Software Design Specification

For

TimeTable Generator

Version <1.0.0>

Prepared by

Group No.: 16

Alasyam Likhith	SE22UCSE021
C Shanmukha Padma Kumar	SE22UCSE061
Chakilam Kunal	SE22UCSE063
Dachepally Sathwik	SE22UCSE073
Gopishetty Sathwik	SE22UCSE103
Kurnala Karthik	SE22UCSE146

Instructor	Dr. Vijay Rao Duddu
Course	Software Engineering
Lab Instructors	Mrs. Swapna
Date	07.04.2025

1. Introduction

1.1 Purpose of this Document

This Software Design Specification (SDS) outlines the architecture and design decisions made for the "Web Utility for Time Table Generator". It translates the Software Requirements Specification (SRS) into a blueprint for construction. It is intended for developers, testers, and future maintainers of the software.

1.2 Scope of the System

The Time Table Generator is a web-based tool that automates the process of creating and managing timetables for educational institutions. It allows administrators to define subjects, assign teachers, allocate classrooms, and generate conflict-free, optimized schedules for different grade sections.

1.3 References

1. **Software Requirements Specification for Time Table Generator**
    - Title: *Web Utility for Time Table Generator – SRS*
    - Author: Shanmukha Padma Kumar Challa
    - Reference: Internal project documentation (PDF provided)
    - Source: Uploaded document `16_Web Utility for Time Table Generator_SRS.pdf`
2. **Final Project Report – Time Table Generator**
    - Title: *SE22UCSE061 – Time Table Generator Full Report*
    - Description: Includes UML diagrams, activity diagrams, component design, testing, and implementation
    - Reference: Internal project documentation
    - Source: Uploaded PDFs `SE22UCSE061_SHANMUKHA PADMA KUMAR.pdf`, `SE22UCSE061_SHANMUKHA PADMA KUMAR CHALLA.pdf`
3. **IEEE SRS Template Reference**

    - Title: *IEEE Std 830-1998 – Recommended Practice for Software Requirements Specifications*
    - URL: https://ieeexplore.ieee.org/document/720574

4. **UML Diagrams and Design Models**
    - Description: Sequence, Class, Activity, and Component diagrams used for system modeling
    - Source: As submitted in final project report and PDFs above
5. **MySQL and PHP Documentation**
    - MySQL: https://dev.mysql.com/doc/
    - PHP: https://www.php.net/docs.php
6. **HTML/CSS and JavaScript References**
    - HTML & CSS: https://developer.mozilla.org/en-US/docs/Web
    - JavaScript: https://javascript.info/

1.4 Definitions, Acronyms, and Abbreviations

- **SRS**: Software Requirements Specification
- **SDS**: Software Design Specification
- **UI**: User Interface
- **DB**: Database
- **CRUD**: Create, Read, Update, Delete
- **Admin**: Authorized user managing timetables
- **User**: General viewer of generated schedules

---

2 Use Case View

The Use Case View of the Time Table Generator system outlines the key interactions between users (actors) and the system's functionalities. It serves as a blueprint that defines how users accomplish tasks using the system. This view is crucial in understanding system behavior from an external perspective, emphasizing user goals and the functional services the system must provide.

The use case diagram below captures the major use cases that define the scope and structure of user interaction. These include essential operations such as logging in, adding faculty and subjects, generating timetables, and viewing or editing them.

Only a selected set of use cases is elaborated in detail, focusing on those that are central to system design, require complex interactions, or span multiple modules.

