**Software Design Specifications**

**for**

**<Time Table Generation and Version 2.0>**

Prepared by:

Likhitha: Backend developer

Bhargavi: Frontend developer

Rashmitha: Frontend developer

Rishika: Backend developer

Joshika: Project manager

**Document Information**

|  |  |
| --- | --- |
| **Title:UNISCHED** |  |
| **Project Manager:Joshika** | **Document Version No:2.0** |
|  | **Document Version Date:1-5-25** |
| **Prepared By:Likhitha,rashmitha,**  **bhargavi,rishika,** | **Preparation Date:1-5-25** |
|  |  |
|  |  |
|  |  |

**Version History**

| **Ver. No.** | **Ver. Date** | **Revised By** | **Description** | **Filename** |
| --- | --- | --- | --- | --- |
| 1.0 | 2025-02-10 | Bhargavi, Likhitha | Initial project outline and requirements | timetable\_srs\_v1.docx |
| 1.1 | 2025-03-05 | Rishika, Rashmitha | Added UI and module descriptions | timetable\_srs\_v1\_1.docx |
| 1.2 | 2025-03-25 | Joshika | Updated use cases and added diagrams | timetable\_srs\_v1\_2.docx |
| 1.3 | 2025-04-15 | Likhitha, Rishika | Backend interface and DB model revisions | timetable\_srs\_v1\_3.docx |
| 1.4 | 2025-05-01 | Team All | Finalized complete documentation | timetable\_srs\_final.docx |

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| **1 INTRODUCTION ............................................................................................................................................................** | | **4** |
| 1.1 | PURPOSE ...................................................................................................................................................................... | 4 |
| 1.2 | SCOPE .......................................................................................................................................................................... | 4 |
| 1.3 | DEFINITIONS, ACRONYMS, AND ABBREVIATIONS ........................................................................................................ | 4 |
| 1.4 | REFERENCES ................................................................................................................................................................ | 4 |

|  |  |  |  |
| --- | --- | --- | --- |
| **2** | **USE CASE VIEW ............................................................................................................................................................** | | **4** |
|  | 2.1 | USE CASE ..................................................................................................................................................................... | 4 |
| **3** | **DESIGN OVERVIEW .....................................................................................................................................................** | | **4** |
|  | 3.1 | DESIGN GOALS AND CONSTRAINTS ............................................................................................................................. | 5 |
|  | 3.2 | DESIGN ASSUMPTIONS ................................................................................................................................................. | 5 |
|  | 3.3 | SIGNIFICANT DESIGN PACKAGES ................................................................................................................................. | 5 |
|  | 3.4 | DEPENDENT EXTERNAL INTERFACES ........................................................................................................................... | 5 |
|  | 3.5 | IMPLEMENTED APPLICATION EXTERNAL INTERFACES ................................................................................................. | 5 |
| **4** | **LOGICAL VIEW .............................................................................................................................................................** | | **5** |
|  | 4.1 | DESIGN MODEL ............................................................................................................................................................ | 6 |
|  | 4.2 | USE CASE REALIZATION .............................................................................................................................................. | 6 |
| **5** | **DATA VIEW ....................................................................................................................................................................** | | **6** |
|  | 5.1 | DOMAIN MODEL .......................................................................................................................................................... | 6 |
|  | 5.2 | D ATA MODEL (PERSISTENT DATA VIEW)...................................................................................................................... | 6 |
|  | *5.2.1 Data Dictionary....................................................................................................................................................* | | *6* |
| **6** | **EXCEPTION HANDLING .............................................................................................................................................** | | **6** |
| **7** | **CONFIGURABLE PARAMETERS ..............................................................................................................................** | | **6** |
| **8** | **QUALITY OF SERVICE ................................................................................................................................................** | | **7** |
|  | 8.1 | AVAILABILITY.............................................................................................................................................................. | 7 |
|  | 8.2 | SECURITY AND AUTHORIZATION ................................................................................................................................. | 7 |
|  | 8.3 | LOAD AND PERFORMANCE IMPLICATIONS ................................................................................................................... | 7 |
|  | 8.4 | MONITORING AND CONTROL ....................................................................................................................................... | 7 |

**1 Introduction**

The Smart Timetable Management System is a web-based application developed using Flask as the backend framework and SQLite as the database. It is designed to simplify and streamline the process of accessing academic timetables for students and faculty members. The system provides user-specific timetable views, enabling students to see their schedules based on year and branch, and faculty to access their own teaching hours and availability.

The application follows a modular design, with a clear separation between the frontend and backend. The frontend is built using HTML, CSS, and JavaScript for functionality, with Jinja2 used for dynamic content rendering. The backend is implemented in Python using the Flask framework, which handles routing, logic, and database interactions. Timetable data is stored in an SQLite database (timetable.db), and administrators can generate or reset schedules using provided backend scripts.

This document provides a detailed overview of the software design, including the architecture, data flow, module responsibilities, and user interaction components necessary to understand and extend the system.

**1.1** **Purpose**

The purpose of this Software Design Specification (SDS) is to provide a detailed technical blueprint for the development and understanding of the **Smart Timetable Management System**. This document serves as a bridge between the system requirements and the implementation by outlining the architecture, components, data structures, and interactions that make up the system.

