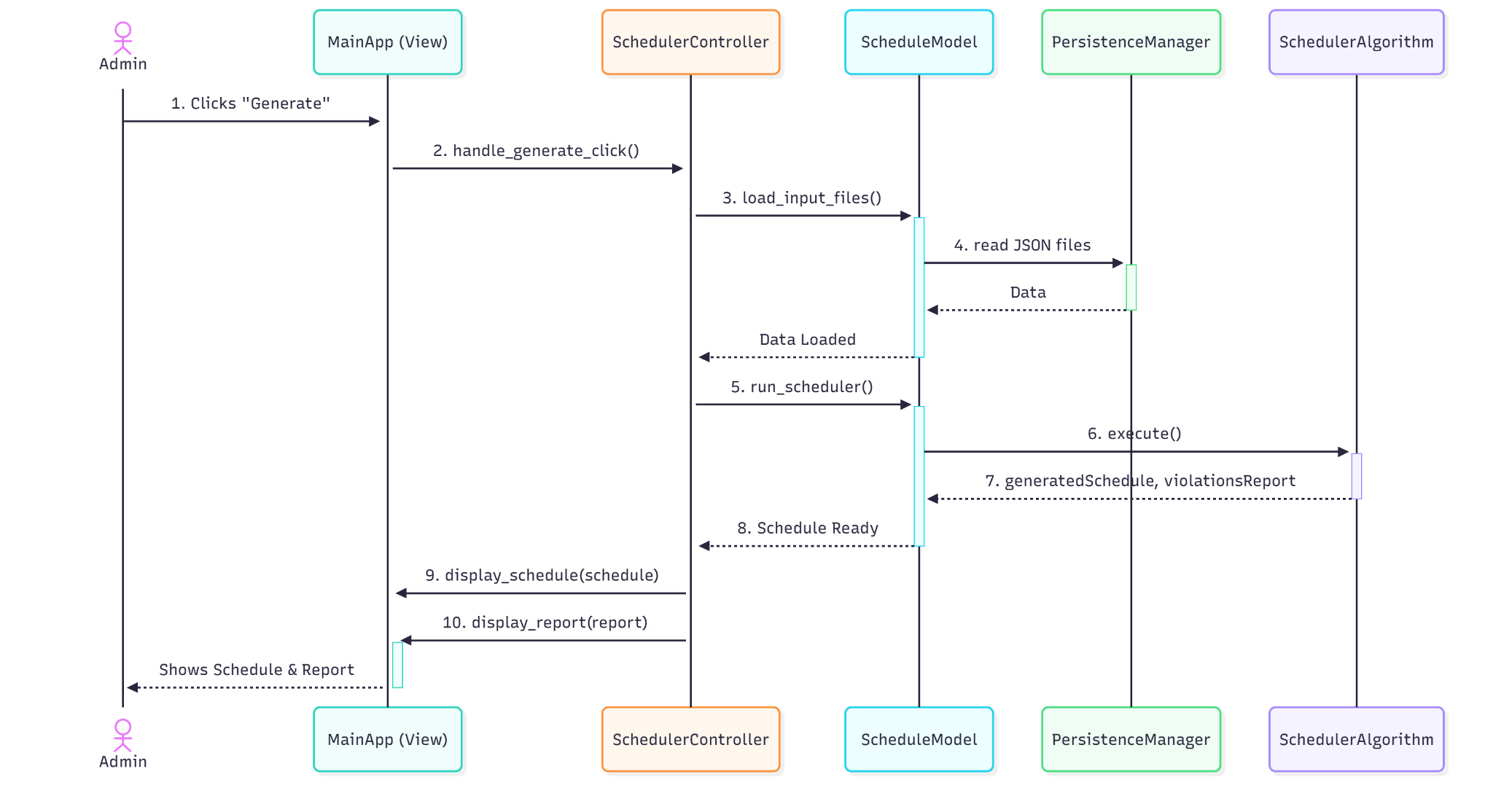
**Class Descriptions:**

* **MainApp (View):** The main QMainWindow. It initializes and holds references to the TimetableView and SchedulerController.
* **TimetableView (View):** A QWidget (likely using QTableWidget) that renders the visual schedule grid.
* **SchedulerController (Controller):** Handles all event logic. It connects GUI signals (like button clicks) to Model functions.
* **ScheduleModel (Model):** Manages all application data (lists of courses, instructors, rooms) and the final generated schedule.
* **SchedulerAlgorithm (Model):** The "brains." Contains the generate() method, which implements the core backtracking or heuristic logic to solve the scheduling constraints.
* **PersistenceManager (Model):** A utility class with static methods like load\_data() and save\_data() to handle JSON file I/O.

### 5. Behavioral Design (Dynamic)

#### 5.1 UML Sequence Diagram: "Generate Schedule" Use Case

This diagram shows the step-by-step flow of events when the administrator clicks the "Generate Schedule" button.



