**System Architecture**

**Definition:**  
Developing a decision support system and management tool designed to handle classroom scheduling on a university campus. Our system offers a solution for managing classroom scheduling at the City Campus, detecting conflicts during updates, and handling issues and requests in real-time while taking into account predefined constraints. It also provides a user-friendly interface.

**Data Structure and System Architecture:**  
The system is based on a **Relational Database Architecture**, where the data is organized in tables connected through **Primary Keys** and **Foreign Keys**.

**Why We Chose MySQL and Not MongoDB For example:**  
We selected MySQL because it allows executing complex queries easily, such as finding conflicts, identifying classroom availability, or generating reports in a relatively stable data structure. The data in the scheduling system is structured and requires relationships, which makes MySQL a better fit. As well as the system requires support for complex queries and analysis, where SQL excels. Ensuring data integrity is critical in a system like classroom scheduling, and MySQL is designed for such use cases while **MongoDB** can be used as an addition for other requirements (such as logs), but it is not suitable as the primary database.

**1. Classrooms Table**

| **Column Name** | **Data Type** | **Description** |
| --- | --- | --- |
| classroom\_id | INT (PK) | Unique identifier for the classroom. |
| classroom\_num | INT | Classroom number within the building. |
| floor\_num | INT | Floor number within the building. |
| capacity | INT | Number of seats in the classroom. |
| is\_remote\_learning | BOOLEAN | Whether the classroom supports remote learning (Yes/No). |
| is\_sheltered | BOOLEAN | Whether the classroom is a sheltered space (Yes/No). |
| building\_id | INT (FK) | Unique identifier for the building. |

**2. Buildings Table**

| **Column Name** | **Data Type** | **Description** |
| --- | --- | --- |
| building\_id | INT (PK) | Unique identifier for the building. |
| building\_name | VARCHAR(50) | Name of the building. |
| rooms\_num | VARCHAR(50) | Number of classrooms in the building. |
| capacity | INT | Capacity of the classrooms in the building. |
| remote\_learning | BOOLEAN | Whether the classrooms support remote learning (Yes/No). |

**3. Boards Table**

| **Column Name** | **Data Type** | **Description** |
| --- | --- | --- |
| board\_id | INT (PK) | Unique identifier for the board. |
| board\_size | VARCHAR(20) | Size of the board. |
| classroom\_id | INT (FK) | Unique identifier for the classroom. |

**4. Courses Table**

| **Column Name** | **Data Type** | **Description** |
| --- | --- | --- |
| course\_id | INT (PK) | Unique identifier for the course. |
| course\_name | VARCHAR(100) | Name of the course. |
| students\_num | INT | Number of students enrolled in the course. |
| start\_time | VARCHAR(50) | Weekly start time of the course. |
| end\_time | VARCHAR(50) | Weekly end time of the course. |
| start\_date | DATE | Start date of the course. |
| end\_date | DATE | End date of the course. |
| course\_type | VARCHAR(50) | Course type (e.g., Lecture, Practice, Lab, Seminar). |

**5. Lecturers Table**

| **Column Name** | **Data Type** | **Description** |
| --- | --- | --- |
| lecturer\_id | INT (PK) | Unique identifier for the lecturer. |
| lecturer\_name | VARCHAR(100) | Name of the lecturer. |
| role | VARCHAR(20) | Role of the lecturer (e.g., Lecturer, Assistant). |

**6. Schedules Table**

| **Column Name** | **Data Type** | **Description** |
| --- | --- | --- |
| schedule\_id | INT (PK) | Unique identifier for the schedule. |
| classroom\_id | INT (FK) | Classroom ID (linked to the Classrooms table). |
| course\_id | INT (FK) | Course ID (linked to the Courses table). |
| schedule\_datetime | DATETIME | Date and time of the schedule. |
| status | ENUM | Schedule status (Pending, Confirmed, Conflict). |
| created\_at | DATETIME | Date and time the schedule was created. |
| updated\_at | DATETIME | Date and time the schedule was last updated. |
| time\_start | TIME | Start time of the schedule. |
| time\_end | TIME | End time of the schedule. |