2.1 Use Case Diagram

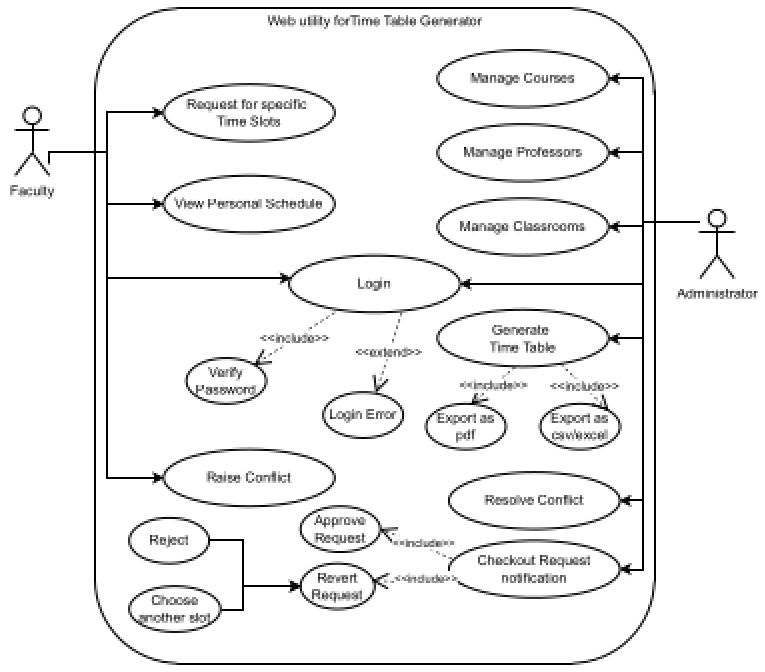

Fig: Use Case diagram for Time Table Generator

3 Design Overview

This section provides a comprehensive overview of the Time Table Generator system's software design. The application is designed using the MERN stack, which provides a modern, full-stack JavaScript-based framework to build scalable and maintainable web applications.

The system architecture follows a modular, layered design:

- Frontend (React.js) handles user interface and interaction.
- Backend (Express.js + Node.js) manages API routes, business logic, and connects with the database.
- Database (MongoDB) stores all application data including faculty, subjects, timings, and timetables.

This architecture supports loose coupling, reusability, high responsiveness, and efficient data management. The application is divided into independent components and RESTful APIs to promote clean communication across layers.

3.1 Design Goals and Constraints

Design Goals

- To generate a conflict-free academic timetable automatically and allow manual edits.
- To provide a smooth and responsive web-based UI for admins and faculty.
- To enable role-based access control (admin, faculty).
- To design a modular and scalable architecture using the MERN stack.
- To allow future integration of analytics or optimization features.

Constraints

- Tech stack limited to MERN: React.js (UI), Node.js + Express.js (Backend), MongoDB (Database).
- Admin is the primary actor responsible for creating and editing timetables.
- Faculty can only view their own timetable.

- The system must handle multiple departments, batches, and labs efficiently.
- Timetable generation algorithm is rule-based, not AI-powered.

3.2 Design Assumptions

- Faculty availability and subject allocations are pre-validated by the admin.
- Faculty will access their timetables only through the web app (no mobile app for now).
- The admin will manually input all required data before generation (e.g., faculty names, subjects, labs).
- MongoDB will be hosted locally or on a cloud service like MongoDB Atlas.
- Node.js server will handle all API requests and business logic centrally.

3.3 Significant Design Packages

Package Name	Responsibility
Frontend/UI (React)	Provides interactive user interface for login, dashboard, and timetable management.
Auth Module (JWT/React)	Handles secure login and token-based authentication for admin/faculty.
Timetable Generator (Node)	Contains logic for generating non-conflicting schedules based on user data.
Database Access Layer (Mongoose)	Manages schema definitions and data interaction with MongoDB.

Admin Operations Module	Provides admin functionalities: add/edit/delete faculty, subjects, slots, etc.
Faculty View Module	Allows faculty to view their assigned schedule in a clean interface.

Each of these is organized in either the frontend `src` folder or backend `routes`, `controllers`, and `models`.