This SDS is part of the overall project documentation and complements other documents such as the requirements specification and user manual. It ensures that developers, testers, and stakeholders have a shared understanding of how the system is structured and how each component is expected to function.

**Intended Audience:**

* **Developers**: To implement the system as per design guidelines.
* **Testers**: To understand system behavior for creating test plans.
* **Project Managers**: To track technical progress and verify design consistency.
* **Future Maintainers**: To maintain, update, or scale the system with minimal rework.

The structure of this document includes detailed design components, such as system architecture, data flow, user interfaces, database schema, and module interactions. It is organized to allow easy navigation and reference for all technical stakeholders involved in the project lifecycle.

**1.2 Scope**

This Software Design Specification applies to the design and development of the **Smart Timetable Management System**, a web-based application that provides academic scheduling features for students, faculty, and administrators.

The document covers the design of components that support:

* **Student functionalities**: Login authentication, viewing personalized timetables based on year and branch, and identifying common or vacant slots.
* **Faculty functionalities**: Login authentication, viewing individual teaching schedules and meeting hours.
* **Admin functionalities**: Managing timetable data, generating schedules using backend scripts, and resetting subject/timetable data via provided utilities.

The system renders all views using Jinja2 templates integrated with Flask, with data managed through an SQLite database. This specification influences all aspects of the project, including UI design, backend logic, database interactions, and role-based access.

**1.3** **Definitions, Acronyms, and Abbreviations**

 **SDS (Software Design Specification)**: A document that outlines the architecture, components, and design choices of the software system.

 **UI (User Interface)**: The part of the system that users interact with, designed using HTML, CSS, and Jinja2 templates.

 **DB (Database)**: A structured storage system used to manage timetable and user data. This project uses SQLite as its database.

 **HTML (HyperText Markup Language)**: A standard language used for creating the structure of web pages.

 **CSS (Cascading Style Sheets)**: A language used to define the style and layout of HTML elements on web pages.

 **Flask**: A lightweight web framework written in Python, used to handle server-side logic, routing, and backend functionalities.

 **Jinja2**: A templating engine used with Flask to generate dynamic HTML content based on backend data.

 **SQLite**: A serverless, file-based relational database used for storing and retrieving data in the Smart Timetable application.

 **CRUD (Create, Read, Update, Delete)**: The four basic operations used to manipulate records in a database.

 **Admin**: A user with elevated privileges responsible for managing timetable generation, subject allocation, and data resets.

 **Student**: A user who logs into the system to view their academic schedule based on their year and branch.

 **Faculty**: A user who accesses the system to view their teaching hours and available time slots for student meetings.

**1.4** **References**

This section provides a list of references and resources used during the design and development of the Smart Timetable Management System. These references serve as important sources of information for understanding frameworks, libraries, and tools utilized within the project.

1. **Flask Official Documentation**  
   URL: https://flask.palletsprojects.com/en/latest/  
   Provides official documentation for the Flask web framework used in the project.
2. **Jinja2 Templating Engine Documentation**  
   URL: https://jinja.palletsprojects.com/en/latest/  
   Reference for Jinja2 syntax and functionalities used for dynamic HTML rendering.
3. **SQLite Documentation**  
   URL: https://www.sqlite.org/docs.html  
   Provides information for working with SQLite databases, including schema definitions and SQL queries.
4. **Python Official Documentation**  
   URL: <https://docs.python.org/3/>  
   Contains general Python language features, syntax, and libraries used in backend development.
5. **Smart Timetable Project Source Files**  
   Location: Smart-timetable project directory (local repository or provided archive).  
   Contains all core project files referenced by this document, including source code, templates, database scripts, and configuration files.

**2 Use Case View**

The use case view presents the core functional interactions between the users (actors) and the system. It identifies the main scenarios through which the **Web Utility for Timetable Generation** provides value.

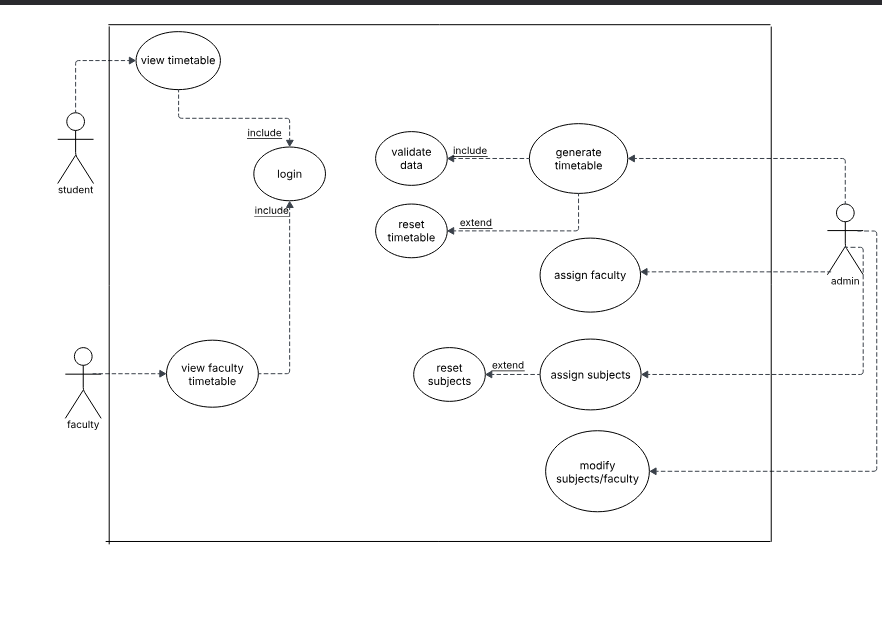
**Actors**

* **Admin**: Manages the system, including faculty and subject assignments, and generates the timetable.
* **Faculty**: Views their individual timetables.
* **Student**: Views their class-wise timetable.

