Low Level Design

QUIZ GAME APP QUIZZYBEE 

Written By	SNEHA KUMARI
Document Version	0.3
Last Revised Date	25-06-2024

Document Control

Change Record:

Version	Date	Author	Comments
0.1	16 – JUNE-2024	SNEHA	Introduction & Architecture defined
0.2	20 – JUNE-2024	SNEHA	Architecture & Architecture Description appended and updated
0.3	25 – JUNE-2024	SNEHA	Unit Test Cases defined and appended

Contents

1. Introduction

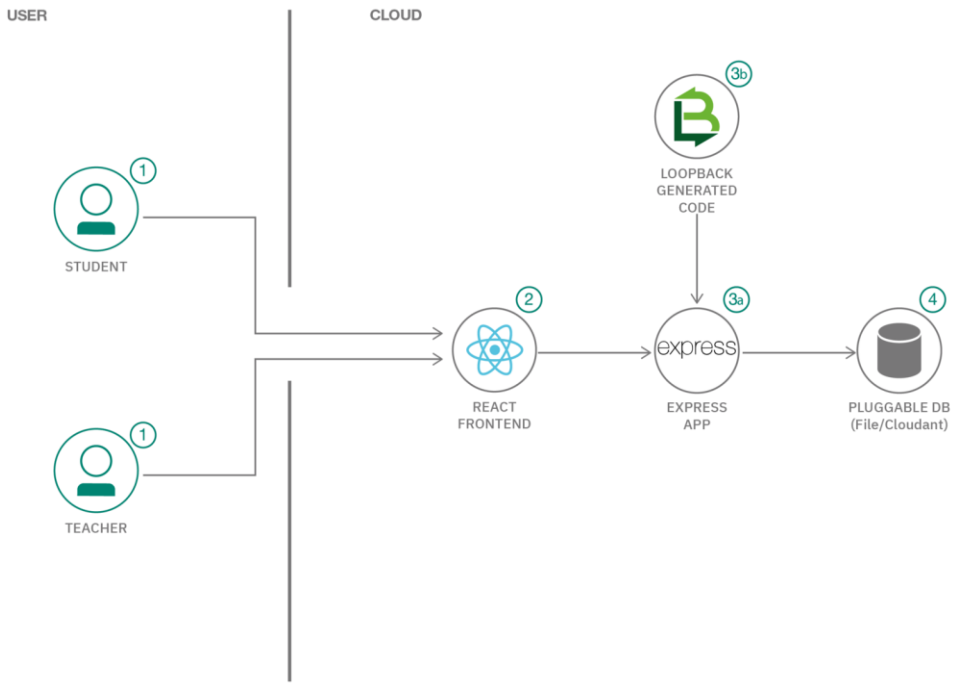
1.1 Purpose

The purpose of this Low-Level Design (LLD) document is to provide the internal logical design of the actual program code for the Quizzy Bee application. This document details the component structure, methods, and interactions within the application to guide programmers in writing the code.

1.2 Scope

The LLD focuses on the frontend component of the Quizzy Bee application, developed using JavaScript, React, and related technologies. It outlines the detailed design of user interface components, state management, routing, API integration, and styling, ensuring a cohesive and efficient implementation.

2. Architecture



2.1 Component Description

App Component

- Entry point of the application.
- Sets up routing and global context providers.

Header Component

- Displays the logo, navigation links, and user actions (login, profile).

Footer Component

- Contains links to company information, social media, and other relevant links.

HomePage Component

- Displays featured quizzes and categories.

QuizList Component

- Displays a list of quizzes based on category or search query.

QuizDetail Component

- Shows detailed information about a single quiz, including questions.

QuizTaking Component

- Allows users to take the quiz and submit answers.

Result Component

- Displays the results of a quiz attempt.

2.2 User Interface Components

Button Component

- Reusable button with props for text, click handlers, and styles.

QuizCard Component

- Displays quiz image, title, and a start button.

SearchBar Component

- Allows users to search for quizzes by name or category.