**7. Users Table**

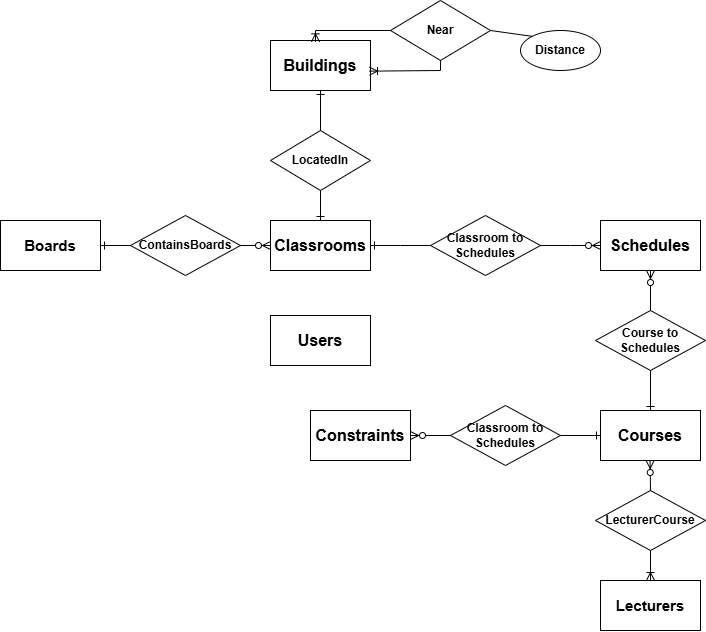
(This table is not directly linked to other tables and is intended for system management and access control.)

| **Column Name** | **Data Type** | **Description** |
| --- | --- | --- |
| user\_id | INT (PK) | Unique identifier for the user. |
| first\_name | VARCHAR(100) | User's first name. |
| last\_name | VARCHAR(100) | User's last name. |
| email | VARCHAR(100) | Email for login. |
| password | VARCHAR(100) | Login password. |
| role | ENUM | User role (e.g., Admin, Manager). |
| permissions | TEXT | User permissions for system access. |

**8. Constraints Table**

| **Column Name** | **Data Type** | **Description** |
| --- | --- | --- |
| constraint\_id | INT (PK) | Unique identifier for the constraint. |
| course\_id | INT (FK) | Course ID associated with the constraint. |
| classroom\_id | INT (FK) | Classroom ID related to the constraint (if applicable). |
| constraint\_type | VARCHAR(50) | Type of constraint (e.g., Capacity, Equipment). |
| constraint\_detail | TEXT | Detailed description of the constraint. |

**Relations between tables:**



**Key Actions on the Client-Side (Front-End):**

* Form for Adding Schedules: Users can fill out a form to add new schedules to the system.
* Weekly Schedule Board View: Displays schedules organized by days and hours in a weekly format.
* Drag-and-Drop Feature for Courses with Instant Updates: Users can drag a course on the board from one slot to another (e.g., from 10:00 AM to 12:00 PM). As soon as the user changes the course's location on the board, the system sends a request to the server to update the new schedule in the database. The system performs immediate checks: Does this change create a conflict, such as overlapping courses or lack of available classrooms? If there is no issue, the update is saved. If a conflict arises, an alert is displayed.
  + - Successful Change: The course is shown in its new location on the board after the user confirms the change.
    - Failed Change: An immediate message is displayed, such as: “The course cannot be moved to this time due to a conflict(with more details).”
* Error Messages for Conflicts: Displays alerts when conflicts or issues are detected.
* Exporting Reports: Generate reports containing data like classroom usage, hours of operation, and statistical reports such as how much students are supposed to be in the campus in a given part of time.