**Use Cases**

* **UC1: Assign Subjects**  
  Admin assigns subjects to departments or courses.
* **UC2: Assign Faculty**  
  Admin maps faculty members to assigned subjects.
* **UC3: Modify Student/Faculty**  
  Admin updates the database of students and faculty (add/edit/remove records).
* **UC4: Generate Timetable**  
  Admin initiates automatic timetable creation based on input constraints and availability.
* **UC5: View Timetable (Student)**  
  Students log in and view their semester-wise timetable.
* **UC6: View Timetable (Faculty)**  
  Faculty members log in to view their weekly lecture schedule.

**2.1** **Use Case**

****

1. **Design Overview**

This section provides a high-level overview of the software design of the **Smart Timetable Management System**. The system is developed as a web-based application using a modular architecture that separates the frontend, backend, and database layers for maintainability and scalability.

The software design complies with the initial requirements of the system, including user role separation, timetable rendering, and data management capabilities. It is guided by the principles of clean separation between interface logic (HTML, CSS, Jinja2 templates) and backend operations (Flask with Python and SQLite).

The design is organized into three core modules:

* **Frontend Interface Module**: Built using HTML and CSS, with Flask-integrated Jinja2 templates to dynamically render timetable views based on user roles and database data.
* **Backend Logic Module**: Implemented using Flask in Python, this module manages routing, request handling, form submissions, and controls the data flow between the frontend and the SQLite database.
* **Database Module**: Uses SQLite to store structured data such as user credentials, subject lists, branch and year mappings, and timetables. This module supports CRUD operations and is initialized/reset using helper scripts.

The system design ensures that each module interacts through well-defined contracts, with templates receiving context data from Flask routes, and route logic interacting with the database through SQL queries. This layered approach supports role-based access and modular extensions.

**3.1** **Design Goals and Constraints**

This section outlines the fundamental assumptions made during the design and development of the **Smart Timetable** system. These assumptions influence architectural decisions, technology choices, and system behavior.

1. **User Environment**  
   It is assumed that users will access the application through modern desktop web browsers such as Google Chrome, Mozilla Firefox, or Microsoft Edge. Mobile responsiveness is not a priority in the current version.
2. **Deployment Platform**  
   The application is designed to be deployed on a local machine or a basic cloud server running Python 3.8+ with Flask. The system assumes a Unix-based or Windows operating environment with standard Python support.
3. **Institution-Specific Usage**  
   The system is intended for use by a single educational institution at a time (e.g., one school or college). Multi-institution or multi-tenant deployment is not supported in this version.
4. **Database Simplicity and Size**  
   The application uses SQLite as the backend database, under the assumption that the total volume of data will be relatively small. The system is not designed to support high-volume, concurrent access or large datasets. Migration to more scalable databases (e.g., MySQL, PostgreSQL) would require significant architectural changes.
5. **Manual Data Initialization**  
   It is assumed that course details, subjects, faculty members, and constraints are entered manually through the interface or preloaded. The timetable generation algorithm relies on having a complete and conflict-free dataset before execution.
6. **Single-User Administrative Access**  
   The system assumes that a single administrator (or a small number of sequential users) will be using the system for managing and generating timetables. Concurrent user support and access control are not part of the current implementation.
7. **Trusted Environment**  
   The system does not implement authentication, authorization, or encryption. It is assumed that the application will be used in a secure, trusted environment (e.g., local network, internal server).
8. **Feasible Constraint Handling**  
   The timetable generation algorithm presumes that the provided inputs and constraints will lead to at least one valid solution. If the constraints are too strict or incomplete, the system may fail to generate a schedule without detailed feedback.
9. **Third-Party Dependency Availability**  
   It is assumed that all required Python packages and dependencies (e.g., Flask) will be available and compatible throughout the application's lifecycle. Internet access may be required during initial setup to install dependencies.

**3.2** **Design Assumptions**

This section outlines the key assumptions made during the software design process, which have significant impacts on the architecture, functionality, and implementation of the system.

