**Low Level Design (LLD)**

**REVISION NO:01**

**LAST DATE OF REVISION:18/11/2024**

**LOW LEVEL DESIGN (HLD)**

**TEACHER’S CONNECT**

**Teachers Connect**

**1**

**Low Level Design (HLD)**

**1**

**Teachers Connect**

**DOCUMENT VERSION CONTROL**

|  |  |  |  |
| --- | --- | --- | --- |
| **DATE ISSUED** | **VERSION** | **DESCRIPTION** | **AUTHOR** |
| 18/11/2024 | 1 | Initial HLD-v1.0 | Theja Ashwin H,  Sowdeshwari B Maheshwar R, |
|  |  |  |  |
|  |  |  |  |

**Teachers Connect**

**2**

**Teachers Connect**

**1**

# Contents

**Low Level Design (LLD)**

1. **Introduction** ................................................................................. 4
2. **Architecture** ................................................................................. 5
3. **Architecture Description** ......................................................... 5
4. **Data Description** ....................................................................... 5
5. **Data from Users** ......................................................................... 7
6. **Data Validation** ......................................................................... 8
7. **API Calls** ...................................................................................... 8
8. **Business Logic** ............................................................................ 9
9. **Security Consideration** ......................................................... 10
10. **Error Handling and Logging** .................................................. 11
11. **Performance Consideration** .................................................. 12
12. **Maintenance and Support** .................................................... 12
13. **Glossary and Terminology** ................................................... 13
14. **Testing** ....................................................................................... 13
15. **Deployment** ............................................................................... 14
16. **Unit Testing** .............................................................................. 15

**3**

**Teachers Connect**

**Low Level Design (LLD)**

### 1. Introduction

#### 1.1. What is a Low-Level Design Document (LLDD)?

A Low-Level Design Document (LLDD) provides detailed technical specifications for implementing program code. It defines the system architecture, class diagrams, methods, and module relationships, guiding developers in transforming high-level design into executable code. Core classes and methods.

* Inter-module relationships.
* Program inputs and outputs.

This document ensures developers can implement the system accurately with minimal ambiguity.

#### 1.2. Scope of Low-Level Design (LLD)

LLD focuses on breaking down the system into manageable components and their interactions. It refines the high-level design into concrete details, ensuring the software operates efficiently. The scope of LLD includes:

* **Data Structures**: Defines data types and flow.
* **Software Architecture**: Structures system components and interactions.
* **Source Code Design**: Details functions, classes, and algorithms.
* **Performance Algorithms**: Optimizes for fast performance, especially in recommendation systems.
* **Design Refinement**: Finalizes data flows and component interactions.

LLD acts as a bridge between conceptual designs and implementation, providing the technical details needed for coding, testing, and deployment.

**4**

**Teachers Connect**

**Low Level Design (LLD)**

**1. Architecture**

The system is designed using a **modern web architecture** that leverages the **MERN stack** (MongoDB, Express.js, React.js, Node.js) for optimal performance and flexibility:

* **Frontend**: The user interface is powered by **React.js**, providing a dynamic, responsive, and interactive experience for end-users.
* **Backend**: **Node.js** combined with **Express.js** serves as the backbone of the system, efficiently managing API requests, server logic, and ensuring seamless communication between the frontend and backend.
* **Database**: **MongoDB** is used as the primary database, offering fast, scalable, and flexible data storage and retrieval, essential for handling large datasets.
* **Hosting**: The system is hosted on **Render**, ensuring high availability, scalability, and easy access to the application for users.

**2. Architecture Description**

The system follows a modular approach, divided into three key layers that interact seamlessly:

* **Presentation Layer**: Built using **React.js**, this layer is responsible for handling all user interactions, providing an intuitive and user-friendly interface for activities such as login, registration, and viewing dashboards.
* **Business Logic Layer**: Powered by **Node.js** and **Express.js**, this layer manages core server-side operations such as **user authentication**, **appointment scheduling**, and **notifications**, ensuring smooth processing of business rules and workflows.
* **Data Layer**: The **MongoDB** database handles the persistent storage of critical data, including user information, appointment details, and communication logs, ensuring reliable and scalable data management.