**Key Actions on the Server-Side (Back-End):**

* API for Data Management:
  + POST: Add a classroom or schedule.
  + GET: Retrieve data (schedules, classrooms, schedule status).
  + PUT: Update existing data.
  + DELETE: Remove a classroom or schedule.
* Conflict/Duplication Detection Algorithm: Identifies issues such as a classroom assigned to more than one schedule at the same time.
* Automatic Scheduling Algorithm: Matches classrooms to courses based on constraints like room size, required equipment, and availability.
* Statistical Reports:
  + Calculates room utilization rates.
  + Provides availability insights by building.
  + Forecasts future workloads.

**Explanation of What Each Service Does:**

1. Uploading an Excel File and Saving Data in the Schedules Table:

* Client-Side:  
   The user uploads an Excel file containing scheduling data (classrooms, courses, dates, times, schedule duration, etc.).  
  The system displays a message indicating whether the file was successfully uploaded or if there were issues with its structure.
* Server-Side:
  + The server receives and processes the file.
  + Checks the file structure to ensure the columns match the required format (e.g., classroom\_id, course\_id, date, etc.).
  + Converts the file data into records and saves them in the Schedules table.
  + Each record from the file is stored as a separate row in the Schedules table.

2. Request to Change a Schedule:

* Client-Side:  
  The user is presented with an option to request a schedule change, such as modifying the time or classroom.  
  The user inputs their request along with constraints (e.g., a larger classroom or availability at a specific time).
* Server-Side:
  + The server checks the request against the entered constraints.
  + Identifies conflicts with existing schedules.
  + Checks the availability of classrooms that meet the requirements.
  + If the request is approved: The new schedule is saved in the Schedules table.
  + If the request is denied: A message is displayed to the user explaining the reason for the rejection.

3. Updating the System After Approving the Request:

* Client-Side: The system updates the schedule board and displays the updated information.
* Server-Side: The corresponding record in the Schedules table is updated with the new details.

4. Generating Reports:

* Client-Side:  
  The user selects the desired report type from the available options:
  + Classroom Schedule Report: Lists all schedules for a specific classroom, including dates and times.
  + Course Schedule Report: Lists all schedules for a specific course within a given period.
  + Classroom Utilization Statistics: Shows the amount of time each classroom is in use.

The user can define filters such as start and end dates, specific classrooms, or specific courses.

* Server-Side:
  + The server receives the request and performs queries on the database.
  + Processes the data and generates the report in the requested format.
* Client-Side: The user is given the option to download the report file or view the results in an interactive table.

**Tools and Technologies:**

Server-side Development:

* Visual Studio Code: Code editor for Python and JavaScript development.
* Python with Flask for creating a simple and flexible API.
* MySQL for database management.

Client-side Development:

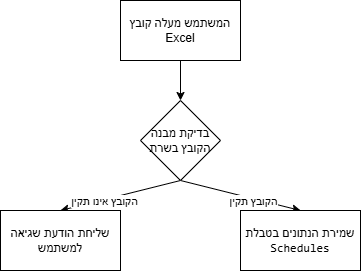
* React for developing an interactive and modular user interface.
* HTML + CSS + JavaScript for designing and displaying the interface.
* FullCalendar: A JavaScript library for building an interactive calendar with drag-and-drop and update options.

**How it works:**

* The user enters data through a form on the client-side using React.
* The data is sent to the Flask server via an API.
* The server stores the data in the MySQL database.
* React retrieves and displays all the schedule entries.
* Project management and collaboration are done through GitHub.

**Process flow in the system:**

Process: Uploading an Excel file:

****

Process: Request for schedule change:

**תמונה שמכילה טקסט, תרשים, קו, צילום מסך

התיאור נוצר באופן אוטומטי**

Process: Checking constraints and updating the system:

**תמונה שמכילה טקסט, תרשים, קו, גופן

התיאור נוצר באופן אוטומטי**

Process: Generating reports:

**תמונה שמכילה טקסט, תרשים, קו

התיאור נוצר באופן אוטומטי**