1. **User Roles and Permissions:**
   * The system will have **three types of users**:
     + **Students** will have access to their own timetable, course registration, and class schedules.
     + **Faculty** will have access to their teaching schedules, meeting hours, and assigned classes.
     + **Admin** will have full control over the system, including user management, timetable configurations, and subject allocations.
2. **Timetable Data Integrity:**
   * It is assumed that the timetable data (e.g., class schedules, course offerings, faculty assignments) will be pre-entered and maintained by authorized administrators.
   * Changes to the timetable will be handled by the **admin** through the backend system, with real-time updates reflected in user views.
3. **Course Enrollment and Student Timetables:**
   * **Course enrollment** is assumed to be handled externally (e.g., via an academic registration system) and will reflect in the timetable once students select their courses.
   * **Timetables for students** will be auto-generated based on their course selections and academic year, with no manual input required by students.
4. **Classroom and Faculty Availability:**
   * The system assumes **faculty schedules** will be predefined and conflict-free.
   * **Classroom availability** will be managed to avoid any scheduling conflicts, ensuring that classrooms are not double-booked.
5. **Scalability and Load Handling:**
   * The system is assumed to support **up to a certain number of users** (e.g., 1000 students and faculty) simultaneously, with minimal latency in timetable generation and updates.
   * The system’s backend will be capable of handling multiple concurrent requests efficiently without performance degradation.
6. **Real-Time Timetable Updates:**
   * **Timetable updates** (e.g., class cancellations, faculty changes) will be reflected in real-time for all users.
   * The system assumes that any updates or modifications to class timings will be immediately available to students and faculty upon login.
7. **Security and Authentication:**
   * The system will use **secure login authentication** (e.g., via username and password) for both students and faculty.
   * Sensitive data (e.g., student records, faculty schedules) will be stored securely and transmitted using **secure protocols** (e.g., HTTPS).
8. **User Interface and Mobile Compatibility:**
   * The primary interface will be **desktop-centric**, though the system will be designed to be **responsive** for mobile use as needed.
   * Users will be able to access their timetables through web browsers on desktop or mobile devices.
9. **Frontend-Backend Communication:**
   * The frontend (HTML, CSS, JavaScript) will interact with the backend (Flask, SQLite) through **API calls**, such as fetching timetable data, checking course availability, and updating schedules.
   * **AJAX requests** will be used to ensure smooth, asynchronous communication between the client and the server.
10. **Admin and Faculty Access Control:**
    * **Admin users** will be able to manage all timetable configurations, including adding/removing courses, assigning faculty to classes, and handling course conflicts.
    * **Faculty** will only have access to their own teaching schedule and will not be able to modify any data outside of their assignments.
11. **No Notification System:**
    * The system will **not include any notification functionality** for updates, class cancellations, or schedule changes. Students and faculty will need to manually check the system for any modifications.
12. **Error Handling and User Feedback:**
    * The system is assumed to handle **errors gracefully**, providing meaningful error messages for invalid actions or failed operations (e.g., login failures, timetable generation errors).
    * **User feedback** will be provided via pop-ups or notifications in the interface when errors or conflicts occur.
13. **Data Validation and Consistency:**
    * All data entered into the system (e.g., course information, faculty assignments) will be **validated** to ensure consistency, and any inconsistencies will be flagged before they can affect the timetable generation.
14. **Offline Data Storage:**
    * **Offline access** is not considered in the design. The system will assume an **active internet connection** is required for users to access and view their timetables.
15. **Long-Term Maintenance:**
    * The system assumes that the data (courses, faculty assignments, timetables) will be maintained by the **academic administration** regularly, with updates added based on new course offerings or faculty changes each semester.

**3.3** **Significant Design Packages**

**. Presentation Layer**

* **Directory**: templates/, static/
* **Purpose**: Handles the user interface (UI) and front-end interaction.
* **Technologies Used**: HTML, CSS, JavaScript, Jinja2 (for Flask templating)
* **Responsibilities**:
  + Render web pages using HTML templates.
  + Display generated timetable and form interfaces for user input.
  + Provide static assets (e.g., stylesheets, scripts).

**2. Application Layer (Controller)**

* **File**: app.py
* **Purpose**: Acts as the central control hub for handling HTTP requests and coordinating between the UI and business logic.
* **Technologies Used**: Flask
* **Responsibilities**:
  + Define application routes (URLs).
  + Receive form submissions (e.g., subject input).
  + Invoke utility functions and render results to the user.

**3. Business Logic Layer**

* **Directory**: utils/
* **Key Files**:
  + generate\_timetable.py: Core logic for creating a valid timetable based on input constraints.
  + **init\_db.py**: Script for initializing database tables with required schema.
* **Responsibilities**:
  + Implement the scheduling algorithm.
  + Enforce constraints such as subject-faculty mapping and time slot availability.
  + Prepare and update data models used by the application layer.

**4. Data Layer**

* **File**: **timetable.db**
* **Purpose**: Stores persistent data required by the system.
* **Technology Used**: SQLite
* **Responsibilities**:
  + Store information about subjects, timetables, and faculty.
  + Provide structured access to data for timetable generation and display.
  + Enable read/write operations from the business logic and controller.

**5. Utility Scripts**

* **Files**: **reset\_subjects.py, reset\_timetable.py**
* **Purpose**: Provide administrative tools for resetting or reinitializing application data.
* **Responsibilities**:
  + Clear or reset specific tables in the database.
  + Support development and testing scenarios by providing quick data refresh.