CategoryMenu Component

- Displays quiz categories for navigation.

2.3 State Management

Redux Store

- Centralized store to manage global state (user information, quiz data).

Reducers

- UserReducer: Manages user state (login, profile update).
- QuizReducer: Manages quiz list and details state.

Actions

- User Actions: loginUser, logoutUser, updateUserProfile.
- Quiz Actions: fetchQuizzes, fetchQuizDetails, submitQuizAnswers.

2.4 Routing

React Router Configuration

- BrowserRouter for wrapping the app.

- Route components for mapping URL paths to components.

Routes

- /: HomePage
- /quizzes: QuizList
- /quiz/:id: QuizDetail
- /take-quiz/:id: QuizTaking
- /result/:id: Result

2.5 API Integration

Axios Setup

- Create an Axios instance for API calls with base URL and default headers.

API Calls

- fetchQuizzes: GET request to retrieve quiz list.
- fetchQuizDetails: GET request to retrieve quiz details by ID.
- submitQuizAnswers: POST request to submit quiz answers and get results.

2.6 Styling

CSS Modules

- Scoped CSS to prevent class name collisions.

Styled-Components

- Utilized for dynamic and reusable styles.

Global Styles

- Define global styles and theme using CSS variables or a theme provider.

3. Detailed Design

3.1 User Management

Registration

- **Component**: RegistrationForm
- **API Call**: POST /register
- **Description**: Users can register by providing necessary details.
- **State Management**: Dispatch `registerUser` action to store user information.

Login/Logout

- **Component**: LoginForm, LogoutButton
- **API Call**: POST /login, POST /logout
- **Description**: Secure authentication for users to log in and out.
- **State Management**: Dispatch `loginUser` and `logoutUser` actions to update user state.

Profile Management

- **Component**: UserProfile
- **API Call**: PUT /user/
- **Description**: Users can update their profile information.
- **State Management**: Dispatch `updateUserProfile` action to update user information.