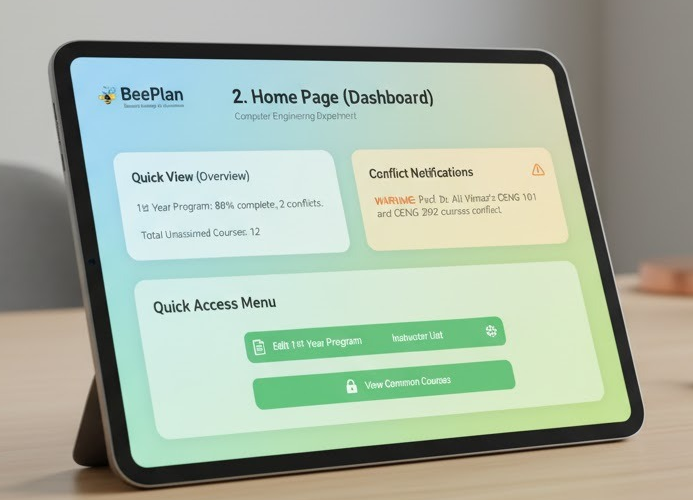
**Sequence Flow:**

1. The AdminUser clicks the "Generate" button on the MainApp (View).
2. MainApp (View) captures this click and calls handle\_generate\_click() on the SchedulerController.
3. The SchedulerController first asks the ScheduleModel to load the latest data (e.g., model.load\_input\_files()).
4. The ScheduleModel uses the PersistenceManager to read the JSON files.
5. After data is loaded, the SchedulerController tells the ScheduleModel to run the algorithm (model.run\_scheduler()).
6. The ScheduleModel instantiates and calls execute() on the SchedulerAlgorithm.
7. The SchedulerAlgorithm performs its logic and returns the generatedSchedule and a violationsReport to the ScheduleModel, which stores them.
8. The SchedulerController retrieves the schedule and report from the ScheduleModel.
9. The SchedulerController calls view.display\_schedule(generatedSchedule) to update the grid.
10. The SchedulerController calls view.display\_report(violationsReport) to show the list of conflicts.

**6. GUI Design**

The GUI for BeePlan was designed entirely using **canva**, leveraging its AI-powered layout generation and refinement capabilities. Gemini provided smart interface suggestions, which were then polished with additional manual adjustments. The goal was to deliver a clean, friendly, and highly intuitive interface tailored for children, parents, and teachers. Below are the main screens produced during the design process.

6.1 Home Page



6.2 Schedule Editor

metin, tablet bilgisayar, ekran görüntüsü, multimedya içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

6.3 Management

metin, tablet bilgisayar, multimedya, İletişim Cihazı içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

**7. AI Tool Evaluation**

* **Canva**
  + **Purpose:** Creating the UML diagrams.
  + **What Went Well:**
    - Had lots of good-looking, pre-built templates for diagrams.
    - The auto-layout and alignment tools made the diagrams look clean and organized.
  + **What Was Difficult:**
    - The AI feature didn't actually *create* the UML diagrams for us; we had to build them all manually, step-by-step.
    - Many of the best templates were locked behind a Pro account.
    - Connecting arrows and lines in UML was tricky and often needed manual fixing.
  + **Verdict:** Good for making diagrams look nice quickly, but it wasn't the "AI" help we expected. It still required a lot of manual work.
* **Figma (with AI)**
  + **Purpose:** Designing the GUI mockups for KidTask.
  + **What Went Well:**
    - The AI feature was incredibly fast. We just described the screens, and it generated them.
    - It was great at creating a consistent look (colors, fonts) across all the different screens.
    - Made prototyping and auto spacing components very easy.
  + **What Was Difficult:**
    - It didn't generate *everything*. We still had to go in and tweak components and add details.
  + **Verdict:** Great for GUI design. It saved a huge amount of time and helped us get a professional looking design to follow.

**8. Reflection**

Using AI tools for this design phase was a real eye opener, and the experience was very different between the two tools.

Figma's AI was the clear winner for us. Being able to just describe the BeePlan app and see it generate complete, accurate screens was a huge time-saver. It handled the boring parts, like making sure the color palette and layout were the same for all user roles, which let us focus on the user flow. The design it gave us felt friendly and usable from the start.

Canva was more of a mixed bag. We used its components to design the BeePlan GUI, and it helped us create a simple, readable interface (with the yellow accent and gray sidebar) that wasn't too cluttered. But when it came to the UML diagrams, its "AI" wasn't what we expected. It's more of a template tool. We couldn't just *describe* the diagram; we had to build it all manually. The free templates were okay, but we spent a lot of time fixing connectors and adjusting layouts to be proper UML.

Overall, we learned that "AI" means different things. Figma's AI felt like a partner that did the heavy lifting for our GUI, while Canva felt more like a traditional (but good) design tool that just had some AI features. Figma's AI is definitely more advanced and was more helpful for our project.