**3.4** **Dependent External Interfaces**

The table below lists the external interfaces this system depends on. These interfaces may be modules, libraries, or services used by internal components of the timetable generation utility.

| **External Application and Interface Name** | **Module Using the Interface** | **Functionality / Description** |
| --- | --- | --- |
| **MySQL Database (JDBC Interface)** | Backend Module (Likhitha, Rishika) | Used to store and retrieve timetable data, faculty info, subjects, and classroom allocations. Enables CRUD operations on all scheduling data. |
| **Email Notification Service (SMTP API)** | Notification Module (Backend) | Sends confirmation or update emails to faculty once the timetable is finalized or modified. |
| **Bootstrap / Tailwind CSS** | Frontend Module (Bhargavi, Rashmitha) | Used to design a responsive and user-friendly web interface for interacting with the system. |
| **JavaScript Libraries (e.g., FullCalendar.js)** | Frontend Module | Supports the dynamic calendar view where users can visualize and interact with the generated timetable. |
| **Authentication Library (e.g., Firebase/Auth or JWT)** | Backend Auth Module | Provides user login functionality and secure session handling for admin and faculty users. |

**3.5** **Implemented Application External Interfaces (and SOA webservices)**

The following table lists the interfaces provided by the timetable generation system for use by external applications or modules. These interfaces are implemented by internal components of the system and are exposed for integration or functionality extension.

| Interface Name | Module Implementing the Interface | Functionality / Description |
| --- | --- | --- |
| REST API – Timetable Data | Backend Controller (Likhitha, Rishika) | Provides endpoints for retrieving, creating, updating, and deleting timetable records in JSON format. |
| REST API – Faculty Info | Backend Controller | Exposes faculty details to external or frontend modules for allocation logic and display. |
| REST API – Room Allocation | Scheduling Module | Handles API calls to validate and assign classrooms to time slots based on availability. |
| Web UI Interface | Frontend Module (Bhargavi, Rashmitha) | Implements the user-facing interface to interact with the system using responsive web pages. |
| Authentication Endpoint | Auth Module (Backend) | Provides login/logout functionality with session token support for admin and faculty users. |

1. **Logical View**

This section outlines the detailed design of the Web Utility for Timetable Generation system. It follows a layered architecture to separate concerns and improve maintainability. The system is divided into the **Frontend**, **Backend**, and **Database** layers. Interactions between modules follow the MVC (Model-View-Controller) design pattern.

**4.1** **Design Model**

The system is decomposed into the following modules and classes:

**Frontend Module (Developed by Bhargavi and Rashmitha)**

* **Classes/Components**:
  + LoginComponent: Manages user login input and API integration.
  + DashboardComponent: Displays an overview of options for admin and users.
  + TimetableFormComponent: Gathers inputs such as faculty, subjects, room availability, and constraints.
  + ViewTimetableComponent: Displays generated timetable in table/grid format.

**Backend Module (Developed by Likhitha and Rishika)**

* **Controllers**:
  + AuthController: Manages login and session control.
  + TimetableController: Orchestrates timetable generation logic.
  + FacultyController: Handles CRUD operations for faculty.
  + RoomController: Manages room data.
* **Services**:
  + SchedulerService: Core logic for generating conflict-free timetables.
  + ValidationService: Ensures input constraints are valid.
* **Models**:
  + User, Faculty, Subject, Room, Timetable: Represent database entities.

**Class Diagram Summary:**

A high-level UML class diagram would include the above classes, with:

* TimetableController using SchedulerService
* SchedulerService depending on Faculty, Room, and Subject entities
* Timetable being the output model stored and retrieved from the database

**4.2** **Use Case Realization**

This section describes the detailed realization of the major use cases identified in the system. Each use case outlines how different components/modules interact to fulfill the system’s functionality.

**UC1: Assign Subjects (Admin)**

**Actors**: Admin  
**Flow**:

1. Admin logs into the system.
2. Navigates to "Assign Subjects" panel.
3. Selects course/year and adds subject details.
4. Submits form to backend.
5. Backend validates and updates the subject database.

**Modules Involved**:

* Frontend UI
* Subject Controller
* Subject Database Model

**UC2: Assign Faculty (Admin)**

**Actors**: Admin  
**Flow**:

1. Admin selects department/year/subject.
2. Picks a faculty member to assign.
3. Data sent to backend.
4. System checks for faculty conflicts.
5. If no conflict, assignment is stored in the database.

**Modules Involved**:

* Faculty Assignment UI
* Faculty-Service Module
* Conflict Checker
* Database Layer

**UC3: Generate Timetable (Admin)**

**Actors**: Admin  
**Activity Diagram Overview**:

* **Start** → Enter Course Details → Allocate Faculty → Generate Timetable  
  → If **conflicts occur** → Back to Enter Course Details  
  → Else → Publish Timetable → **End**

**Sequence**:

1. Admin inputs constraints (courses, faculties, rooms).
2. Timetable generation algorithm is triggered.
3. Conflict resolver verifies schedule.
4. Timetable saved to database.
5. Timetable published for students and faculty.

