RAJALAKSHMI ENGINEERING COLLEGE

An AUTONOMOUS Institution

Affiliated to ANNA UNIVERSITY, Chennai

ONLINE COURSE ENROLLMENT SYSTEM

A MINI PROJECT REPORT

SUBMITTED BY

K SHALINI – 231801162

M R THAVATHARANI - 231801180

CS23332 DATABASE MANAGEMENT SYSTEM

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

RAJALAKSHMI ENGINEERING COLLEGE, THANDALAM.

BONAFIDE CERTIFICATE

Certified that this project report "ONLINE COURSE ENROLLMENT SYSTEM" is the bonafide work of "SHALINI K(231801162), THAVATHARANI M R (231801180) " who carried out the project work under my supervision.

Submitted for the Practical Examination held on ______________________

SIGNATURE	**SIGNATURE**
Dr. GNANASEKAR J M	**Dr. JEYASRI ARCHANADEVI**
Head of the Department, Artificial intelligence and data Science, Rajalakshmi Engineering College (Autonomous),Chennai-602105	**Assoc.Professor, Artificial Intelligence and Data Science, Rajalakshmi Engineering College (Autonomous), Thandalam, Chennai-602105**

INTERNAL EXAMINER **EXTERNAL EXAMINER**

Table of Content

TABLE OF FIGURES

Abstract

The **Online Course Enrollment System** is a web-based application designed to streamline the process of course registration and management for students and instructors. This system enables students to view available courses, enroll in them, and track their enrollment status, while also allowing instructors to manage and update course details. The system is built using **MongoDB** as the database for storing course information, student records, and enrollment data, and **Java** as the primary programming language for backend development . The use of MongoDB allows for a flexible, schema-less design, which can easily accommodate the dynamic nature of course offerings and student enrollments. Java, used for backend development, ensures a robust, scalable, and efficient architecture, supporting the overall user experience. This mini-project provides a practical implementation of a **Database-Driven Application** using **MongoDB** for data storage and **Java** for application logic, demonstrating the integration of web technologies and database management in an educational context.

1. Introduction to ONLINE COURSE ENROLLMENT SYSTEM

1.1 Introduction

In today’s digital era, educational institutions are increasingly adopting online systems for managing various aspects of academic operations, including course enrollment, student management, and course scheduling. Traditional manual processes for handling course registration often lead to inefficiencies, errors, and delays, which can hinder the smooth operation of educational programs. The **Online Course Enrollment System** aims to address these challenges by providing a user-friendly, efficient, and automated solution for students and administrators alike.

This system is designed to simplify the process of **course enrollment** for students while providing instructors and administrators with tools to manage course offerings and student records. Students can easily browse available courses, view detailed information, and register for classes with just a few clicks. Instructors, on the other hand, can create, update, and manage their courses, ensuring that course schedules, descriptions, and available slots are always up-to-date.

The backend of the system is built using **Java**, a widely used and reliable programming language that allows for scalable and maintainable application development. **MongoDB**, a NoSQL database, is employed to store and manage various types of data, such as student profiles, course details, and enrollment records. The flexibility of MongoDB allows for easy storage and retrieval of data without the need for complex relational database schemas, making it ideal for a system that requires frequent updates and changes.

1.2 Objectives

The objective of the Online Course Enrollment System is to create an efficient, user-friendly platform that simplifies the process of course registration and management for both students and instructors. The system aims to enable students to easily browse, enroll in, and manage their courses, while allowing instructors to create, update, and manage course details. By leveraging MongoDB as the database for flexible and scalable data storage, the system ensures efficient handling of student information, course details, and enrollment records. Additionally, the system incorporates secure user authentication, real-time updates for course availability, and a seamless user experience. Ultimately, the goal is to streamline administrative tasks, reduce manual errors, and provide a scalable solution that can grow with the needs of educational institutions.

1.3 Scope

The Online Course Enrollment System is designed to address the needs of both students and instructors in managing course registrations and academic schedules in a streamlined, automated manner. The system's scope encompasses various aspects of course management, starting with student enrollment. Students will be able to register for an account, browse available courses, and enroll in classes based on course type, schedule, and availability. Additionally, students will have the ability to manage their enrollment by adding, dropping, or modifying course selections as needed. For instructors, the system allows for the creation and management of courses, including adding new courses, updating course details (such as timings, syllabus, and available slots), and removing outdated or closed courses. Real-time updates will be implemented to ensure that any changes in course availability, student enrollment status, or course schedules are immediately reflected for both students and instructors, ensuring a smooth and timely experience for all users. The system will leverage MongoDB, a NoSQL database, to store and manage all relevant data, offering flexibility, scalability, and efficiency in handling diverse types of data such as student profiles, course details, and enrollment records. The scope also includes secure user authentication to protect sensitive information and provide role-based access, ensuring that only authorized individuals can access and manage specific features of the system. Ultimately, this system will offer an integrated platform that improves operational efficiency, reduces administrative burden, and enhances the user experience for both students and instructors, all while being adaptable to future needs and growth.

2 System Overview

The **Online Course Enrollment System** is a web-based application designed to automate the process of course registration and management for students and instructors. Students can create accounts, browse available courses, enroll, and manage their schedules. Instructors can create, update, and manage courses, as well as view enrolled students. The system uses **MongoDB** for flexible and scalable data storage, managing student profiles, course details, and enrollments. It provides real-time updates on course availability and enrollment status, ensuring efficient and accurate management of academic data. The platform enhances user experience by streamlining administrative tasks for both students and instructors..

2.1. Database Management (MongoDB)

MongoDB is utilized in the **Online Course Enrollment System** to efficiently manage dynamic data such as student profiles, course details, and enrollments. It offers several key features that make it ideal for this system:

Key Features:

1. **Schema Flexibility**: MongoDB allows for a flexible, schema-less design, making it easy to accommodate changes in course or student data without restructuring the database.
2. **Collections and Documents**: Data is stored in collections (such as **Students**, **Courses**, and **Enrollments**), where each entry is a document. This structure is ideal for storing complex and hierarchical data.
3. **Scalability**: MongoDB can scale horizontally across multiple servers, making it capable of handling growing amounts of data and increasing numbers of users without sacrificing performance.

2.2. Backend Logic (Java)

The backend of the **Online Course Enrollment System** is developed using **Java**, responsible for handling business logic, data processing, and interacting with the **MongoDB** database. Java ensures a robust, scalable, and maintainable architecture, providing core functionality for both students and instructors.

Key responsibilities:

User Authentication: Manage secure login and registration for students and instructors.

Course Enrollment: Handle student enrollment, course selection, and status management.

Course Management: Enable instructors to create, update, and delete courses.

Data Validation: Ensure correctness and consistency of course and student data.

Real-Time Updates: Process and update changes in real-time for course availability and enrollment .

Database Interaction: Perform CRUD operations on MongoDB for managing student and Course data.

2.3. User Interaction

The **Online Course Enrollment System** involves two primary types of users: **Students** and **Instructors**. Each user role has distinct interactions with the system, allowing them to perform specific actions based on their permissions. Below are the key user interactions for both student and instructor roles:

1. Student Interactions:

- **Account Registration and Login**: Students can create an account with personal information and log in using credentials (username/email and password) to access the system.
- **Browse Courses**: Students can search for available courses by subject, instructor, or schedule. They can view detailed information about each course, including descriptions, timings, and available slots.
- **Enroll in Courses**: Students can select a course from the list and enroll, given that there is availability. They can view the status of the courses they have enrolled in.
- **View Enrollment Status**: Students can check their enrolled courses, view schedules, and manage their course list (e.g., drop a course).
- **Notifications**: Students receive notifications for any changes to their enrolled courses, such

as schedule changes, course cancellations, or seat availability updates.

2. Instructor Interactions:

- **Account Registration and Login**: Instructors can register and log in to the system to access their dashboard and course management tools.
- **Create and Manage Courses**: Instructors can create new courses by providing course details (name, description, schedule, etc.) and can modify or delete existing courses.
- **View Enrollments**: Instructors can view the list of students enrolled in their courses, track enrollment status, and manage class sizes or availability.
- **Send Course Updates**: Instructors can send notifications to students about course-related updates, such as schedule changes or new materials.
- **Manage Course Materials**: Instructors can upload course materials, such as syllabi, reading lists, and assignments, for enrolled students to access.

3. Admin (if applicable):

- **Manage Users**: Admins can manage both student and instructor accounts, including creating, updating, and deleting profiles.
- **Monitor Course Data**: Admins can oversee the entire course offering, enrollment statistics, and system performance.