3.4 Dependent External Interfaces

External Application	Module Using the Interface	Interface Name	Description and Interface Use
MongoDB	Database Access Layer (Mongoose)	`mongoose.connect`, `Schema`	Used to define collections and connect with the MongoDB database.
Web Browser	UI Module	HTTP/React REST calls	Renders the UI, handles routing and makes API calls to backend.
Node.js Server	Backend/Express Module	`express.Router`, `axios`, `cors`	Handles HTTP requests from frontend and applies business logic before responding.
JSON Web Token (JWT)	Authentication Module	`jsonwebtoken`	Used for secure token-based authentication and session management.

3.5 Implemented Application External Interfaces (and SOA Web Services)

Interface Name	Module Implementing the Interface	Functionality / Description
`/api/login`	Auth Controller	Verifies credentials and returns JWT token.
`/api/faculty/view-timetable`	Faculty Controller	Fetches the logged-in faculty's timetable from database.
`/api/admin/add-faculty`	Admin Controller	Allows admin to add a new faculty into the system.
`/api/admin/add-subject`	Admin Controller	Stores subject details to be associated with faculty/class.
`/api/generate-timetable`	Timetable Generator Controller	Auto-generates a clash-free schedule and stores in database.
`/api/edit-slot`	Timetable Edit Module	Allows admin to modify a specific slot manually.

---

4 Logical View

This section provides the detailed internal software design of the Time Table Generator. The application has been decomposed into several interconnected modules organized across three major layers:

- **Presentation Layer (React.js)** – Handles all user interaction and sends/receives data via API calls.

- **Application Logic Layer (Node.js + Express.js)** – Implements all backend business logic and coordinates data flow between UI and database.
- **Data Layer (MongoDB via Mongoose)** – Responsible for data storage, retrieval, and persistence.

Each of these layers is further divided into classes (or components in React) and modules that collaborate to complete use cases identified in Section 2.

4.1 Design Model

The software is modularized into distinct components. The following design model is explained through packages/modules and the significant classes or files within each.

A. Frontend (React.js)

- **LoginComponent**: UI for logging in and verifying user credentials.
- **DashboardComponent**: Main interface for admin or faculty after login.
- **TimetableViewerComponent**: Displays generated timetable for viewing.
- **FormComponents**: To enter subjects, faculty, slots.
- **TimetableGridComponent**: Shows editable timetable for admin.

  Responsibilities:

  - Display UI
  - Validate inputs
  - Make API calls to backend using `axios`

B. Backend (Node.js + Express.js)

- **authController.js**
  Handles login, JWT token generation and validation.
  - `loginUser()`
  - `verifyToken()`
- **facultyController.js**
  Manages faculty data fetching and individual timetable views.
  - `getFacultyTimetable()`
- **adminController.js**
  Admin-only functions like add/edit faculty, subjects, and slots.
  - `addFaculty()`
  - `addSubject()`
  - `editSlot()`
- **timetableGenerator.js**
  Contains logic for generating a non-conflicting timetable based on rules and input data.
  - `generateTimetable()`
  - `checkConflicts()`
  - `assignSlots()`

Relationships:

- All controllers use Mongoose models to access MongoDB data.
- JWT middleware ensures secure access.

C. Database Models (Mongoose)

- **FacultyModel.js**
    - Attributes: `name`, `id`, `subjectsHandled`, `availability`
- **SubjectModel.js**
    - Attributes: `name`, `code`, `credits`, `assignedFaculty`
- **SlotModel.js**
    - Attributes: `day`, `hour`, `classroom`, `facultyId`, `subjectCode`
- **TimetableModel.js**
    - Attributes: `department`, `semester`, `scheduleMatrix`

4.2 Use Case Realization

Below are examples of how the core use cases are realized within the software.