**Modules Involved**:

* Generation Engine (Algorithm)
* Conflict Detection Module
* Timetable DB Handler
* Notification/Publishing Module

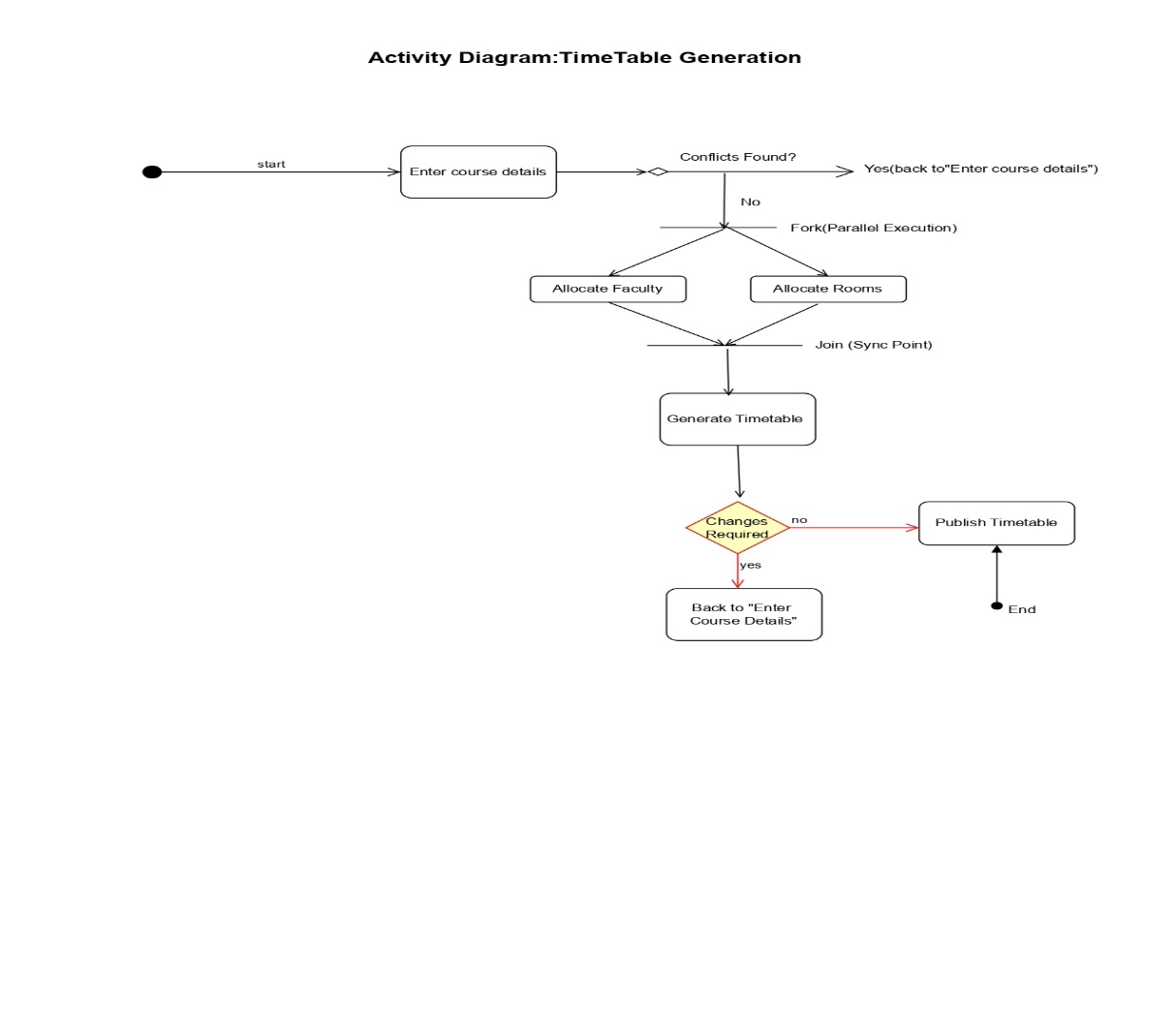
**UC4: View Timetable (Student / Faculty)**

**Actors**: Student, Faculty  
**Flow**:

1. User logs into the system.
2. Frontend displays “View Timetable” option.
3. User selects relevant filters (semester/faculty name).
4. Timetable fetched from backend.
5. Displayed in tabular format.

**Modules Involved**:

* UI Renderer
* Timetable Fetch API
* Timetable Display Component

****

**5 Data View**

The **Data View** describes the data structure and persistent storage needed for the Timetable Generation system. This system primarily stores course details, faculty schedules, generated timetables, and conflict data.

**5.1 Key Data Entities**

* **Course**: Stores course details like ID, name, duration, and times.
* **Faculty**: Stores faculty details, including availability and assigned courses.
* **Timetable**: Stores final timetable entries with course, faculty..
* **Conflict**: Tracks conflicts like scheduling overlaps or double-booked faculty.

**5.2 Data Storage**

* **Relational Database**: Data will be stored in tables (e.g., Courses, Faculty, Timetable, Conflicts).
* **Relationships**:
  + One-to-many: Courses to Timetable.
  + Many-to-many: Faculty to Courses.
  + Timetable to Conflict (for scheduling issues).

**5.3 Data Operations**

* **CRUD Operations**:
  + **Create**: New courses, faculty, and timetables.
  + **Read**: Retrieve course schedules, faculty availability.
  + **Update**: Modify timetable or resolve conflicts.
  + **Delete**: Remove canceled courses or unavailable faculty.

**5.4 Data Integrity and Security**

* **Data Validation**: Ensures consistency (e.g., no overlapping courses).
* **Backup and Recovery**: Periodic backups to prevent data loss and ensure recovery.

]

**5.1** **Domain Model**

The Domain Model represents the key entities in the Timetable Generation system, showing their attributes and relationships. The model highlights the persistent data entities involved in the system and how they relate to each other.