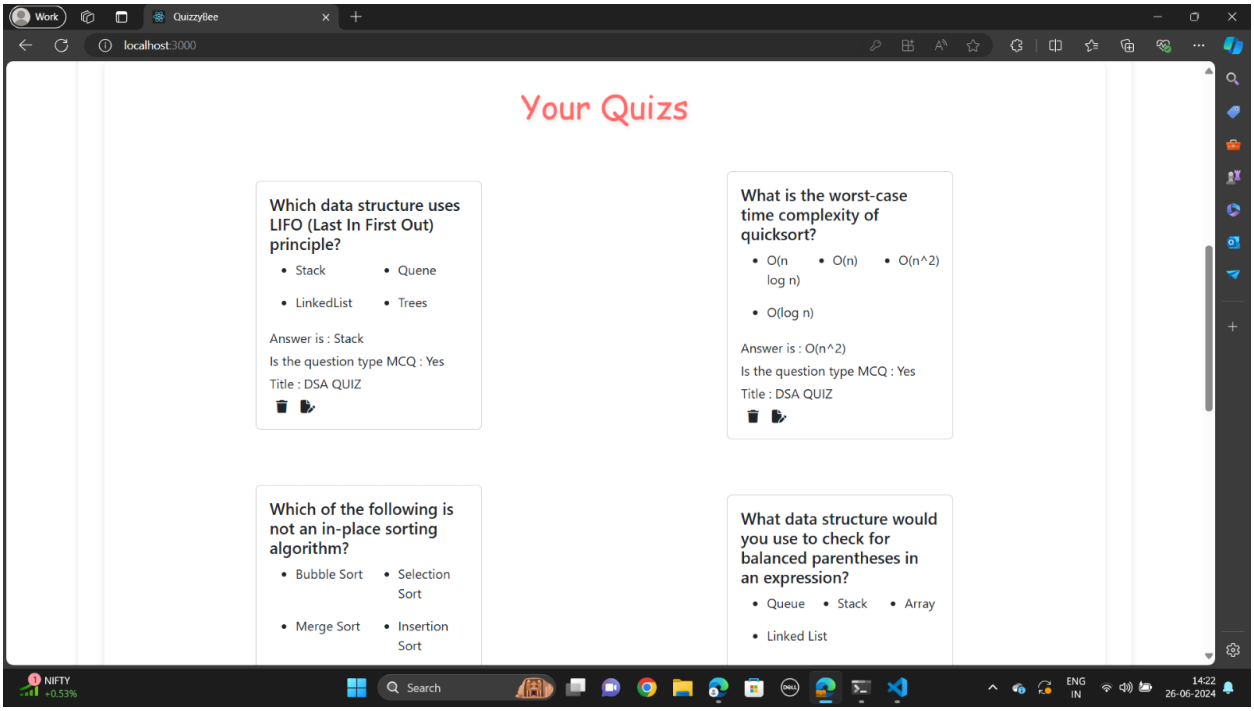
3.2 Quiz Management

Create Quiz

- **Component**: QuizForm
- **API Call**: POST /quizzes
- **Description**: Users can create quizzes by adding questions and answers.
- **State Management**: Dispatch `createQuiz` action to add the quiz to the store.

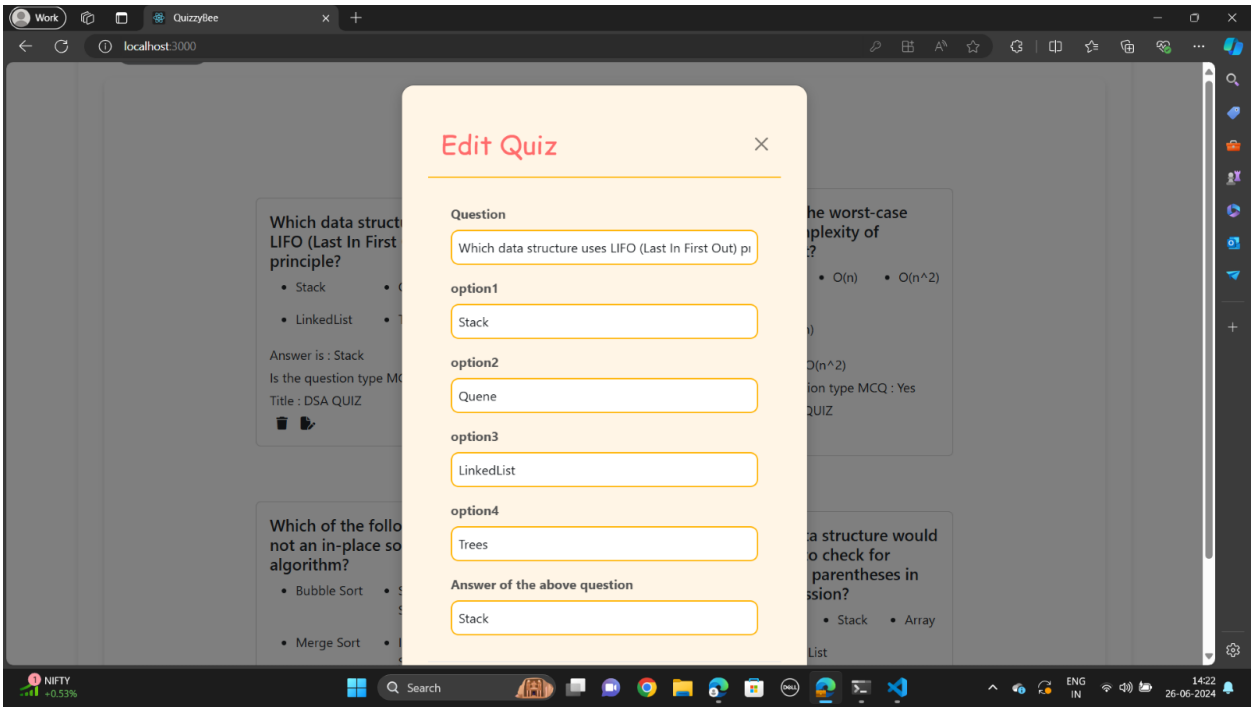
Edit Quiz

- **Component**: QuizForm
- **API Call**: PUT /quizzes/
- **Description**: Users can edit existing quizzes.
- **State Management**: Dispatch `editQuiz` action to update the quiz in the store.