**Data Description**

The system’s database is structured into the following collections, each serving a critical function for storing and managing the system's data:

* **Users**: This collection stores essential user information, such as:
  + ID: A unique identifier for each user.
  + Name: The full name of the user.
  + Email: The user’s email address for communication and authentication.
  + Role: Defines the user's role (e.g., student, teacher, admin).
  + Password: Encrypted password for secure authentication.

**Teachers Connect**

**5**

**Low Level Design (LLD)**

* **Appointments**: This collection tracks details related to the appointment bookings, including:
  + Teacher: ID of the teacher involved in the appointment.
  + Student: ID of the student booking the appointment.
  + Date: The date when the appointment is scheduled.
  + Time: The time of the appointment.
  + Status: The current state of the appointment (e.g., confirmed, canceled, pending).
* **Messages**: This collection is used for storing communication between users, enabling real-time messaging and tracking:
  + SenderID: ID of the user sending the message.
  + ReceiverID: ID of the user receiving the message.
  + MessageContent: The content of the message.
  + Timestamps: The date and time the message was sent.

**4. Model Building**

The application utilizes the following models, each representing a core entity within the system and designed to ensure data consistency and integrity:

* **User Model**: This model defines the user-related data structure and includes the following fields:
  + Name: User's full name.
  + Email: User’s email for authentication.
  + Password: A hashed password for secure login.
  + Role: Defines the user's role (e.g., admin, teacher, student).
  + Timestamps: Auto-generated timestamps for creation and modification dates.
* **Appointment Model**: This model tracks appointment-related information, including:
  + TeacherID: The ID of the teacher for the appointment.
  + StudentID: The ID of the student who booked the appointment.
  + Date: The scheduled date of the appointment.
  + Time: The scheduled time of the appointment.
  + Status: Current status (e.g., confirmed, canceled).
* **Message Model**: This model stores messages exchanged between users and includes the following fields:

**6**

**Teachers Connect**

**Low Level Design (LLD)**

* + Set their availability for appointments based on their schedules.
  + Approve, reschedule, or cancel student appointment requests.
  + Communicate directly with students regarding appointment details or changes.
* **Seamless Search and Booking Functionalities for Students:**  
  Students benefit from an intuitive interface that allows them to:
  + Search for teachers based on criteria such as name, department, or subject expertise.
  + Book appointments in real-time, ensuring immediate confirmation or feedback.
  + Send messages or queries to teachers, providing clarity on appointment purposes or requirements.
* **Real-Time Updates and Email Notifications:**  
  The system keeps all users informed through automated email alerts and real-time updates, minimizing miscommunication and ensuring that appointments are effectively managed.

**Data from Users**

The application collects the following types of data from users in order to facilitate the system’s functionality:

* **Registration Data**:
  + Name: The full name of the user, which is required for identification.
  + Email: The user’s email address, which will be used for login and communication purposes.
  + Password: A secure, encrypted password for authentication.
  + Role: Defines the user’s role in the system (e.g., admin, teacher, or student).
* **Appointment Data**:
  + Selected Teacher: The teacher chosen by the student for the appointment.
  + Date: The date when the appointment is scheduled to take place.
  + Time: The specific time of the appointment.
  + Purpose: The reason or topic of discussion for the appointment.
* **Messages**:
  + Communication Data: Messages exchanged between users (teachers and students) to enable effective communication and tracking of interactions.

**7**

**Teachers Connect**

**Low Level Design (LLD)**

**Data Validation**

To ensure the integrity, consistency, and quality of the data entered into the system, the following validation mechanisms are implemented:

* **Input Validation**:
  + Ensures that all fields are properly filled with appropriate data types, such as non-empty strings for name, valid email addresses for email, and correctly formatted timestamps for date and time.
* **Unique Constraints**:
  + The system enforces uniqueness for fields such as email to prevent multiple users from registering with the same email address and to ensure proper user identification.
* **Format Validation**:
  + Verifies that fields like email and date follow the correct format, ensuring they are valid and suitable for their intended purposes. For example, email addresses must conform to the standard email format, and appointment dates must match expected date formats.