Interaction Flow:

- **Login/Signup → Dashboard → Course Interaction (Browse/Enroll/Manage) → Real-time Notifications/Updates**

3 Survey of Technologies

The Online Course Enrollment System leverages a combination of technologies to ensure a smooth, scalable, and efficient solution for both students and instructors. Below is a survey of the key technologies used in the system:

3.1. MongoDB

MongoDB is a NoSQL database designed for flexibility, scalability, and high performance. Unlike traditional relational databases, MongoDB stores data in JSON-like documents, offering a schema-less structure that adapts to dynamic applications like quizzes.

Key Features:

- Document-Based Storage: Data is stored in BSON (Binary JSON), making it easy to model complex structures like questions and options.
- Schema Flexibility: MongoDB allows adding or modifying fields without restructuring the database, ideal for dynamic applications.
- Scalability: Built for horizontal scaling, MongoDB can handle growing datasets and increased traffic efficiently.
- Indexing and Querying: MongoDB supports advanced queries and indexing for fast and efficient data retrieval.

Role in the Project:

In the **Online Course Enrollment System**, MongoDB is used to store quiz questions in a collection named questions. Each question includes fields for the text, multiple-choice options, the correct answer, and category. This structure allows dynamic retrieval and updates without the rigidity of fixed schemas in relational databases.

Advantages for the Project:

- Easy addition of new questions or categories.
- Fast query performance for fetching quiz data.
- Simplified data management compared to traditional SQL databases.

3.2. Java

Java is a robust, platform-independent object-oriented programming language widely used for application development. Its flexibility and extensive library ecosystem make it an ideal choice for backend development in this project.

Key Features:

- Platform Independence: Java's "Write Once, Run Anywhere" feature ensures the application can run on multiple platforms without modification.
- Rich Libraries: Provides a vast collection of libraries for handling database connections, I/O operations, and user interaction.
- Scalability: Java's modular design supports the development of scalable and maintainable applications.
- Concurrency Support: Enables efficient handling of multiple tasks, ensuring smooth application performance.

Role in the Project:

In this project, Java is used to:Connect to MongoDB using the MongoDB Java Driver.Retrieve quiz questions dynamically and display them to the user.Validate user responses and compute scores in real-time.Advantages for the Project: Robust error handling for a seamless user experience. Easy integration with MongoDB for dynamic data operations. Modular coding structure for maintainability and scalability.

3.3. MongoDB Java Driver

The MongoDB Java Driver is a library that facilitates interaction between Java applications and MongoDB databases. It provides APIs for connecting to the database, performing CRUD operations, and executing queries.

Key Features:

- Easy Connection Setup: Simplifies establishing a connection to MongoDB.
- CRUD Operations: Supports Create, Read, Update, and Delete operations on MongoDB collections.
- Advanced Querying: Allows the execution of complex queries for precise data retrieval.

Role in the Project:

The MongoDB Java Driver is used to establish a connection to the MongoDB database, retrieve quiz data from the questions collection, and update the database as needed.

3.4. Eclipse IDE

Eclipse is a widely-used Integrated Development Environment (IDE) for Java development. It offers tools and features that simplify the coding, debugging, and deployment process.

Key Features:

- Code Editor: Supports intelligent code suggestions, syntax highlighting, and error detection.
- Integrated Build Tools: Simplifies dependency management with Maven integration.
- Debugging Tools: Provides advanced debugging capabilities to identify and fix issues efficiently.

Role in the Project:

Eclipse is used to write and manage the Java code for the application. It streamlines the development process by offering features like real-time error detection and integrated support for testing and debugging.

3.5. Maven

Maven is a build automation and dependency management tool for Java projects. It simplifies the process of adding external libraries and managing project configurations.

Key Features:

- Dependency Management: Automatically downloads and configures libraries like the MongoDB Java Driver.
- Project Configuration: Standardizes project structure and build processes.
- Build Automation: Facilitates compiling, testing, and packaging the application.

Role in the Project:

In this project, Maven is used to manage dependencies like the MongoDB Java Driver, ensuring they are correctly integrated into the project.

4 Requirements and Analysis

4.1 Functional Requirements

User Management

- User Registration: Allow users (students and instructors) to register with details like

- name, email, password, and role (student or instructor).
- User Authentication: Enable login/logout functionality using email and password.
- Profile Management: Allow users to update their profiles (e.g., change name, password).

2. **Course Management**

- Create Courses (Instructor):
  - Instructors can create a new course with fields such as course title, description, duration, fee, and prerequisites.
- View All Courses:
  - Students and instructors can view the list of available courses.
- Update Courses (Instructor):
  - Instructors can edit course details they created.
- Delete Courses (Instructor):
  - Instructors can delete their courses if no students are enrolled.

3. **Enrollment Management**

- Enroll in a Course (Student):
  - Students can enroll in a course by selecting it from the available list.
- Drop a Course (Student):
  - Students can unenroll from a course they are currently enrolled in.
- View Enrollments (Student):
  - Students can view all courses they have enrolled in.
- Manage Enrollments (Instructor):
  - Instructors can view the list of students enrolled in their courses.

4. **Search and Filtering**

- Search Courses:
  - Allow users to search courses by title, instructor name, or category.
- Filter Courses:
  - Provide filters for categories like fee range, duration, and prerequisites.

5. **Notifications**

- Enrollment Notifications: Notify students of successful enrollments.
- Course Update Notifications: Notify enrolled students if their instructor updates the course details.

6. **Reporting**

- Student Report:
  - View all courses a student has enrolled in with their progress status.

- Course Report:
    - Instructors can view enrollment statistics for their courses.

7. **Admin Management (Optional)**

- Admin Login:
    - Admins can log in to manage users and courses.
- User Management:
    - Admins can view, update, and delete user accounts.
- Course Moderation:
    - Admins can remove courses violating policies.

8. **MongoDB Integration**

- Store all data in MongoDB collections:
    - users: User information (students and instructors).
    - courses: Course details.
    - enrollments: Information about which student is enrolled in which course.

4.2 Non-Functional Requirements

1. **Performance**
    - Fast and efficient retrieval of quiz questions from the database.
    - Real-time processing of user inputs and score calculations.
2. **Reliability**
    - Ensuring consistent and accurate quiz results.
    - Robust error handling for system stability.
3. **Usability**
    - Intuitive design for both end-users and administrators.
    - Compatibility across multiple devices and screen resolutions.
4. **Security**
    - Secure handling of sensitive user data.

   - Preventing unauthorized access to quiz content and administrative features.

5. **Maintainability**
   - Modular code structure for easy updates and bug fixes.
   - Comprehensive documentation for developers and administrators.

4.3 System Analysis

1. **Problem Definition**

   Managing course enrollment manually is inefficient and error-prone. Students face difficulties in finding and enrolling in courses, while instructors struggle to organize and track enrollments. This project aims to develop an **Online Course Enrollment System** using **Java** and **MongoDB** to enable seamless course creation, enrollment, and management for students and instructors.