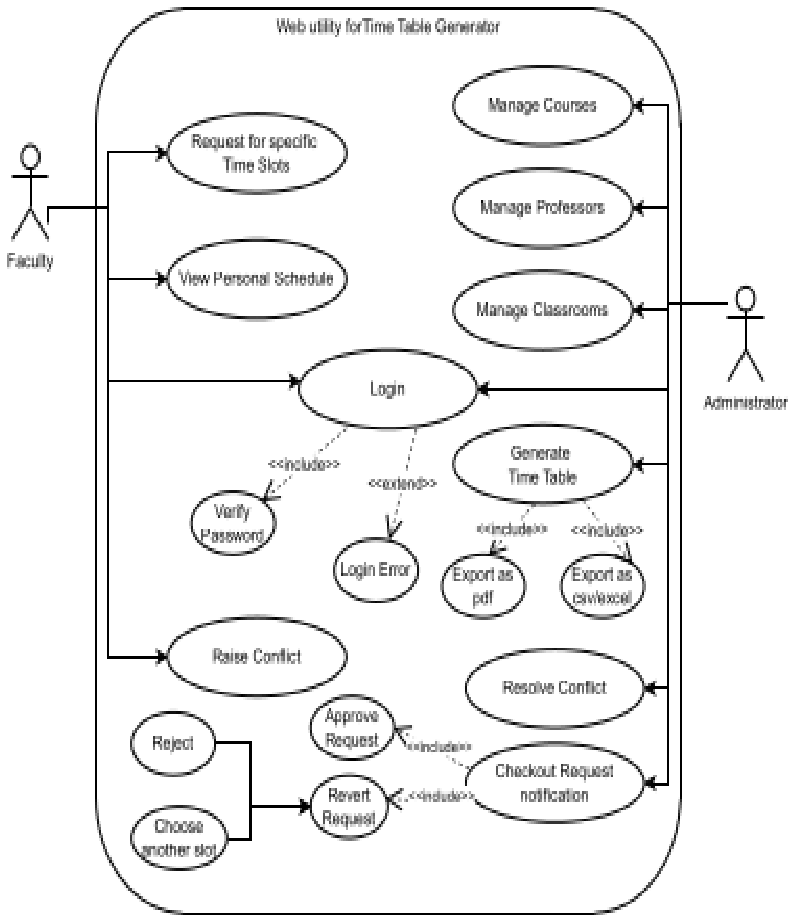

Fig: Use Case diagram for Time Table Generator

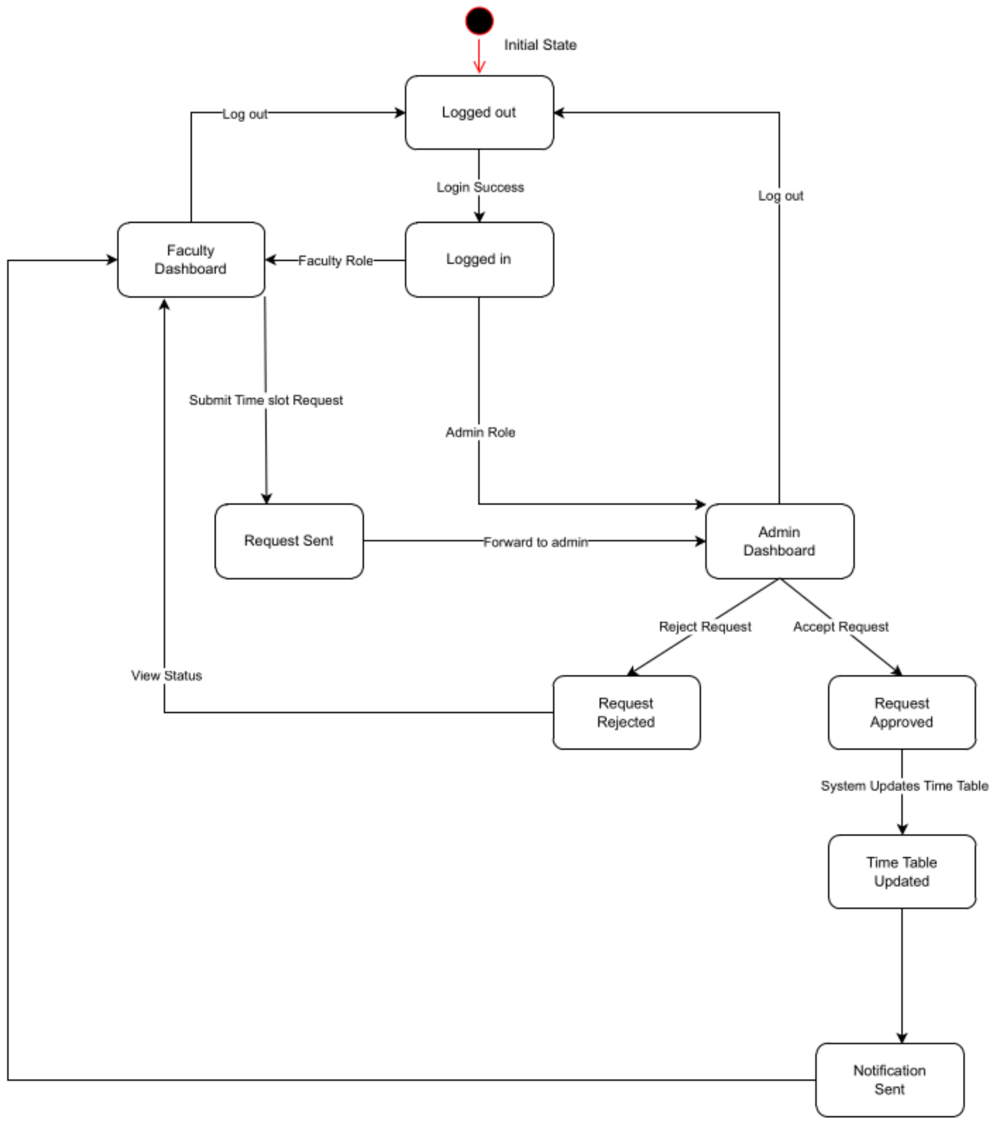

Fig: State Diagram of Time Table Generator

---

5 Data View

This section provides a detailed view of how the application handles persistent data. The project uses MongoDB as its NoSQL database for storing and managing all data objects, such as faculty details, subjects, slots, and generated timetables. Each data entity is mapped using Mongoose models in the backend.

5.1 Domain Model

The domain model for the Time Table Generator system captures the key entities and their relationships required to manage and generate academic timetables. This model abstracts the real-world components into software entities and focuses on how administrators interact with the system to assign faculties, subjects, and time slots.

The core domain entities are:

- Admin: Manages the entire system and performs key operations like adding faculties, subjects, defining slots, and generating timetables.
- Faculty: Represents professors or teachers who are assigned to teach specific subjects. Faculties have basic identification and department information.
- Subject: Represents academic subjects. Each subject is linked to a single faculty member who will handle that subject.
- Slot: Represents a specific time interval on a specific day. Each slot maps one subject and one faculty to a time window.
- Timetable: Represents the organized matrix of slots per department and semester.

5.2 Data Model (Persistent Data View)

Each of the core entities is represented as a separate collection in MongoDB. Below is a summary of each Mongoose schema:

Entity Descriptions:

Entity	Attributes
Admin	admin_id (PK), name, email, password

Faculty	faculty_id (PK), name, department, email
Subject	subject_id (PK), name, code, semester, faculty_id (FK)
Slot	slot_id (PK), day, time, faculty_id (FK), subject_id (FK)
Timetable	timetable_id (PK), department, semester, matrix_of_slots (2D structure of slots)

Relationships:

- An Admin can manage all operations but is not part of the timetable.
- A Faculty can handle multiple Subjects.
- A Subject is assigned to exactly one Faculty.
- A Slot involves one Faculty and one Subject at a specific time.
- A Timetable organizes multiple Slots by department and semester.