## API Calls

The application uses the following routes for handling API calls:

|  |  |
| --- | --- |
| Route | Description |
| /api/v1/auth | Handles authentication (login, registration). |
| /api/v1/users | Manages user data (view, update, delete profiles). |
| /api/v1/teacher | Handles teacher-specific functionalities. |
| /api/v1/appointments | Manages appointment creation, viewing, and updates. |
| /api/v1/admin | Admin functionalities (user approvals, system monitoring). |
| /api/v1/messages | Facilitates messaging between users. |

**8**

**Teachers Connect**

**Low Level Design (LLD)**

**Business Logic**

**Algorithms**

* **Recommendation Algorithm**: The core algorithm that processes user preferences, historical data, and food features to suggest meals. It likely involves filtering, sorting, and ranking meals based on a scoring system.
* **Matching Algorithm for Appointments**: Logic that matches available times for students and teachers, ensuring no double-booking and taking into account user availability.
* **Role-based Logic**: Logic for defining user roles (admin, teacher, student) and determining the allowed actions for each role (e.g., only admins can create new appointments, students can only book appointments).

**Methods/Functions**

* **User Registration**: The registerUser() function processes the user registration, validating input, and assigning the role.
  + **Inputs**: Name, email, password, and role.
  + **Outputs**: Success message or error details (e.g., email already in use).
  + **Expected Behavior**: The function ensures that email uniqueness and role validation are in place.
  + **Edge Cases**: Invalid email format, password strength, and duplicate emails.
* **Appointment Scheduling**: The scheduleAppointment() function verifies user availability, prevents overlapping appointments, and stores the data in the database.
  + **Inputs**: Teacher, student, date, time, and purpose.
  + **Outputs**: Confirmation message or error message (e.g., slot unavailable).
  + **Expected Behavior**: Checks if the time slot is available and creates the appointment record.
  + **Edge Cases**: User tries to book an already occupied slot, invalid data, or unavailable teacher.

**9**

**Teachers Connect**

**Low Level Design (LLD)**

**Security Considerations**

**Authentication & Authorization**

* **Token-Based Authentication**: The system uses JWT (JSON Web Tokens) for securing routes. When a user logs in, a token is issued and stored in the client’s local storage. The token is then used to authenticate future requests to the backend.
  + **Authorization**: Role-based access control is applied to ensure that only authorized users (admin, teacher, student) can access specific resources.
  + **Middleware**: Custom middleware is used to check if the user’s JWT is valid and matches the role required for the endpoint.

**Data Protection**

* **Password Encryption**: Passwords are hashed using bcrypt before being stored in the database. This ensures that even if the database is compromised, passwords are not retrievable.
* **Sensitive Data**: Data such as user details and appointment information are transmitted over HTTPS to ensure encryption in transit.
* **Database Encryption**: Sensitive information such as user data and appointment history are stored using encrypted fields in MongoDB (optional).

**Threat Mitigation**

* **SQL Injection**: Although MongoDB is used, the system implements input validation and parameterized queries to mitigate injection attacks.
* **Cross-Site Scripting (XSS)**: Proper sanitization and escaping of input data are done to prevent XSS attacks in the web application.
* **Cross-Site Request Forgery (CSRF)**: CSRF tokens are implemented for any form submissions to prevent unauthorized actions.

**10**

**Teachers Connect**

**Low Level Design (LLD)**

**Error Handling and Logging**

**Error Codes and Messages**

* **400 Bad Request**: If the user provides invalid input during registration or appointment scheduling.
* **401 Unauthorized**: When trying to access a protected route without proper authentication.
* **404 Not Found**: When an appointment or user cannot be found.
* **500 Internal Server Error**: For unexpected errors during API calls or database interactions.

**Logging Strategy**

* **Error Logging**: All errors are logged using a logging library (e.g., Winston or Bunyan) for later debugging and monitoring.
* **Activity Logging**: User actions (e.g., login, appointment creation) are logged to monitor usage patterns and ensure accountability.
* **Third-Party Logging**: Services like Sentry or Loggly can be used to monitor and report real-time errors.