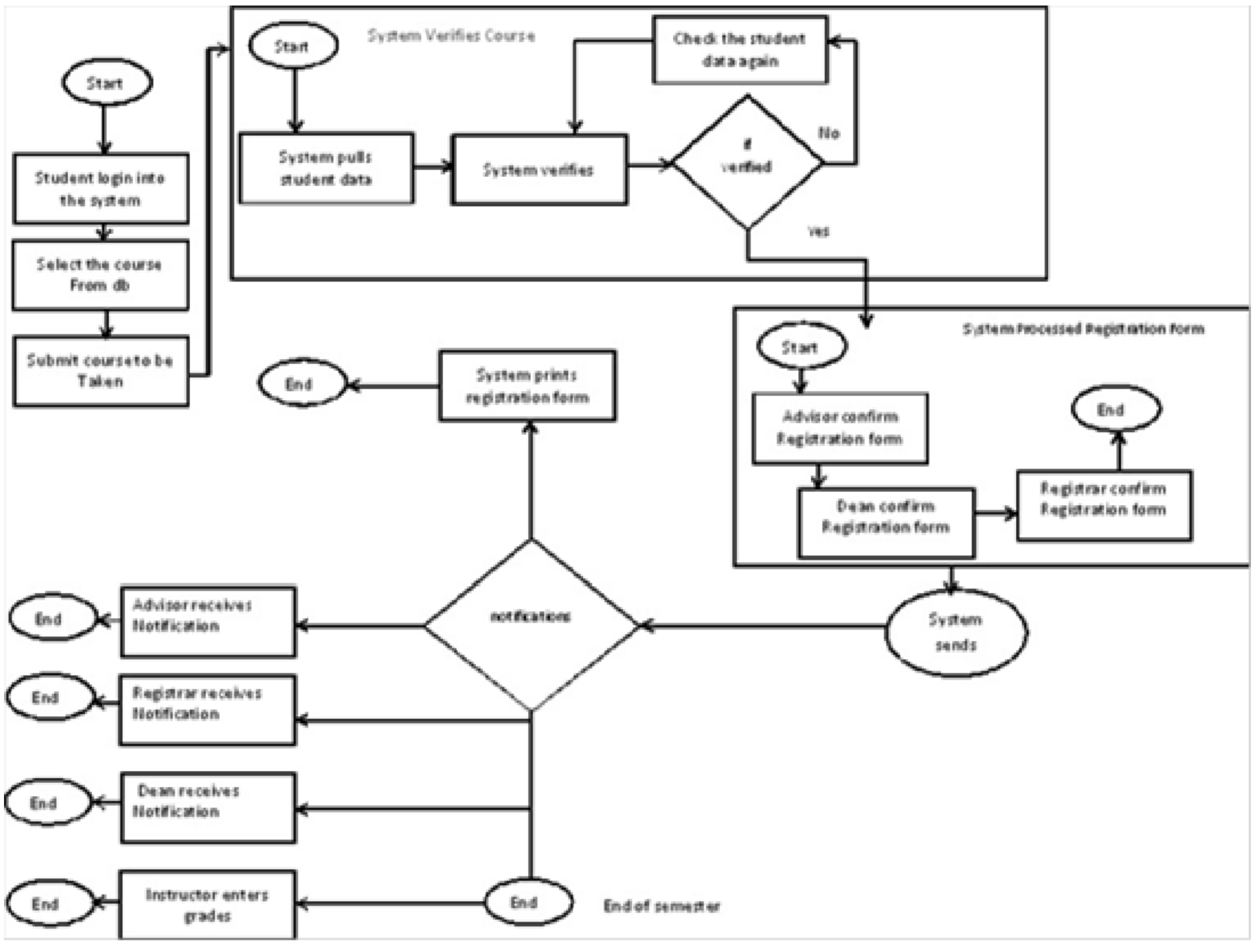
2. **Feasibility Study**
   - **Technical Feasibility**: Leverages Java for core logic and MongoDB for scalable, NoSQL database management.
   - **Operational Feasibility**: Intuitive design ensures ease of use for both users and administrators.
   - **Economic Feasibility**: Minimal cost for development tools, focusing on open-source technologies.

3. **System Design Considerations**
   - Modular architecture to simplify integration and feature addition.
   - Separation of user and admin functionalities for better role management.

4. **Target Users**

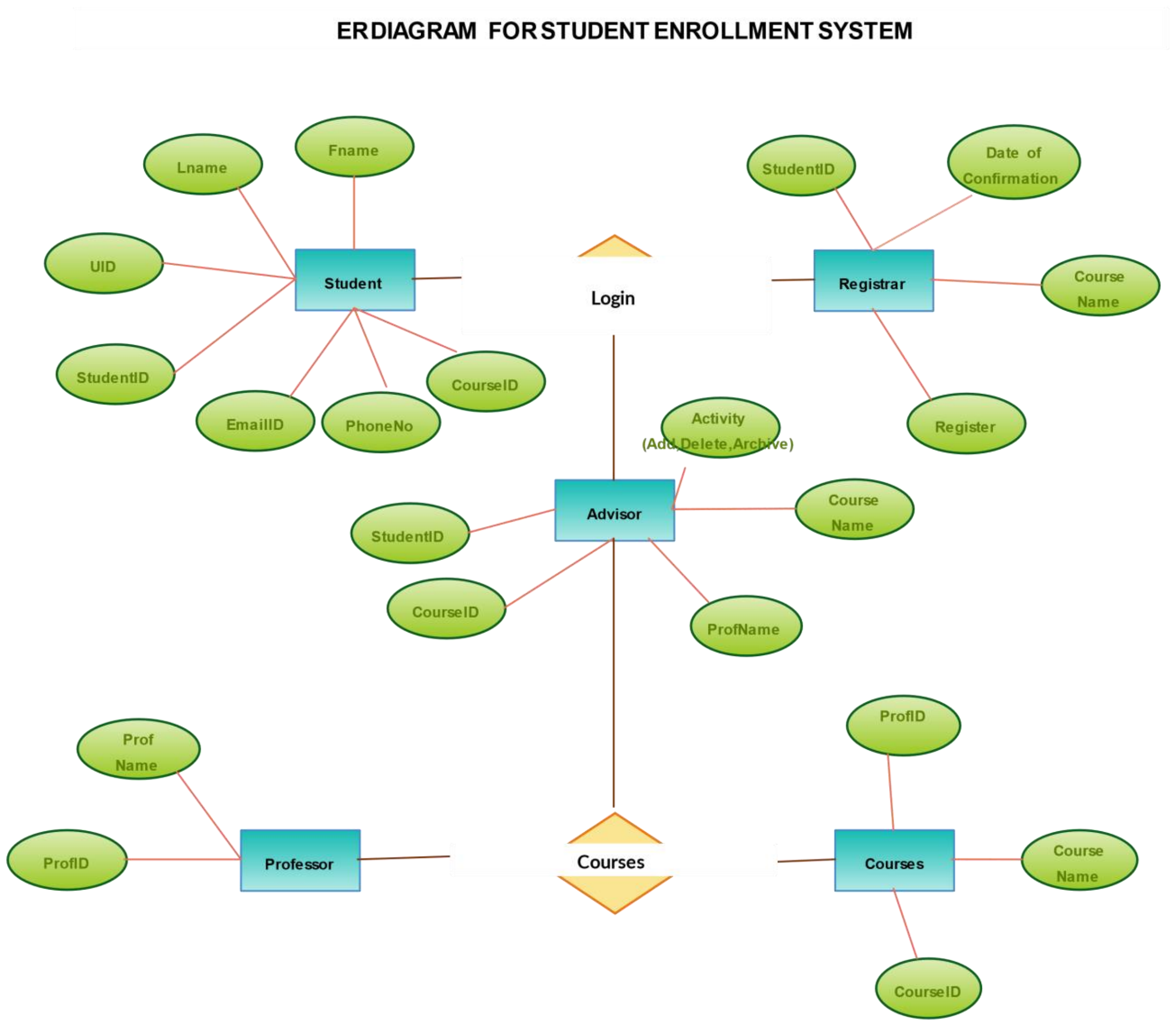
- Students and learners seeking self-assessment tools.
- Educational institutions and organizations for training purposes.
- General users interested in improving knowledge across domains.

4.4 Architecture diagram


Start
Student login into the system
Select the course From db
Submit course to be Taken
System Verifies Course
Start
System pulls student data
System verifies
Check the student data again
if verified
No
Yes
System Processed Registration form
Start
Advisor confirm Registration form
Dean confirm Registration form
Registrar confirm Registration form
End
End
System prints registration form
notifications
System sends
End
Advisor receives Notification
End
Registrar receives Notification
End
Dean receives Notification
End
Instructor enters grades
End
End of semester

[illegible]

4.5 ER diagram



5 System Design for online course enrollment system

The system design for online course enrollment system focuses on key components that enable efficient data storage, user interaction, and smooth process flow. The design includes database design, UI design overview, and workflow diagrams, collectively outlining the structure, functionality, and user experience.

5.1 Database Design and Tables

The database design ensures efficient storage, retrieval, and management of quiz-related data. Proper normalization is applied to maintain data integrity and minimize redundancy.

5.1.1 Entity Relationships

The core entities in the system include *Quiz, Question, User*, *Category*, *Score Record*, and *Admin*. These entities have relationships such as:

- Quiz-Question (one-to-many): A quiz can contain multiple questions.
- User-Score Record (one-to-many): A user can have multiple score records for different quiz attempts.
- Category-Question (one-to-many): A category can encompass multiple questions.

5.1.2 Table Structures

Users Collection: Stores details of students and instructors.

Fields: _id, name, email, password, role, created_at, updated_at.

Courses Collection : Stores course information created by instructors.

Fields: _id, title, description, duration, fee, prerequisites, instructor_id, created_at, updated_at.

Enrollments Collection:Tracks student enrollments in courses.

Fields: _id, student_id, course_id, enrollment_date, progress.

These collections form relationships to manage users, courses, and enrollments effectively.

5.1.3 Key Fields and Relationships

- Admin Table: Contains admin credentials for managing quizzes, questions, and categories.
- Question-Quiz Relationship: Links questions to the quizzes they belong to, allowing dynamic retrieval during a quiz session.
- User-Score Relationship: Associates users with their performance records for personalized tracking.