Delete Quiz

- **Component**: QuizList
- **API Call**: DELETE /quizzes/
- **Description**: Users can delete their quizzes.
- **State Management**: Dispatch `deleteQuiz` action to remove the quiz from the store.



3.3 Quiz Taking

Start Quiz

- **Component**: QuizTaking
- **API Call**: GET /quiz/
- **Description**: Users can start a quiz.
- **State Management**: Dispatch `startQuiz` action to fetch quiz details.

Answer Questions

- **Component**: Question
- **Description**: Users can answer quiz questions.
- **State Management**: Update local state with user answers.

Submit Quiz

- **Component**: QuizTaking
- **API Call**: POST /quiz/

/submit

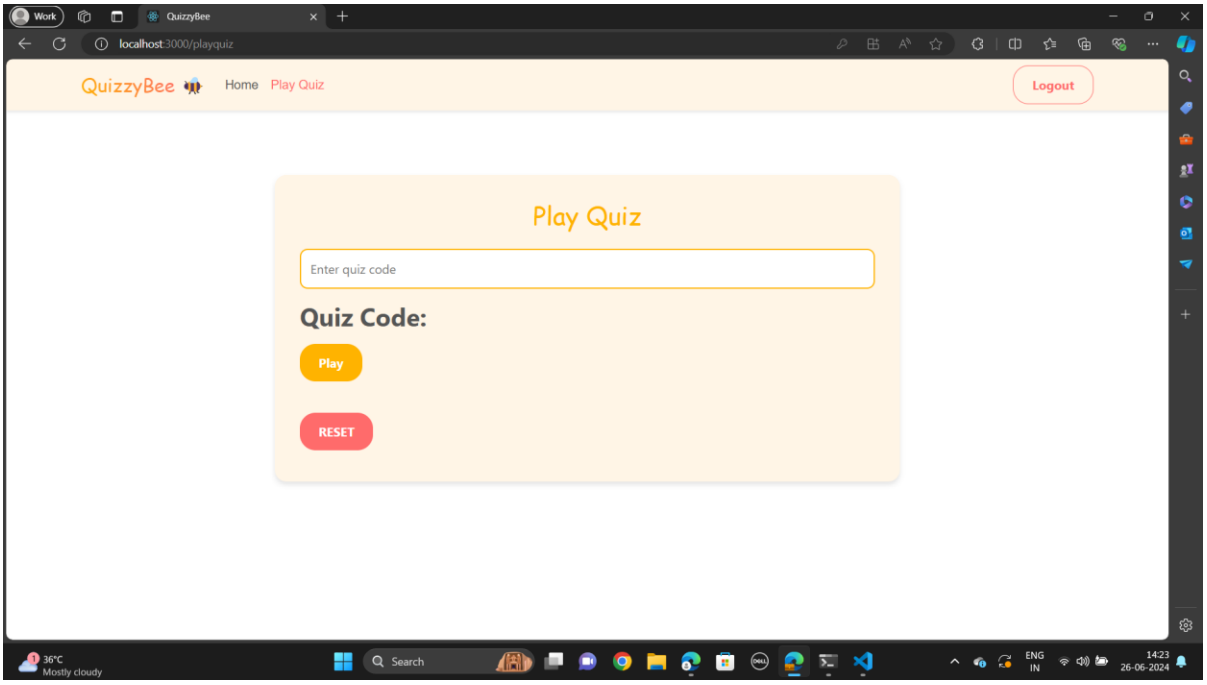
- **Description**: Users can submit the quiz to get results.
- **State Management**: Dispatch `submitQuiz` action to submit answers and fetch results.

3.4 Results and Analytics

View Results

- **Component**: Result
- **API Call**: GET /result/
- **Description**: Users can view their quiz results.
- **State Management**: Dispatch `fetchResult` action to fetch and store result data.

