LOW LEVEL DOCUMENT - EduManage

EduManage

BHAVDEEP SINGH NIJHAWAN

CONTENTS

1. INTRODUCTION

1.1 What is a Low-Level Design Document?

The goal of a Low-Level Design (LLD) document is to provide the internal logical design of the actual program code for the EduManage system. The LLD describes class diagrams with methods, relationships between classes, and program specifications. It details the modules so that programmers can directly code the program from the document.

1.2 Scope

Low-level design (LLD) is a component-level design process that follows a step-by-step refinement approach. This process is used for designing data structures, the required software architecture, source code, and performance algorithms. The organization of data is initially defined during requirement analysis and then refined during the data design phase, ensuring that the EduManage system is both robust and efficient in its operations.

1. ARCHITECTURE

2.1 Frontend:

- React.js: A JavaScript library for building user interfaces. It will handle the client-side of the application.
- Components: Various UI components for different pages and functionalities.
- Routing: React Router for handling navigation between pages.
- State Management: Redux or Context API for managing application state.
- Styling: CSS or CSS frameworks like Bootstrap for styling components.

2.2 Backend:

- Node.js: A JavaScript runtime environment for building server-side applications.
- Express.js: A web application framework for Node.js to build APIs and handle HTTP requests.
- Routes: Define API endpoints to handle various CRUD operations.
- Controllers: Logic for handling requests and interacting with the database.
- Middleware: Express middleware for authentication, error handling, etc.

2.3 Database:

- MongoDB: A NoSQL database for storing application data.
- Collections: Different collections for storing data related to users, courses, assignments, etc.
- Schema: Mongoose schema for defining the structure of data models.
- Queries: MongoDB queries for interacting with the database.

2.4 Authentication:

- JWT (JSON Web Tokens): For user authentication and authorization.
- Register: API endpoint for user registration.
- Login: API endpoint for user login to generate JWT tokens.
- Middleware: Authentication middleware to validate JWT tokens for protected routes.

2.5 Frontend-Backend Communication:

- REST API: Use HTTP methods (GET, POST, PUT, DELETE) for client-server communication.
- Axios: A promise-based HTTP client for making AJAX requests from the frontend to the backend.

2.6 Deployment:

- Hosting: Deploy the frontend on platforms like Netlify or Vercel and the backend on platforms like Heroku.
- Domain: Purchase and configure a domain name for the application.

2.7 Version Control:

- Git: Version control system for tracking changes in the codebase.
- GitHub: Host the project repository on GitHub for collaboration and version history.

2.8 Testing:

- Unit Testing: Write and execute unit tests using Jest for frontend components and Mocha/Chai for backend APIs.
- Integration Testing: Test the interaction between frontend and backend components.
- End-to-End Testing: Test the entire application flow using tools like Cypress.

2.9 CI/CD (Continuous Integration/Continuous Deployment):

- Automated Builds: Set up automated build pipelines using GitHub Actions or Travis CI.
- Automated Tests: Run automated tests on each commit to ensure code quality.
- Continuous Deployment: Automatically deploy changes to staging or production environments after passing tests.

3. ARCHITECTURE DESCRIPTION

3.1 Data Description:

- Utilize datasets containing educational content, such as course materials, lesson plans, and learning resources.
- Additional datasets may include student performance data, ratings, and user feedback.

3. 2 Web Scraping:

- Scrape educational websites and platforms to gather comprehensive course information, including syllabi, instructor details, and course ratings.

3.3 Data Transformation:

- Convert the collected data into a structured format, such as JSON or CSV, and perform necessary data cleaning and preprocessing steps.

3.4 Data Insertion into Database:

- Create a database schema to store educational content, user profiles, course details, and feedback.
- Insert the transformed data into the database tables, ensuring proper indexing and relationships between entities.

3.5 Export Data from Database:

- Export relevant data from the database for further analysis, processing, and model training.

3.6 Data Preprocessing:

- Preprocess the exported data, including handling missing values, text normalization, tokenization, and feature extraction.
- Techniques such as TF-IDF, word embeddings, and sentiment analysis may be applied to enhance data quality.

3.7 Data Clustering:

- Utilize clustering algorithms to group similar courses, users, or educational resources based on predefined features.
- Determine the optimal number of clusters using techniques like the elbow method or silhouette score.

3.8 Model Building:

- Train machine learning models for personalized recommendation, course prediction, or student performance analysis.
- Evaluate model performance using metrics such as accuracy, precision, recall, and F1-score.

3.9 User Data Collection:

- Collect user data, including demographic information, educational background, learning preferences, and goals.

3.10 Data Validation:

- Validate user-provided data to ensure accuracy, consistency, and relevance for personalized recommendations.

3.11 User Data Insertion into Database:

- Store user profiles and preferences in the database, allowing for efficient retrieval and personalized recommendations.

3.12 Model Call for Specific User Profile:

- Load the appropriate machine learning model based on the user's profile and preferences.
- Generate personalized recommendations for courses, learning paths, or educational resources.

3.13 Recommendation & Saving Output:

- Present recommended courses, learning materials, or resources to users based on their preferences and interests.
- Save user interactions and feedback to improve recommendation algorithms over time.

3.14 Deployment:

- Deploy the recommendation system and web application to a cloud platform such as AWS, Azure, or Google Cloud.
- Ensure scalability, reliability, and security of the deployed application.

4. UNIT TEST CASES

S.No.	Test Case Description	Prerequisite	Expected Result
1.	Verify Application URL Accessibility	Application URL is defined	Application URL is accessible to the user
2.	Verify Application Load Completion	Application URL is accessible	Application loads completely for the user
3.	Verify User Sign Up	Application is accessible	User is able to sign up in the application
4.	Verify User Login	Application is accessible	User is able to successfully login to the application
5.	Verify Input Field Visibility on Login	Application is accessible	User can see input fields on logging in
6.	Verify User Input Field Editing Ability	Application is accessible	User can edit all input fields
7.	Verify Submit Button Availability	Application is accessible	User gets Submit button to submit the inputs
8.	Verify Recommended Results Display on Submit	Application is accessible	User is presented with recommended results on clicking submit
9.	Verify Recommended Results Alignment with User Selections	Application is accessible	Recommended results align with the selections user made
10.	Verify Filter Options Availability for Recommended Results	Application is accessible	User has options to filter the recommended results