5.2 UI Design Overview

The User Interface (UI) design ensures a seamless and intuitive experience for quiz participants, administrators, and other users. The design emphasizes simplicity, easy navigation, and minimal distractions.

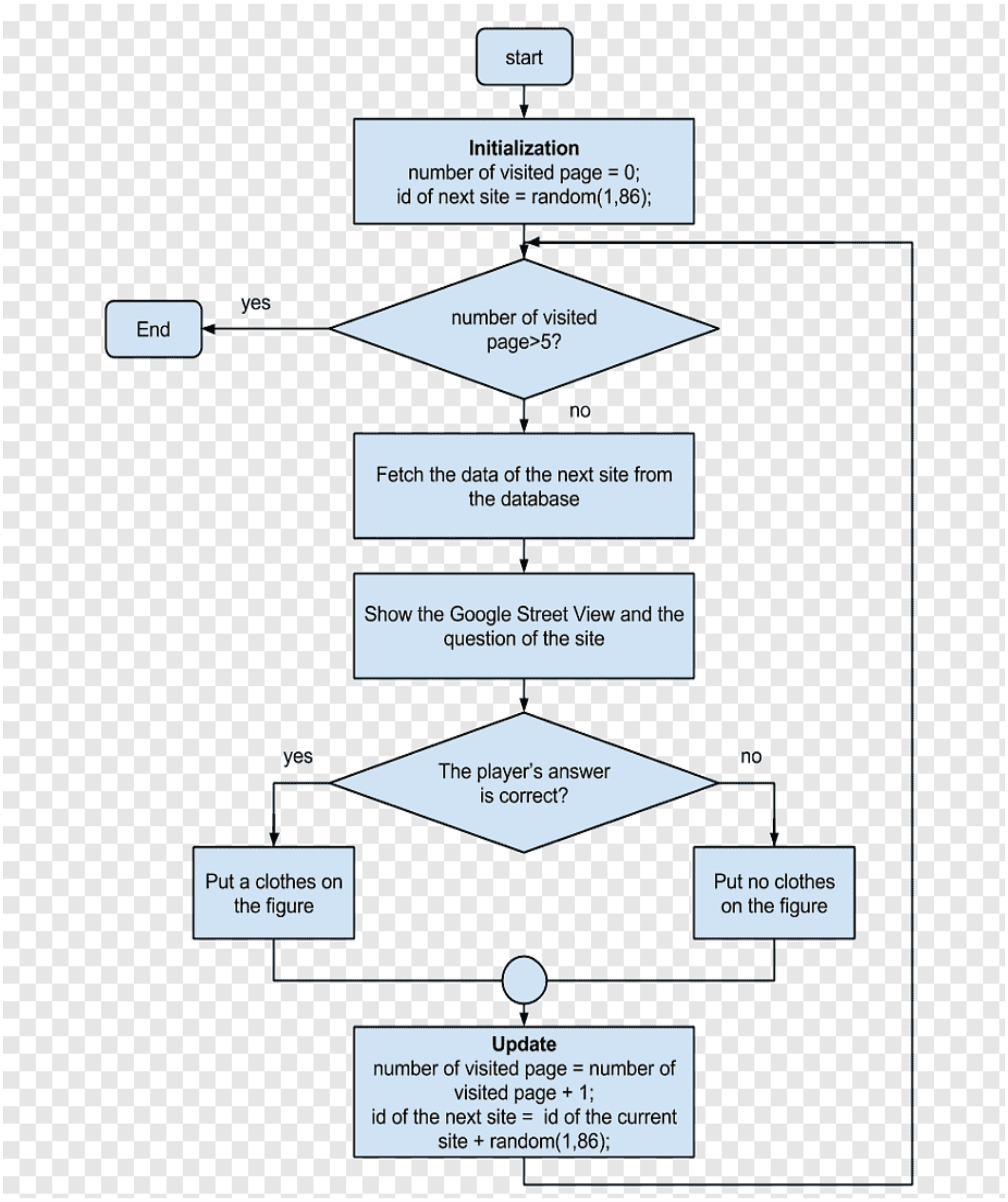
5.2.1 Dashboard and Navigation

- Centralized Dashboard: Provides an overview of available quizzes, user scores, and recent activity. Admins can access quiz creation and management tools from the same dashboard.
- Sidebar Navigation: Enables quick access to sections such as Quiz Selection, Categories, Performance Reports, and User Settings.

5.2.3 Admin Panel

- A dedicated interface for administrators to create, update, or delete quizzes, questions, and categories.
- Features tools for monitoring user performance and exporting data for reports.

5.3. Workflow and Process Diagram


start
Initialization
number of visited page = 0;
id of next site = random(1,86);
number of visited page>5?
yes
End
no
Fetch the data of the next site from the database
Show the Google Street View and the question of the site
The player's answer is correct?
yes
no
Put a clothes on the figure
Put no clothes on the figure
Update
number of visited page = number of visited page + 1;
id of the next site = id of the current site + random(1,86);

6. Implementation Code Structure and Organization

The Java online course enrollment system is built using a modular and organized code structure to ensure maintainability, scalability, and clarity. The system is divided into multiple components, each responsible for a specific aspect of quiz functionality. The project includes backend, frontend, database, and utility services, each fulfilling distinct responsibilities.

6.1 Overall Project Structure

The Java online course enrollment system is structured to separate concerns for better maintainability and development efficiency. The main sections are:

1. **Backend**: Handles the core logic, including database interactions, business rules, and exposing data to the frontend through APIs.
2. **Frontend**: Manages user interaction and displays quiz content, user scores, and other UI components.
3. **Database**: Stores all data related to quizzes, questions, users, and performance records.
4. **Configuration & Utilities**: Includes configuration files, utility functions, and external integrations, such as logging or connecting to cloud services.
5. **Testing**: Implements tests for backend APIs, frontend components, and integration flows to ensure system reliability.

The project follows a **Model-View-Controller (MVC)** or modular architecture to separate business logic, data handling, and user interaction into distinct layers.

6.2 Backend Code Structure

The backend is responsible for handling business logic, interacting with the database, and exposing functionality via API endpoints. It is divided into the following key sections:

6.2.1 Controllers

UserController:

- Register: POST /users/register
- Login: POST /users/login
- Update Profile: PUT /users/{id}
- Get User: GET /users/{id}

CourseController:

- Create Course: POST /courses
- Update Course: PUT /courses/{id}
- Delete Course: DELETE /courses/{id}
- Get All Courses: GET /courses
- Get Course: GET /courses/{id}

EnrollmentController:

- Enroll: POST /enrollments
- Drop Course: DELETE /enrollments/{id}
- Get Student Enrollments: GET /enrollments/student/{student_id}
- Get Course Enrollments: GET /enrollments/course/{course_id}

6.2.2 Models

Models define the data structure and handle database interactions. Each model corresponds to a database collection or table:

User Model:

Represents students and instructors with fields:

- id, name, email, password, role, createdAt, updatedAt.

Course Model:

Represents courses with fields:

- id, title, description, duration, fee, prerequisites, instructorId, createdAt, updatedAt.

Enrollment Model:

Tracks course enrollments with fields:

- id, studentId, courseId, enrollmentDate, progress.

6.2.4 Services

UserService:

Handles user-related logic.

- registerUser(User user): Registers a new user.
- login(String email, String password): Authenticates the user.
- updateUser(String id, User user): Updates user details.

- getUserById(String id): Retrieves user information.

CourseService:

Manages course operations.
- createCourse(Course course): Adds a new course.
- updateCourse(String id, Course course): Updates course details.
- deleteCourse(String id): Removes a course.
- getAllCourses(): Retrieves all courses.
- getCourseById(String id): Fetches a specific course.

EnrollmentService:
Handles enrollment functionality.
- enrollStudent(String studentId, String courseId): Enrolls a student in a course.
- dropCourse(String enrollmentId): Removes a student's enrollment.
- getEnrollmentsByStudent(String studentId): Retrieves all enrollments for a student.
- getEnrollmentsByCourse(String courseId): Retrieves all students enrolled in a course.

These services ensure the business logic is separate from the controllers.

6.3 Frontend Code Structure

The frontend is responsible for delivering an interactive user interface where users can take quizzes, view scores, and navigate the application.

6.3.1 User Interface Components

Login/Register Page:
- Login: Email, Password, Login Button.

- Register: Name, Email, Password, Role (Student/Instructor).

Dashboard:

- Student: View enrolled courses, browse available courses, enroll/drop courses.
- Instructor: Create/manage courses, view enrollment stats.

Course Page:

- Display course details (title, description, fee, etc.).
- Enroll (students) or Edit/Delete (instructors).