**Performance Considerations**

**Optimization Techniques**

* **Caching**: Frequently accessed data (e.g., popular meals or teacher availability) is cached in-memory using Redis to speed up responses.
* **Database Indexing**: Indexing on key fields such as userId, appointmentDate, and email ensures fast querying and retrieval.
* **Lazy Loading**: Lazy loading of data (e.g., meal recommendations) to prevent large payloads and improve frontend performance.

**11**

**Teachers Connect**

**Low Level Design (LLD)**

**Scalability**

* **Horizontal Scaling**: The backend is designed to scale horizontally, meaning additional server instances can be added easily.
* **Auto-Scaling**: Deployed on platforms like Render or AWS, which automatically scale the application depending on traffic load.
* **Database Sharding**: MongoDB can be configured for sharding to handle large datasets by distributing the data across multiple servers.

**Latency and Throughput**

* **Response Time**: The system is optimized to respond to user queries (like meal recommendations or appointment checks) within 500ms under normal load.
* **Throughput**: The system can handle up to 1000 concurrent users and 10,000 appointment requests per day, with adequate load balancing in place.

**Maintenance and Support**

**Update Strategy**

* **Version Control**: The project uses Git for version control, allowing changes to be tracked and managed through GitHub or GitLab. All code updates will be committed regularly, and version tags will be applied to mark major releases. A clear branching strategy (e.g., main, develop, feature) will be followed.
* **Patching**: Regular patches will be applied to address bugs, security vulnerabilities, or performance issues. This will be handled by a dedicated team member or by automated dependency management tools such as **Dependabot** to keep the system up to date with security patches.
* **Deployment Frequency**: Updates will follow a bi-weekly or monthly release cycle, with emergency patches deployed as needed for critical issues.

**Monitoring**

* **System Health**: Tools such as **New Relic**, **Datadog**, or **Prometheus** will be used for continuous monitoring of system performance, server health, and response times. These tools will provide real-time data about the system’s performance, resource usage, and traffic.
* **Error Monitoring**: Services like **Sentry** or **Rollbar** will be employed to track errors in real time. Alerts will be sent to the development team for any high-priority errors, ensuring that issues are addressed quickly.
* **User Activity Tracking**: **Google Analytics** or **Mixpanel** can track user interactions within the system, allowing for improved user experience analysis and performance optimization.

**12**

**Teachers Connect**

**Low Level Design (LLD)**

**Glossary and Terminology**

**Definitions**

* **User Roles**: Different roles in the system (e.g., **Admin**, **Teacher**, **Student**) that have varying levels of access and permissions.
* **JWT (JSON Web Token)**: A secure method for transmitting user authentication data between client and server, used for managing user sessions.
* **Appointments**: Scheduled meetings between users (teachers and students) that are tracked by the system, including the time, date, and status.
* **Recommendation Engine**: The system’s algorithm responsible for suggesting meals based on user preferences, availability, and historical data.
* **API Endpoints**: The defined paths for interacting with the backend system, e.g., **POST /appointments** to create a new appointment.
* **MongoDB**: A NoSQL database used for storing user data, appointment records, and other system information in a flexible, schema-less format.
* **CI/CD (Continuous Integration/Continuous Deployment)**: The practice of automating code integration and deployment to streamline updates and ensure rapid delivery of features and bug fixes.

**Testing**

**Unit Tests**

* **User Registration Module**: Unit tests ensure that user registration works as expected, validating email formats, password strengths, and role assignments.
* **Appointment Scheduling**: Unit tests check if appointments are scheduled correctly, ensuring that overlapping appointments are prevented and the correct teacher and student data are linked.
* **Recommendation Logic**: Unit tests validate the accuracy of the recommendation engine, ensuring that meal suggestions align with user preferences.

**Integration Tests**

* **User Registration & Login**: Integration tests verify that the entire user registration and login process works as expected, including email validation, password hashing, and token generation.
* **Appointment Creation and Notifications**: Integration tests ensure that once an appointment is created, it triggers the appropriate notifications and updates the database.