This model is designed to store persistent data using MongoDB collections where documents are referenced via unique IDs (`ObjectId`). The relationships are preserved using foreign key references.

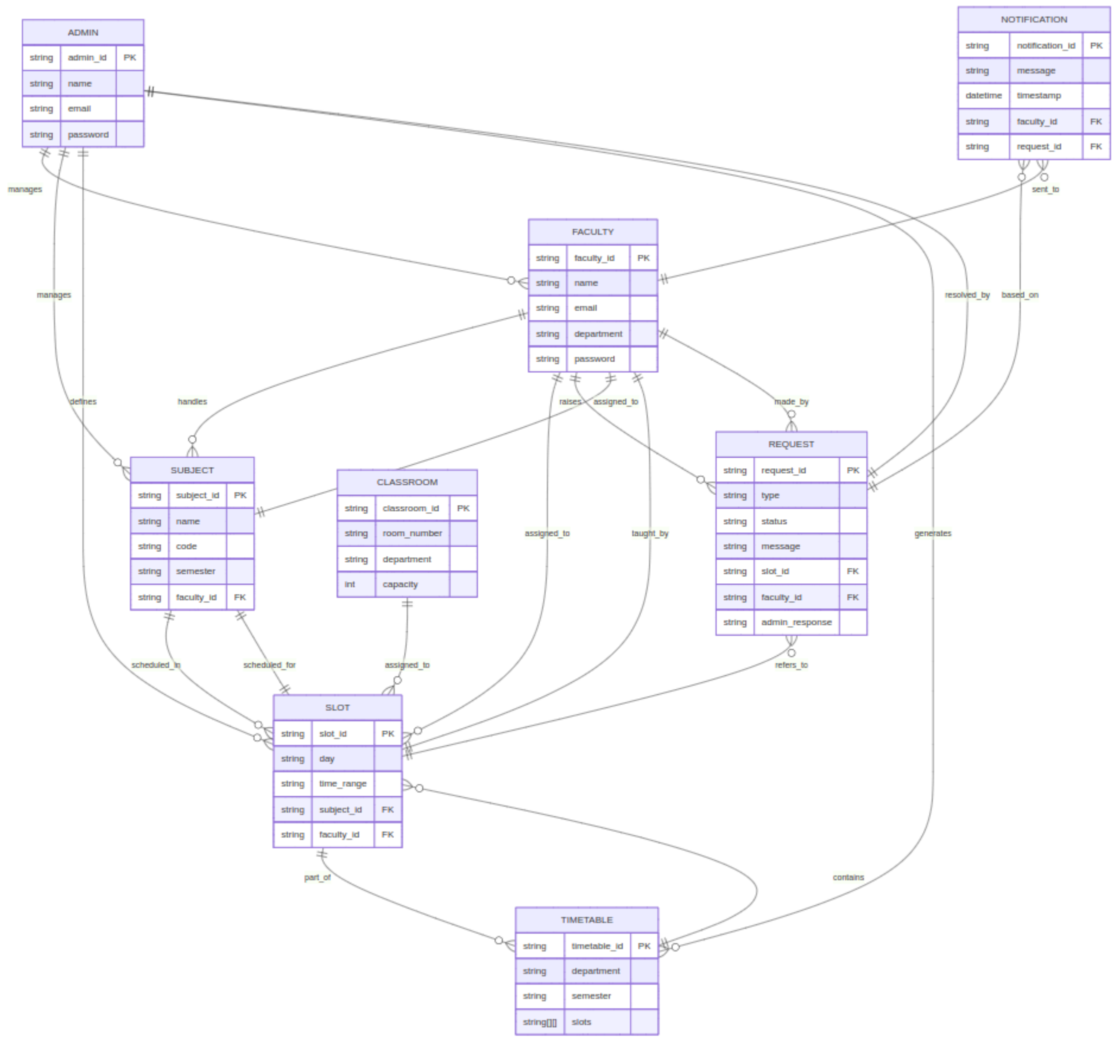

Fig: ER Diagram for our project

5.2.1 Data Dictionary

Entity	Field Name	Type	Description
Faculty	`facultyId`	String	Unique ID for faculty member
	`name`	String	Full name of the faculty
	`department`	String	Department name