Profile Page:

- View and update user details like name and password.

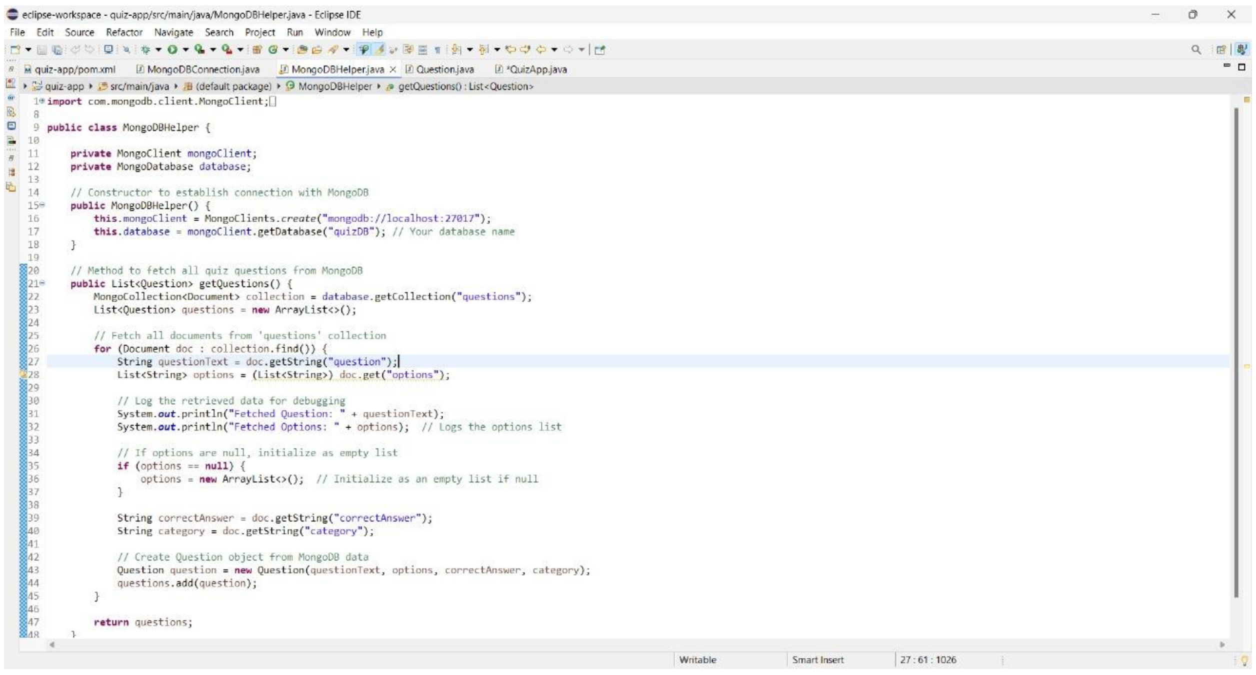
Optional Admin Panel:

- Manage users, courses, and enrollments.

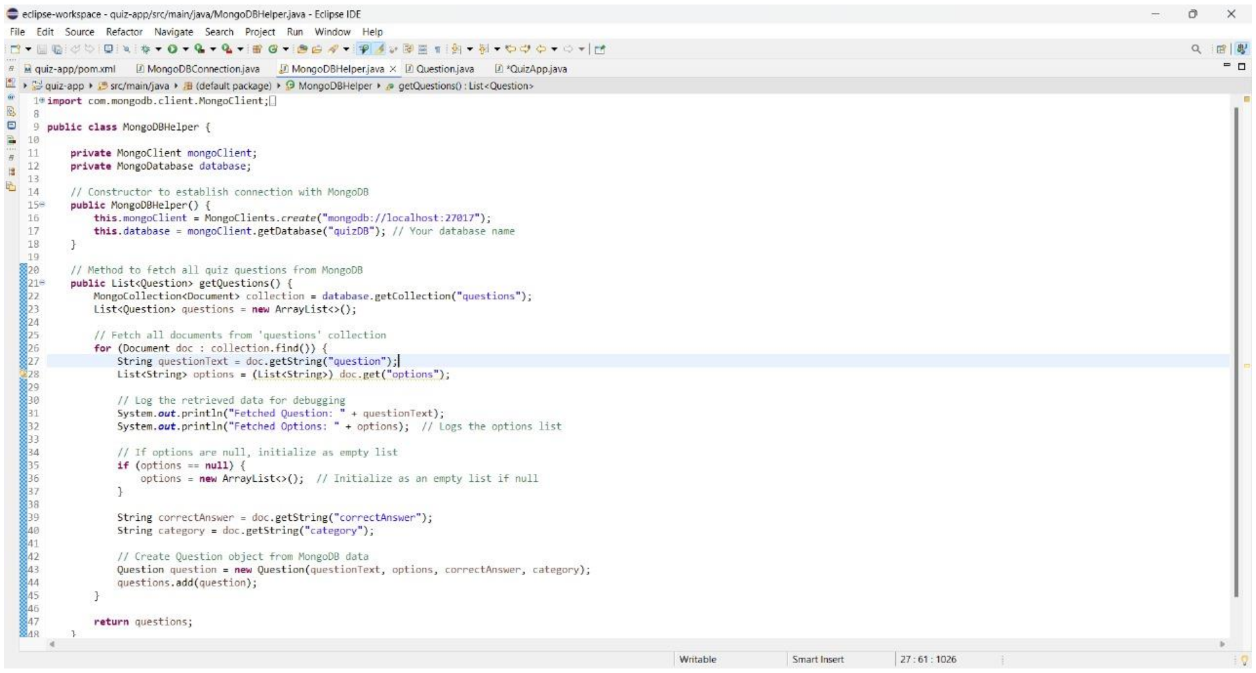
Tools: HTML, CSS, JavaScript (or React/Angular), Bootstrap for responsiveness.

6.3.3 Responsive Design

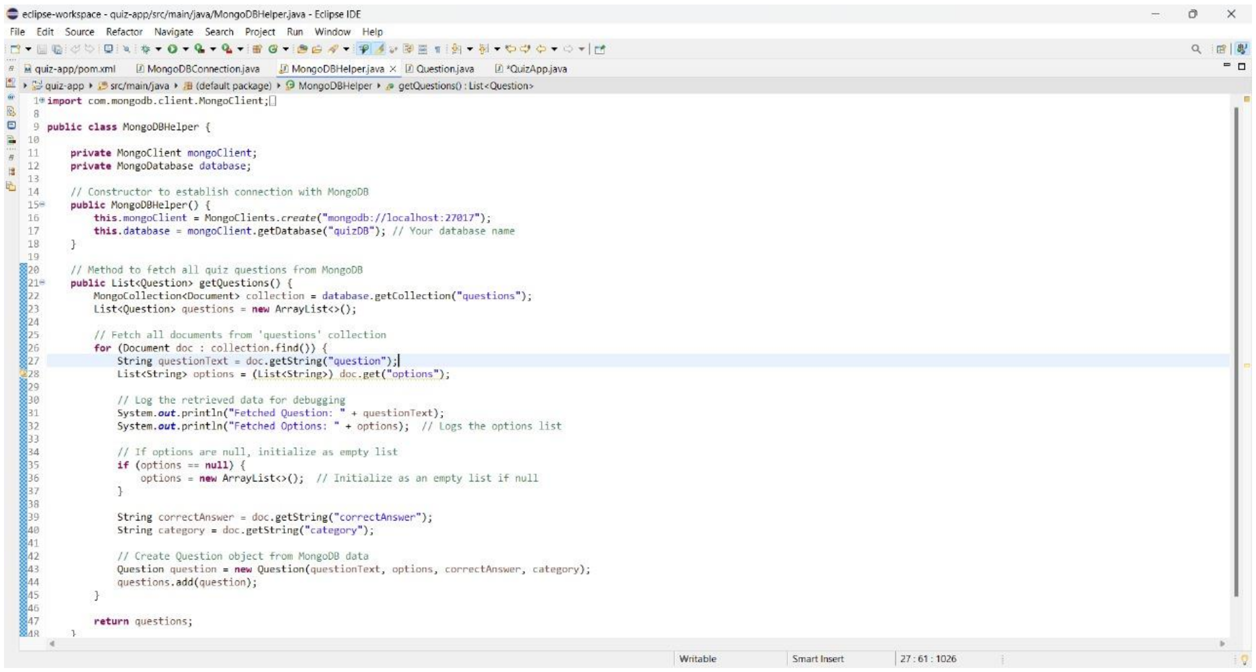
The frontend ensures compatibility across devices, providing a seamless experience on desktops, tablets, and mobile devices.