Analytics Dashboard

- **Component**: Dashboard
- **API Call**: GET /analytics
- **Description**: Users can see their performance analytics.
- **State Management**: Dispatch `fetchAnalytics` action to fetch and store analytics data.
- 

3.5 Notifications

Email Notifications

- **Component**: NotificationSettings
- **API Call**: POST /notifications/email
- **Description**: Users receive email notifications for important updates.
- **State Management**: Dispatch `updateEmailNotifications` action to update email notification settings.

In-App Notifications

- **Component**: NotificationCenter
- **API Call**: GET /notifications
- **Description**: Users receive notifications within the app.
- **State Management**: Dispatch `fetchNotifications` action to fetch and store notifications.

4. Database Design

4.1 Schema Diagram

4.2 Table Definitions

Users

Field	Type	Description
id	ObjectId	Primary key
username	String	Unique username
password	String	Hashed password
email	String	User email
profileInfo	Object	Additional profile info

Quizzes

Field	Type	Description
id	ObjectId	Primary key
title	String	Quiz title

description	String	Quiz description
questions	Array	List of quiz questions
creatorId	ObjectId	Reference to user id

Questions

Field	Type	Description
id	ObjectId	Primary key
quizId	ObjectId	Reference to quiz id
questionText	String	Question text
options	Array	List of answer options
correctAnswer	String	Correct answer

Results

Field	Type	Description
id	ObjectId	Primary key
quizId	ObjectId	Reference to quiz id
userId	ObjectId	Reference to user id
answers	Array	User's answers
score	Number	Quiz score

5. Unit Test Cases

5.1 App Component

Test Case 1: Verify that the App component renders without crashing.

- **Pre-Requisite**: Application setup is complete.
- **Expected Result**: App component renders successfully.

5.2 Header Component

Test Case 1: Verify that the Header displays navigation links.

- **Pre-Requisite**: Header component is rendered.
- **Expected Result**: Navigation links are displayed correctly.

Test Case 2: Verify that the Header shows the profile icon.

- **Pre-Requisite**: Header component is rendered.
- **Expected Result**: Profile icon is displayed correctly.

5.3 HomePage Component

Test Case 1: Verify that the HomePage displays featured quizzes.

- **Pre-Requisite**: HomePage component is rendered.
- **Expected Result**: Featured quizzes are displayed on the homepage.

5.4 QuizList Component

Test Case 1: Verify that the QuizList displays a list of quizzes.

- **Pre-Requisite**: QuizList component is rendered.
- **Expected Result**: A list of quizzes is displayed.

Test Case 2: Verify that the QuizList handles an empty quiz list.

- **Pre-Requisite**: QuizList component is rendered with an empty quiz list.
- **Expected Result**: Appropriate message is displayed when no quizzes are available.

5.5 QuizDetail Component

Test Case 1: Verify that the QuizDetail displays quiz information.

- **Pre-Requisite**: QuizDetail component is rendered with quiz data.
- **Expected Result**: Quiz information is displayed correctly.

Test Case 2: Verify that the start quiz button works.

- **Pre-Requisite**: QuizDetail component is rendered with quiz data.
- **Expected Result**: Clicking the start quiz button initiates the quiz.

5.6 QuizTaking Component

Test Case 1: Verify that the QuizTaking component allows answering questions.

- **Pre-Requisite**: QuizTaking component is rendered with quiz data.
- **Expected Result**: Users can answer quiz questions.

Test Case 2: Verify that the QuizTaking component handles quiz submission.

- **Pre-Requisite**: QuizTaking component is rendered with answered questions.
- **Expected Result**: Users can submit the quiz and get results.

5.7 Result Component

Test Case 1: Verify that the Result component displays quiz results.

- **Pre-Requisite**: Result component is rendered with result data.
- **Expected Result**: Quiz results are displayed correctly.

This LLD provides a comprehensive view of the internal design and implementation details for the Quizzy Bee application, ensuring all critical aspects are covered for a successful development process.