	`availability`	Array[String]	Available slots (e.g., "Mon-1")
	`subjectsHandled`	Array[String]	Subject codes handled
Subject	`subjectCode`	String	Unique subject code
	`name`	String	Subject name
	`credits`	Number	Credit count of the subject
	`assignedFaculty`	String	Faculty ID assigned
Slot	`day`	String	Day of the week
	`hour`	Number	Hour number (1–7)
	`subjectCode`	String	Code of the subject assigned
	`facultyId`	String	Faculty teaching the subject
	`classroom`	String	Classroom location
Timetable	`department`	String	Department to which timetable belongs
	`semester`	Number	Semester number
	`schedule`	2D Array[Slot]	Timetable matrix of slots

6 Exception Handling

This section describes the various exceptions that may occur within the Time Table Generator system, how these exceptions are handled, logged, and the follow-up actions taken to ensure system stability and a smooth user experience.

6.1 Exception Categories

Category	Description
Input Validation Errors	**Occur when users provide invalid or missing data in forms.**
Database Errors	Issues related to database connectivity, schema mismatches, or CRUD failures.
Authentication Errors	Triggered during login or session validation if credentials are incorrect.
Scheduling Conflicts	Arise during timetable generation when faculty or slots are double-booked.
Server Errors	Node.js server-related issues or internal logic exceptions.
API Request Failures	Errors during frontend-backend communication due to malformed or failed requests.

6.2 Exception Handling Strategy

Frontend (React) Handling

- **Client-side validation** is used to prevent obvious incorrect input.
- **Error boundaries** in React components catch rendering errors.
- **Toast/Alert notifications** are used to inform users about issues.
- API responses are checked for success/failure and corresponding UI feedback is shown.

Backend (Node.js + Express) Handling

- **Middleware-based error handling** is implemented for centralized exception processing.
- Errors are caught using `try-catch` blocks in asynchronous controller logic.
- HTTP response status codes (e.g., 400, 401, 500) are sent appropriately with error messages.

6.3 Logging of Exceptions

- All backend exceptions are logged using a logging mechanism (e.g., console, future use of Winston/Log4js).
- Logs are used during debugging to trace failures.

Example log:

```
[ERROR] FacultyController.js: Unable to fetch faculty data - MongoDB
connection error
```

6.4 Follow-up Actions

Exception Type	Follow-up Action
Invalid Input	Prompt user with form error hints and prevent form submission.
Database Failure	Retry the operation or show “Database Unavailable” message.
Login Failure	Show “Invalid Credentials” message and request re-entry.
Timetable Conflict	Prevent saving and highlight conflicting slots for resolution.
API Failure	Notify user with a retry option or fallback UI.
Server Error	Log error, show generic failure message to user, and notify admin if critical.

6.5 Sample Exception Response

```
{
  "success": false,
  "message": "Faculty with ID SE22CSE101 not found",
  "errorCode": 404
}
```

7 Configurable Parameters

This section describes the parameters in the Time Table Generator system that are configurable. These parameters allow the application to be easily adapted for different deployment environments or user requirements without changing the core source code.