eclipse-workspace - quiz-app/src/main/java/MongoDBHelper.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
quiz-app/pom.xml
MongoDBConnection.java
MongoDBHelper.java
Question.java
*QuizApp.java
quiz-app › src/main/java › (default package) › MongoDBHelper › getQuestions() : List<Question>
Writable
Smart Insert
27 : 61 : 1026

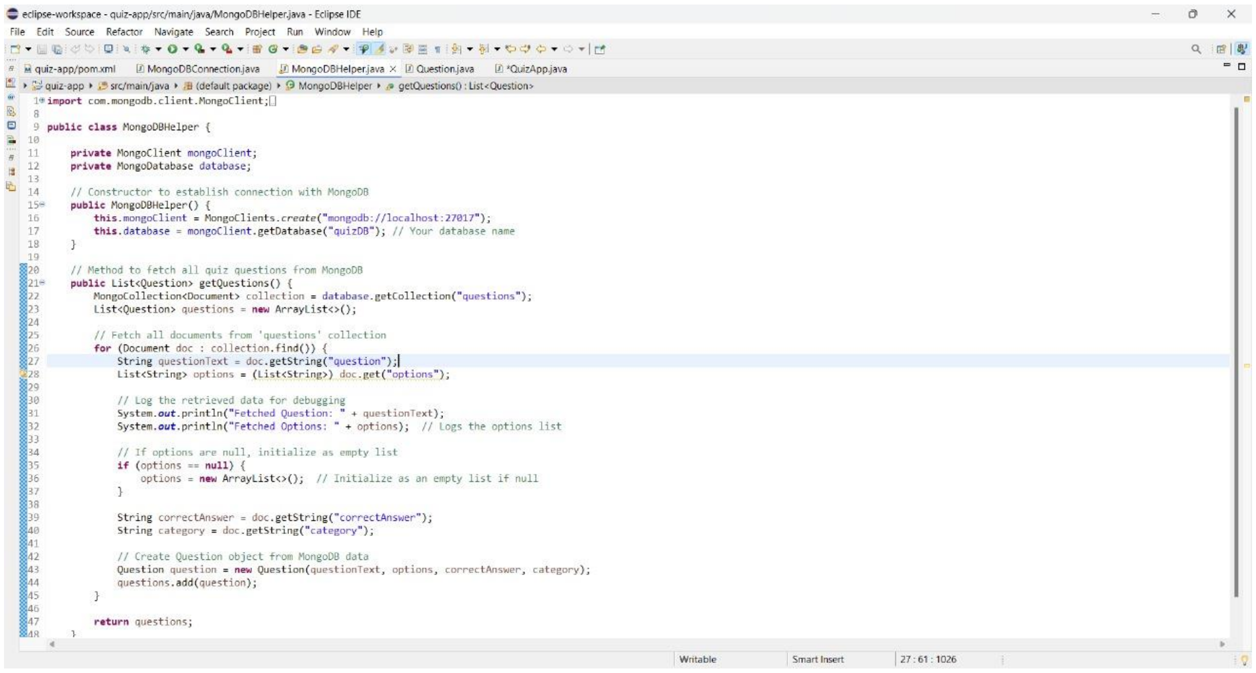
```java
import com.mongodb.client.MongoClient;

public class MongoDBHelper {

    private MongoClient mongoClient;
    private MongoDatabase database;

    // Constructor to establish connection with MongoDB
    public MongoDBHelper() {
        this.mongoClient = MongoClients.create("mongodb://localhost:27017");
        this.database = mongoClient.getDatabase("quizDB"); // Your database name
    }

    // Method to fetch all quiz questions from MongoDB
    public List<Question> getQuestions() {
        MongoCollection<Document> collection = database.getCollection("questions");
        List<Question> questions = new ArrayList<>();

        // Fetch all documents from 'questions' collection
        for (Document doc : collection.find()) {
            String questionText = doc.getString("question");
            List<String> options = (List<String>) doc.get("options");

            // Log the retrieved data for debugging
            System.out.println("Fetched Question: " + questionText);
            System.out.println("Fetched Options: " + options);  // Logs the options list

            // If options are null, initialize as empty list
            if (options == null) {
                options = new ArrayList<>();  // Initialize as an empty list if null
            }

            String correctAnswer = doc.getString("correctAnswer");
            String category = doc.getString("category");

            // Create Question object from MongoDB data
            Question question = new Question(questionText, options, correctAnswer, category);
            questions.add(question);
        }

        return questions;
    }
```


eclipse-workspace - quiz-app/src/main/java/MongoDBHelper.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
quiz-app/pom.xml
MongoDBConnection.java
MongoDBHelper.java
Question.java
*QuizApp.java
quiz-app › src/main/java › (default package) › MongoDBHelper › getQuestions() : List<Question>
Writable
Smart Insert
27 : 61 : 1026

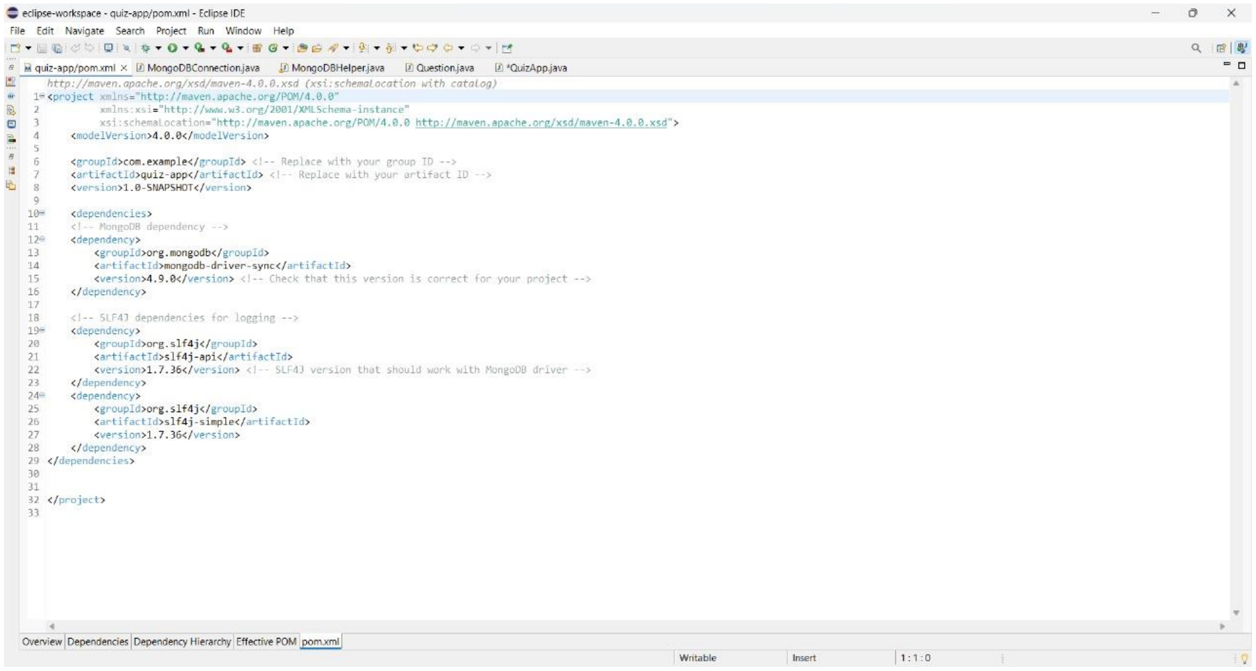
```java
import com.mongodb.client.MongoClient;[]

public class MongoDBHelper {

    private MongoClient mongoClient;
    private MongoDatabase database;

    // Constructor to establish connection with MongoDB
    public MongoDBHelper() {
        this.mongoClient = MongoClients.create("mongodb://localhost:27017");
        this.database = mongoClient.getDatabase("quizDB"); // Your database name
    }

    // Method to fetch all quiz questions from MongoDB
    public List<Question> getQuestions() {
        MongoCollection<Document> collection = database.getCollection("questions");
        List<Question> questions = new ArrayList<>();

        // Fetch all documents from 'questions' collection
        for (Document doc : collection.find()) {
            String questionText = doc.getString("question");
            List<String> options = (List<String>) doc.get("options");

            // Log the retrieved data for debugging
            System.out.println("Fetched Question: " + questionText);
            System.out.println("Fetched Options: " + options);  // Logs the options list

            // If options are null, initialize as empty list
            if (options == null) {
                options = new ArrayList<>();  // Initialize as an empty list if null
            }

            String correctAnswer = doc.getString("correctAnswer");
            String category = doc.getString("category");

            // Create Question object from MongoDB data
            Question question = new Question(questionText, options, correctAnswer, category);
            questions.add(question);
        }

        return questions;
    }
```



```java
import com.mongodb.client.MongoClient;[]

public class MongoDBHelper {

    private MongoClient mongoClient;
    private MongoDatabase database;

    // Constructor to establish connection with MongoDB
    public MongoDBHelper() {
        this.mongoClient = MongoClients.create("mongodb://localhost:27017");
        this.database = mongoClient.getDatabase("quizDB"); // Your database name
    }

    // Method to fetch all quiz questions from MongoDB
    public List<Question> getQuestions() {
        MongoCollection<Document> collection = database.getCollection("questions");
        List<Question> questions = new ArrayList<>();

        // Fetch all documents from 'questions' collection
        for (Document doc : collection.find()) {
            String questionText = doc.getString("question");
            List<String> options = (List<String>) doc.get("options");

            // Log the retrieved data for debugging
            System.out.println("Fetched Question: " + questionText);
            System.out.println("Fetched Options: " + options);  // Logs the options list

            // If options are null, initialize as empty list
            if (options == null) {
                options = new ArrayList<>();  // Initialize as an empty list if null
            }

            String correctAnswer = doc.getString("correctAnswer");
            String category = doc.getString("category");

            // Create Question object from MongoDB data
            Question question = new Question(questionText, options, correctAnswer, category);
            questions.add(question);
        }

        return questions;
    }
```