**Teachers Connect**

**13**

**Low Level Design (LLD)**

* **Recommendation System & Data**: Ensures the recommendation engine properly interacts with the database and serves personalized meal suggestions based on stored data.

**Performance Testing**

* **Load Testing**: Tools like **Apache JMeter** or **Gatling** will simulate 1000+ concurrent users to evaluate how well the system handles high traffic, ensuring minimal response delays.
* **Stress Testing**: This will test the system’s stability under peak load conditions, identifying the maximum load the system can handle before failure.
* **Scalability Testing**: The system will be tested for its ability to scale with increasing traffic, checking whether it can handle added loads via horizontal scaling or load balancing.

**Deployment**

#### ****Environment Setup****

* **Development Environment**:
  + Local machines used for development with both frontend (React.js) and backend (Node.js with Express) services running locally.
  + MongoDB can be set up locally for development and testing purposes.
* **Production Environment**:
  + Hosted on **Render**:
    - Frontend (React.js) served as a static site.
    - Backend (Node.js/Express) deployed as a web service.
    - **MongoDB Atlas** used for database management with automatic scaling and backups.

#### ****Deployment Process****

* **CI/CD Pipeline using GitHub & Render**:
  1. **Code Commit**: Developers push code to the **GitHub repository**.
  2. **GitHub Actions**: Automated CI pipeline triggers on each commit to:
     + **Build**: React app is built for production, backend code is compiled.
     + **Test**: Unit tests and integration tests are run.
  3. **Deploy to Render**: Once tests pass, the app is deployed automatically:
     + **Frontend**: Deployed as a static site on **Render**.
     + **Backend**: Deployed to **Render’s web service** for API management.
* **Hosting**:
  1. **Render** handles both frontend and backend hosting with a streamlined deployment process.
  2. **MongoDB Atlas** manages the database in production with scaling, backups, and monitoring features.

**14**

**Teachers Connect**

**Low Level Design (LLD)**

#### ****Rollback Strategy****

* **Version Control**:
  + In case of issues, developers can roll back to the previous stable version by using **Git tags** or commit hashes on **GitHub**.
* **Automated Rollback**:
  + **Render** offers automatic rollback in case of deployment failure, reverting to the previous stable version.
* **Database Rollback**:
  + **MongoDB Atlas** provides automatic backups and snapshots, allowing restoration to a previous database state if needed.

Unit testing

**Teachers Connect**

**15**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| | **Test Case** | **Description** | **Pre-Requisite** | **Expected Result** | | --- | --- | --- | --- | | **User Registration** | Test the user registration functionality. | The user must provide a unique email, name, and password. | A new user account is created, and a success message is returned. | | **User Login** | Test user login with valid credentials. | The user must be registered with valid credentials. | User should be logged in successfully, and a session token should be returned. | | **User Login (Invalid Credentials)** | Test user login with invalid credentials. | The user must attempt login with incorrect credentials. | Login attempt should fail, and an error message should be returned. | | **Create Appointment** | Test the appointment creation process. | The user should be logged in, and a valid teacher must be available. | Appointment should be created with the specified teacher, student, date, and time. | | **View Appointments** | Test if the logged-in user can view their appointments. | The user must be logged in and have at least one scheduled appointment. | All appointments for the logged-in user should be displayed correctly. | | **Message Sending** | Test if a user can send a message to another user. | Both the sender and receiver should be registered users. | The message should be successfully sent and stored in the database. | | **Appointment Cancellation** | Test if a user can cancel an appointment. | The user should have a scheduled appointment. | The appointment should be successfully canceled, and the status updated in the database. | | **Database Query for User** | Test the database query to fetch user details by email. | The user must exist in the system. | The system should return the correct user details. | | **Appointment Status Update** | Test the system's ability to update appointment status. | The appointment should exist in the system. | The status of the appointment should be successfully updated (e.g., from 'Pending' to 'Confirmed'). | | **Password Hashing** | Test if the system correctly hashes the user password. | A user registration request with a password. | The password should be securely hashed and stored in the database, not as plain text. | |

**16**

**Teachers Connect**