The following table outlines the primary configuration parameters:

Configuration Parameter Name	Definition and Usage	Dynamic ?
`PORT`	The port number on which the Express.js server listens.	No
`MONGODB_URI`	The connection string for the MongoDB database. It allows switching between development and production DBs.	No
`JWT_SECRET`	Secret key used for signing and verifying JSON Web Tokens (JWT) for authentication.	No
`REACT_APP_API_BASE_URL`	The base URL used by the React frontend to make API calls to the backend server.	No
`MAX_TIMETABLE_GENERATION_ATTEMPTS`	Maximum number of iterations the system will try to resolve a timetable without conflict.	Yes
`SESSION_TIMEOUT`	Duration after which a user session expires (in minutes).	Yes
`DEFAULT_USER_ROLE`	Role assigned to a newly registered user (e.g., “Faculty” or “Admin”).	Yes
`EMAIL_NOTIFICATIONS_ENABLED`	Enables/disables sending email notifications (e.g., timetable updates).	Yes

---

Notes:

- Environment variables are primarily stored in `.env` files for both frontend and backend and are read during application startup.

- **Dynamic parameters** (like `MAX_TIMETABLE_GENERATION_ATTEMPTS`) can be made editable from the Admin dashboard in future iterations without restarting the server.

8 Quality of Service

This section outlines the Time Table Generator application's design considerations related to availability, security, performance, and system monitoring. These aspects ensure that the system meets business requirements, remains robust, and performs optimally under varying loads.

8.1 Availability

The application is designed with a focus on high availability and minimal downtime. Key design aspects supporting this include:

- **Modular MERN Architecture**: By separating frontend, backend, and database layers, components can be updated or restarted independently, reducing the risk of total system outages.
- **Stateless Backend**: Express.js server is stateless, which simplifies scaling and deployment of multiple server instances behind a load balancer.
- **No Scheduled Downtime**: There are no tasks in the system (e.g., data loading, cleanup) that require full application downtime. Maintenance tasks such as data backup can be executed during non-peak hours.
- **Deployment Strategy**: Docker containers and version-controlled deployment allow zero-downtime rollouts.

8.2 Security and Authorization

Security is a core consideration in the design:

- **JWT Authentication**: The backend uses JSON Web Tokens for secure user authentication. Each request must carry a valid token to access protected routes.

- **Role-Based Access Control (RBAC)**: The system supports multiple roles such as Admin, Faculty, and Student. Role-based authorization ensures users only access data and features permitted to their role.
- **Password Hashing**: All passwords are hashed using industry-standard bcrypt before being stored in MongoDB.
- **Frontend Route Protection**: React routes are protected by role-aware middleware logic to prevent unauthorized access at the UI level.
- **Access Management**: Admin users can manage and assign roles, providing flexible and secure access control without external tools.

8.3 Load and Performance Implications

The Time Table Generator is designed to support multiple users and handle complex timetable generation operations efficiently:

- **Timetable Generation Performance**:
    - The generation algorithm runs on the backend and is optimized to avoid unnecessary recomputation.
    - It supports handling large input data (e.g., 100+ courses, faculty, and classrooms) in seconds.
- **Load Testing Strategy**:
    - Future test plans include simulating concurrent logins and timetable generations.
- **Database Scaling**:
    - MongoDB is chosen for flexible schema and fast read/write performance.
    - Collections are indexed appropriately for search efficiency (e.g., faculty ID, course codes).
- **Expected Usage**:
    - Designed to support institutional usage with up to 100 concurrent users in phase 1.

8.4 Monitoring and Control

To maintain system health and detect failures early, the following monitoring and control measures are integrated or planned:

- **Logging**:
  - Backend uses logging middleware (e.g., `morgan`) to log each request with status codes and response times.
  - Error handling is centralized to capture and log backend errors with stack traces.
- **Process Monitoring**:
  - PM2 or similar Node.js process managers can be used to monitor uptime and restart on crash.
- **Health Checks**:
  - Custom health-check APIs will be included to allow infrastructure to monitor server status.
- **Frontend Logging (Planned)**:
  - User interaction events and critical frontend errors can be logged for analysis.
- **Admin Dashboard (Future Feature)**:
  - Real-time logs, user activity statistics, and timetable generation reports may be added for better observability.