eclipse-workspace - quiz-app/src/main/java/MongoDBHelper.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
quiz-app/pom.xml
MongoDBConnection.java
MongoDBHelper.java
Question.java
*QuizApp.java
quiz-app › src/main/java › (default package) › MongoDBHelper › getQuestions() : List<Question>
Writable
Smart Insert
27 : 61 : 1026

```java
import com.mongodb.client.MongoClient;

public class MongoDBHelper {

    private MongoClient mongoClient;
    private MongoDatabase database;

    // Constructor to establish connection with MongoDB
    public MongoDBHelper() {
        this.mongoClient = MongoClients.create("mongodb://localhost:27017");
        this.database = mongoClient.getDatabase("quizDB"); // Your database name
    }

    // Method to fetch all quiz questions from MongoDB
    public List<Question> getQuestions() {
        MongoCollection<Document> collection = database.getCollection("questions");
        List<Question> questions = new ArrayList<>();

        // Fetch all documents from 'questions' collection
        for (Document doc : collection.find()) {
            String questionText = doc.getString("question");
            List<String> options = (List<String>) doc.get("options");

            // Log the retrieved data for debugging
            System.out.println("Fetched Question: " + questionText);
            System.out.println("Fetched Options: " + options);  // Logs the options list

            // If options are null, initialize as empty list
            if (options == null) {
                options = new ArrayList<>();  // Initialize as an empty list if null
            }

            String correctAnswer = doc.getString("correctAnswer");
            String category = doc.getString("category");

            // Create Question object from MongoDB data
            Question question = new Question(questionText, options, correctAnswer, category);
            questions.add(question);
        }

        return questions;
    }
```



```xml
http://maven.apache.org/xsd/maven-4.0.0.xsd (xsi:schemaLocation with catalog)
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.example</groupId> <!-- Replace with your group ID -->
    <artifactId>quiz-app</artifactId> <!-- Replace with your artifact ID -->
    <version>1.0-SNAPSHOT</version>

    <dependencies>
    <!-- MongoDB dependency -->
    <dependency>
        <groupId>org.mongodb</groupId>
        <artifactId>mongodb-driver-sync</artifactId>
        <version>4.9.0</version> <!-- Check that this version is correct for your project -->
    </dependency>

    <!-- SLF4J dependencies for logging -->
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-api</artifactId>
        <version>1.7.36</version> <!-- SLF4J version that should work with MongoDB driver -->
    </dependency>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-simple</artifactId>
        <version>1.7.36</version>
    </dependency>
</dependencies>


</project>
```

Testing and Validation

7.1 Testing Strategies

To ensure the Java online course enrollment system is robust, reliable, and meets the defined requirements, various testing strategies are implemented at different stages of development:

7.1.1 Unit Testing

Unit testing involves testing individual components or functions of the application in isolation. This includes:

- Verifying backend logic, such as question randomization, score calculations, and user authentication.
- Testing utility functions, like timer handling and validation of user inputs.
- Ensuring correct functionality of database operations, such as creating, retrieving, updating, or deleting quiz questions.

7.1.2 Integration Testing

Integration testing focuses on verifying that different modules of the **online course enrollment system** work seamlessly together. For example:

- Testing the interaction between the frontend and backend to ensure quiz questions are fetched correctly.
- Validating communication between the backend and database to check if user scores are accurately stored and retrieved.

7.1.4 Functional Testing

Functional testing ensures that the Java online course enrollment system performs according to the defined requirements. Key functionalities tested include:

- Starting and completing a course.
- Dynamic question loading and real-time score calculation.
- Admin capabilities like creating new quizzes, editing questions, and managing categories.

7.1.5 Performance Testing

Performance testing evaluates the application's speed, scalability, and responsiveness. Scenarios include:

- Simulating multiple users taking courses simultaneously.
- Testing the application with large datasets, such as thousands of course questions or user records.
- Measuring response times for user actions like starting a course or viewing results.

7.2 Functional Test Cases and Results

This section evaluates the results of functional and performance testing to assess the system's effectiveness and identify areas for improvement.

7.2.1 Functional Test Results

Each core module of the Java online course enrollment system was tested for functionality:

- **Course Module**: Successfully fetched and displayed quiz questions dynamically, tracked user progress, and calculated scores accurately.
- **Admin Module**: Enabled the creation, modification, and deletion of quizzes and questions without errors.
- **User Module**: Supported seamless registration, login, and tracking of user performance.

- **Score Tracking**: Displayed real-time scores upon quiz completion and allowed users to review answers.

7.2.2 Performance Testing Results

Performance tests highlighted the application's ability to handle concurrent users and large datasets:

- With up to **50 concurrent users**, the system maintained response times under 2 seconds for most actions.
- Data retrieval for quizzes and score reports was consistently under **3 seconds**, even with a database containing over 10,000 entries.
- During stress testing with **100 simultaneous users**, minor slowdowns occurred, but the system remained stable without crashes or data loss.

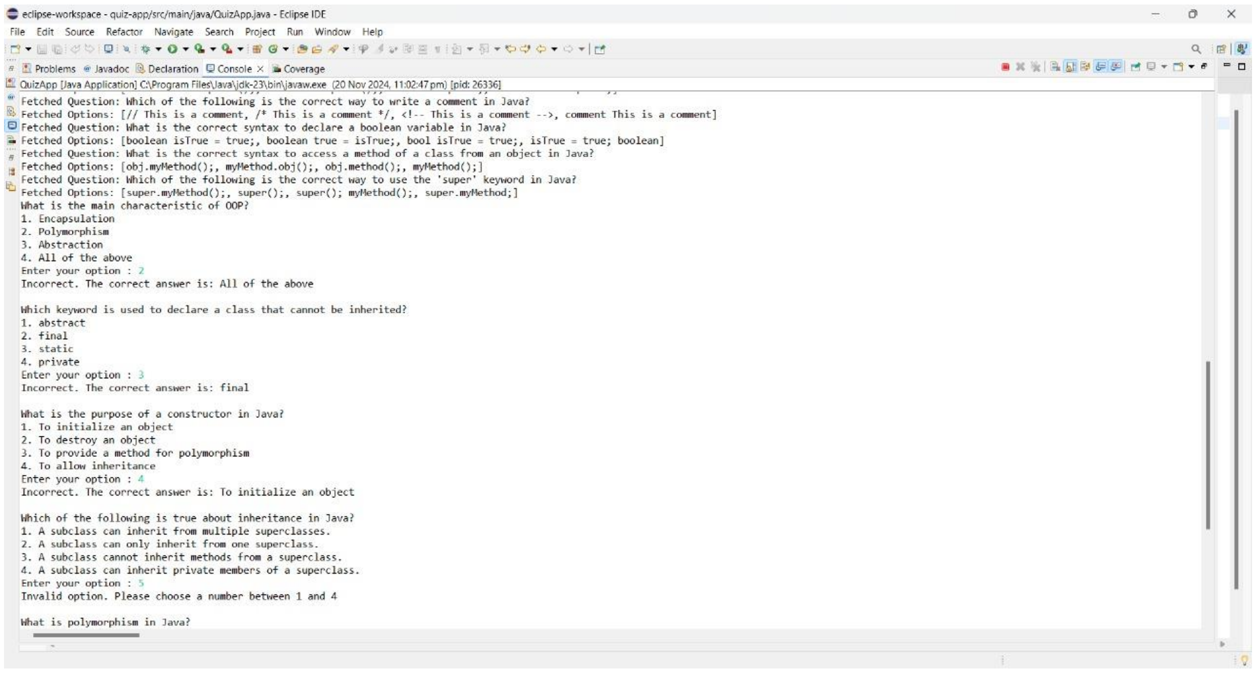
7.3 Challenges and Limitations

While the Java online course enrollment system performed well, a few challenges and limitations were identified:

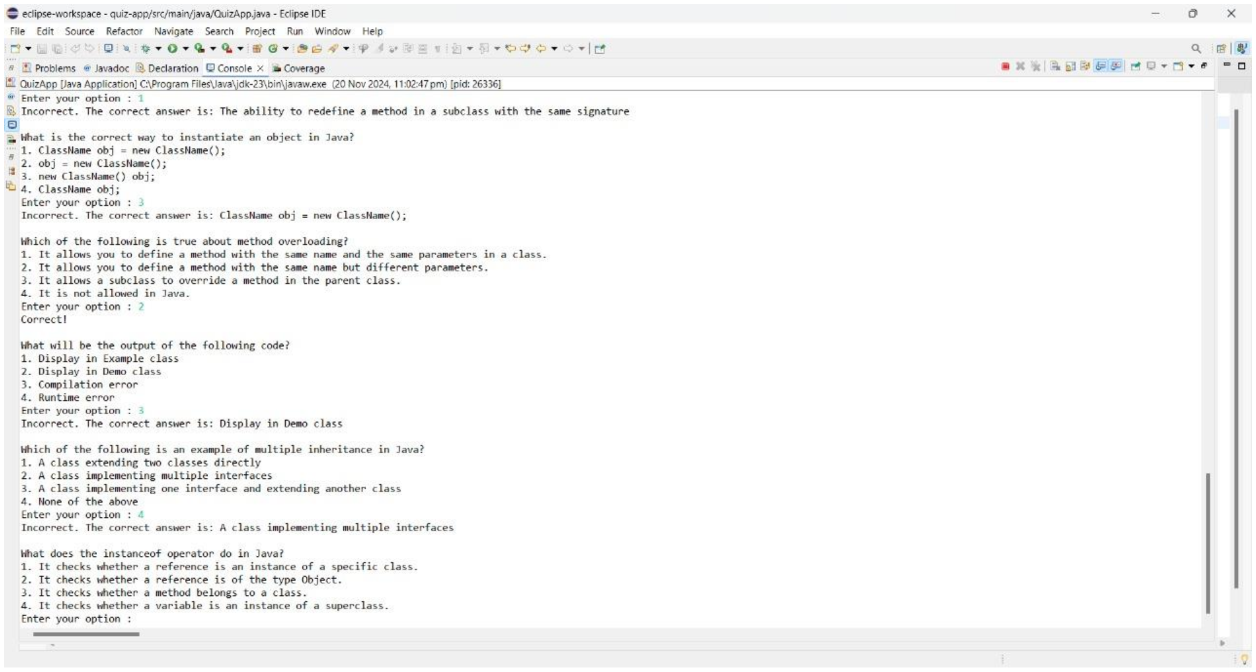
1. **Browser Compatibility**: Minor issues were observed with older browsers not supporting modern JavaScript features.

2. **Offline Functionality**: Users in areas with intermittent internet connections experienced difficulties. Adding offline capabilities, such as local quiz storage, would address this limitation.
3. **User Training**: Some users, especially first-time participants, required guidance on navigating the quiz interface.
4. **Database Scaling**: With a rapidly growing user base, performance optimizations for database queries may be necessary to ensure scalability.

8 Output


eclipse-workspace - quiz-app/src/main/java/QuizApp.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Console Coverage

```
QuizApp [Java Application] C:\Program Files\Java\jdk-23\bin\javaw.exe (20 Nov 2024, 11:02:47 pm) [pid: 26336]
Fetched Question: Which of the following is the correct way to write a comment in Java?
Fetched Options: [// This is a comment, /* This is a comment */, <!-- This is a comment -->, comment This is a comment]
Fetched Question: What is the correct syntax to declare a boolean variable in Java?
Fetched Options: [boolean isTrue = true;, boolean true = isTrue;, bool isTrue = true;, isTrue = true; boolean]
Fetched Question: What is the correct syntax to access a method of a class from an object in Java?
Fetched Options: [obj.myMethod();, myMethod.obj();, obj.method();, myMethod();]
Fetched Question: Which of the following is the correct way to use the 'super' keyword in Java?
Fetched Options: [super.myMethod();, super();, super(); myMethod();, super.myMethod;]
What is the main characteristic of OOP?
1. Encapsulation
2. Polymorphism
3. Abstraction
4. All of the above
Enter your option : 2
Incorrect. The correct answer is: All of the above

Which keyword is used to declare a class that cannot be inherited?
1. abstract
2. final
3. static
4. private
Enter your option : 3
Incorrect. The correct answer is: final

What is the purpose of a constructor in Java?
1. To initialize an object
2. To destroy an object
3. To provide a method for polymorphism
4. To allow inheritance
Enter your option : 4
Incorrect. The correct answer is: To initialize an object

Which of the following is true about inheritance in Java?
1. A subclass can inherit from multiple superclasses.
2. A subclass can only inherit from one superclass.
3. A subclass cannot inherit methods from a superclass.
4. A subclass can inherit private members of a superclass.
Enter your option : 5
Invalid option. Please choose a number between 1 and 4

What is polymorphism in Java?
```


eclipse-workspace - quiz-app/src/main/java/QuizApp.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Console Coverage

```
QuizApp [Java Application] C:\Program Files\Java\jdk-23\bin\javaw.exe (20 Nov 2024, 11:02:47 pm) [pid: 26336]
Enter your option : 1
Incorrect. The correct answer is: The ability to redefine a method in a subclass with the same signature

What is the correct way to instantiate an object in Java?
1. ClassName obj = new ClassName();
2. obj = new ClassName();
3. new ClassName() obj;
4. ClassName obj;
Enter your option : 3
Incorrect. The correct answer is: ClassName obj = new ClassName();

Which of the following is true about method overloading?
1. It allows you to define a method with the same name and the same parameters in a class.
2. It allows you to define a method with the same name but different parameters.
3. It allows a subclass to override a method in the parent class.
4. It is not allowed in Java.
Enter your option : 2
Correct!

What will be the output of the following code?
1. Display in Example class
2. Display in Demo class
3. Compilation error
4. Runtime error
Enter your option : 3
Incorrect. The correct answer is: Display in Demo class

Which of the following is an example of multiple inheritance in Java?
1. A class extending two classes directly
2. A class implementing multiple interfaces
3. A class implementing one interface and extending another class
4. None of the above
Enter your option : 4
Incorrect. The correct answer is: A class implementing multiple interfaces

What does the instanceof operator do in Java?
1. It checks whether a reference is an instance of a specific class.
2. It checks whether a reference is of the type Object.
3. It checks whether a method belongs to a class.
4. It checks whether a variable is an instance of a superclass.
Enter your option :
```

9. Conclusion

1. The online course enrollment system is an essential tool for facilitating knowledge assessment and interactive learning. By integrating dynamic quiz functionalities, user score tracking, and modular design, the system ensures a seamless and engaging user experience. It simplifies quiz management for administrators while providing learners with an intuitive platform to test and improve their skills. With its scalable and extensible architecture, the application is well-suited for future enhancements such as user authentication, leaderboards, and online deployment. By fostering real-time interaction and automated feedback, the Java Quiz Application supports diverse learning needs, ensuring adaptability for educational and corporate training purposes. Ultimately, this system empowers users with an efficient and user-friendly platform for knowledge evaluation, fostering growth and skill development.

---

10. References

1. Gupta, R., & Mehta, A. (2021). *"Developing Interactive Quiz Applications Using Java."* International Journal of Computer Applications, 183(4), 56-63.
2. Sharma, P., & Verma, S. (2020). *"Enhancing Learning Outcomes with Technology-Driven Assessment Systems."* Journal of Educational Technology, 18(2), 34-49.
3. Patel, K., & Joshi, R. (2019). *"Scalable Architectures for Knowledge Assessment Tools."* International Journal of Software Engineering and Knowledge Engineering, 29(5), 1123-1136.

4. Rani, M., & Singh, D. (2020). *"Implementing Modular Design in Java-Based Applications."* Journal of Computer Science and Engineering, 25(3), 78-89.