Entities and Their Relationships

1. Course
   * Attributes:
     + CourseID: Unique identifier for the course.
     + CourseName: Name of the course.
     + Duration: Duration of the course (in hours).
2. Faculty
   * Attributes:
     + FacultyID: Unique identifier for the faculty.
     + FacultyName: Name of the faculty member.

AssignedCourses: List of courses assigned to the faculty.

1. Timetable
   * Attributes:
     + TimetableID: Unique identifier for the timetable.
     + CourseID: Foreign key linking to the Course entity.
     + FacultyID: Foreign key linking to the Faculty entity.
     + TimeSlot: Assigned time slot for the course.
2. Conflict
   * Attributes:
     + ConflictID: Unique identifier for a conflict.
     + CourseID: The course involved in the conflict.
     + FacultyID: The faculty member involved in the conflict.
     + ConflictType: Type of conflict (e.g., overlapping schedule, unavailable faculty).

Relationships

* Course - Timetable:
  + One-to-many relationship. A course can appear in multiple timetables (for different sections or semesters).
* Faculty - Timetable:
  + Many-to-many relationship. A faculty member can be assigned to multiple courses, and each course can have multiple faculty members (depending on the course size and needs).
* Course - Conflict:
  + One-to-many relationship. A course may have multiple conflicts (overlapping with other courses or faculty availability issues).
* Faculty - Conflict:
  + One-to-many relationship. A faculty member may be involved in multiple conflicts (e.g., double-booked for two courses).

**5.2** **Data Model (persistent data view)**

The **Data Model** defines the structure of the persistent data stored in the system, including the entities, their attributes, and the relationships between them. This view focuses on how the system's data is stored, retrieved, and updated, ensuring data integrity and consistency across the application.

**5.2.1** **Data Dictionary**

The following table lists the data elements (entities, attributes, relationships) used in the system, along with their descriptions.

| Entity | Attribute | Description |
| --- | --- | --- |
| Course | CourseID | A unique identifier for the course. |
|  | CourseName | The name of the course being offered. |
|  | Duration | The duration of the course in hours. |
|  | TimeSlot | The preferred or assigned time slot for the course. |
|  | Room | The classroom or lab assigned to the course. |
| Faculty | FacultyID | A unique identifier for the faculty member. |
|  | FacultyName | The name of the faculty member. |
|  | Availability | The list of time slots when the faculty member is available to teach. |
|  | AssignedCourses | The list of courses assigned to the faculty member. |
| Timetable | TimetableID | A unique identifier for the timetable entry. |
|  | CourseID | A foreign key that links to the associated Course entity. |
|  | FacultyID | A foreign key that links to the associated Faculty entity. |
|  | Room | The assigned room for the course in the timetable. |
|  | TimeSlot | The assigned time slot for the course in the timetable. |
| Conflict | ConflictID | A unique identifier for the conflict. |
|  | CourseID | A foreign key that links to the associated Course entity involved in the conflict. |
|  | FacultyID | A foreign key that links to the associated Faculty entity involved in the conflict. |
|  | ConflictType | The type of conflict, such as double-booked course or unavailable faculty. |

5.2.2 Relationships

| Relationship | Description |
| --- | --- |
| Course - Timetable | A one-to-many relationship where one course can appear in multiple timetables. |
| Faculty - Timetable | A many-to-many relationship, where a faculty member can be assigned to multiple courses and each course may have multiple faculty members. |
| Course - Conflict | A one-to-many relationship where a course may have multiple conflicts. |
| Faculty - Conflict | A one-to-many relationship where a faculty member may be involved in multiple conflicts. |

This Data Dictionary captures the necessary entities, their attributes, and relationships for the Timetable Generation system. It ensures the system's data structure is clearly defined and helps developers manage the persistent data used throughout the application.

**6 Exception Handling**

To ensure system robustness, maintain user trust, and minimize disruptions during operation, this application employs a layered and categorized approach to exception handling. Exceptions are handled gracefully at various levels of the application: **input validation**, **business logic**, and **database interaction**.

**1. Exception Categories**

1. **Input-Level Exceptions**
   * **Examples**: Missing or invalid fields during course or faculty input.
   * **Handling**:
     + Client-side form validation (HTML5 + JavaScript).
     + Server-side validation to catch tampered or malformed data.
     + Error messages shown near respective fields, preventing form submission.
2. **Business Logic Exceptions**
   * **Examples**: Course-faculty assignment conflicts, duplicate allocations, schedule overlaps.
   * **Handling**:
     + Rules are pre-validated before execution (e.g., checking availability).
     + If conflict arises, user is redirected to reassign or resolve issues.
     + Users receive contextual prompts like: “Faculty X is already scheduled at this time.”
3. **Database Exceptions**
   * **Examples**: Connection timeout, failed query, record not found.
   * **Handling**:
     + Try-catch blocks around DB operations.
     + Error logs are created using a logging tool (e.g., Python’s logging module or Java's log4j).
     + End-user receives a message: “Something went wrong while saving. Please try again.”
4. **System Exceptions**
   * **Examples**: Unhandled exceptions, resource limits, unexpected server crash.
   * **Handling**:
     + Global exception handler implemented at framework level (e.g., Express middleware in Node.js or Django middleware).
     + Users are redirected to a generic error page with a session-safe retry mechanism.
     + Admins are notified via email/slack for critical issues.

**2. Logging & Monitoring**

* All exceptions are logged with timestamp, user session, and error trace.
* Logs are stored in structured formats (JSON, CSV, or database logs).
* Monitoring tools like **Sentry**, **LogRocket**, or **ELK Stack** can be integrated.

**3. User Notifications**

* Messages avoid technical jargon.
* Errors are explained clearly with corrective instructions.
  + Example: “Please select a faculty member available during the chosen slot.”
* Success and error feedback use modal alerts or toast messages for minimal disruption.

**4. Recovery Measures**

* **Auto-Save**: Partial data is saved periodically to prevent loss.
* **Conflict Resolution**: If timetable generation fails due to conflict, the user is guided back to the input form.
* **Retry Option**: Critical failures (like DB errors) offer a “Try Again” option

**7 Configurable Parameters**

This table describes the simple configurable parameters (name/value pairs) used in the system.

| **Configuration Parameter Name** | **Definition and Usage** | **Dynamic?** |
| --- | --- | --- |
| faculty.course.limit | Maximum number of lecture sessions a faculty member can be assigned per day | Yes |
| generation.mode | Determines if the timetable is generated automatically or manually | Yes |
| db.connection.string | Connection string for accessing the backend database | No |
| session.timeout.minutes | Duration of user inactivity before session timeout | Yes |
| log.level | Level of logging detail (e.g., DEBUG, INFO, ERROR) | Yes |
| admin.contact.email | Email address where system notifications or error reports are sent | Yes |
| timetable.time.slots | List of valid time slots for scheduling classes | Yes |

**8 Quality of Service**

This section outlines the key quality attributes essential for the reliable and secure operation of the timetable generation system.

**8.1 Availability**

The system is designed to be available 24/7 during the academic semester, with scheduled maintenance windows announced in advance. Failover handling is minimal but planned for in case of server or network downtime.

**8.2 Security**

User authentication is required for all faculty and admin users. Role-based access controls are implemented to restrict access to administrative functions. All sensitive data (e.g., login credentials) are stored securely using encryption and protected channels.

**8.3 Performance**

The system responds to most user actions, including timetable generation and data updates, within 2–3 seconds. Timetable generation for an average department (30+ courses) completes in under 10 seconds under normal load.

**8.4 Monitoring and Control**

Error logs and user activity logs are maintained to support debugging and usage tracking. The system sends notifications to the administrator in case of repeated failures, scheduling conflicts, or system anomalies.

**8.1** **Availability**

The timetable generation system is expected to be available throughout the academic term with minimal downtime. It is designed as a web-based application hosted on a reliable server infrastructure to ensure continuous access for faculty and administrators.

Regular maintenance activities such as database backups, system updates, and data cleanup are scheduled during off-peak hours to avoid disrupting normal usage. Any major data loads, such as bulk upload of course or faculty details, are planned during these maintenance windows to preserve system availability.

The design avoids any long-running or blocking processes during working hours to prevent performance bottlenecks that could reduce availability. Additionally, all user inputs are validated to minimize system crashes caused by invalid data or unexpected interactions

**8.2** **Security and Authorization**

The system restricts access based on user roles to ensure that only authorized users can access or modify sensitive data. There are three primary roles in the system: Admin, Faculty, and Viewer. Admins have full access, including the ability to create, edit, and publish timetables. Faculty members can view and request changes relevant to their schedules. Viewers can only view the final timetable.

User authentication is handled through secure login with password protection. Role-based authorization controls are implemented to ensure that users can only perform actions permitted by their roles.

All data transmissions between the user and the server are encrypted using HTTPS. The system maintains a user access log to record login attempts, actions performed, and changes made, supporting auditing and traceability.

User access rights can be managed through an admin dashboard, allowing administrators to add or remove users and assign appropriate roles.

**8.3** **Load and Performance Implications**

The timetable generation system is expected to support multiple concurrent users, especially during peak usage periods such as semester planning. The system is designed to handle up to 100 simultaneous users, with expected business transactions including course entry, faculty allocation, and timetable generation.

Database queries are optimized to ensure timetable generation completes within 2–3 seconds under normal load. Indexing is used on key tables like courses, faculty, and schedules to support fast read/write operations. The backend is designed using asynchronous operations for non-blocking processing during bulk operations, like timetable publishing.

Load testing will simulate real-time usage with concurrent requests to validate system stability and responsiveness. Performance metrics such as response time, server CPU/memory utilization, and query execution time will be monitored.

Scalability is supported via modular backend components that allow for horizontal scaling if user volume increases significantly.

**8.4** **Monitoring and Control**

The application implements monitoring through server-side logging and status tracking mechanisms. Key backend processes such as timetable generation, faculty allocation, and conflict resolution are instrumented with logging hooks to track execution time, error states, and success/failure counts.

Monitoring tools (e.g., Node.js logging with Winston or PM2 monitoring) provide real-time feedback on application health. The following measurable values are monitored:

* API response times
* Number of timetable generation requests per hour
* Error logs and conflict resolution occurrences
* Server resource usage (CPU, memory)

Administrators have access to a control dashboard where they can view logs, system status, and restart services if needed. These controls allow timely identification and resolution of issues, ensuring system reliability and availability during high-use periods.