Tribhuvan University

Faculty of Humanities and Social Science

A Project Report On

"Quiz Management System with Fisher-Yates Shuffle Algorithm"

In partial fulfillment of requirements in Bachelor in Computer Applications

Submitted by:

Lusana Shakya (6-2-410-128-2021)

June, 2025 A.D.

Under the Supervision of

Mr. Anand K.C.

Tribhuvan University

Faculty of Humanities and Social Science

A Project Report On

"Quiz Management System with Fisher-Yates Shuffle Algorithm"

In partial fulfillment of requirements in Bachelor in Computer Applications

Submitted by:

Lusana Shakya (6-2-410-128-2021)

June, 2025 A.D.

Under the Supervision of

Mr. Anand K.C.

Tribhuvan University

Faculty of Humanities and Social Science

Prime College

SUPERVISORS MANAGEMENT

It is my pleasure to recommend that a project report on "Quiz Management System" has been prepared under my supervision by Lusana Shakya in partial fulfillment of the requirement of the degree of Bachelor of Computer Applications. Her report is satisfactory and is an original work done by her to process for the future evaluation.

Mr. Anand K.C.

SUPERVISOR

Lecturer, Department of IT

Prime College

Tribhuvan University

Faculty of Humanities and Social Science

Prime College

LETTER OF APPROVAL

This is to certify that this project report, prepared by Lusana Shakya on "Quiz Management System with Fisher-Yates Shuffle Algorithm" in partial fulfillment of the requirements for the degree of Bachelor in Computer Application, has been evaluated.

In our opinion, it is satisfactory in the scope and quality as a project for the required degree.

Mr. Anand KC **Supervisor** **Prime College** **Khusibu, Nayabajar, Kathmandu**	**Ms. Rolisha Sthapit** **Program Co-Ordinator/ Internal Examiner** **BCA Department** **Prime College** **Khusibu, Nayabajar, Kathmandu**
	Mr. **External Examiner**

ACKNOWLEDGEMENT

This project on building a **Quiz Management System**, which gives personalized quizzes using the **Fisher-Yates Shuffle** algorithm, would not have been possible without the help and support of many people. I am very thankful to **Mr. Anand K.C**., my Project Supervisor and Coordinator, for his continuous guidance and support throughout the project. His knowledge and encouragement helped me move in the right direction and complete the system successfully. I also want to thank my friends, teachers, mentors, and classmates who supported me, gave useful feedback, and helped improve the project. Their ideas made the system better and more useful for real users. I truly appreciate everyone's support and am thankful for their role in making this project a success.

-Lusana Shakya

ABSTRACT

These days, many people enjoy playing quizzes tailored to their interests, such as selecting the number of questions and preferred categories. To enhance this experience, this project introduces a Quiz Management System that allows users to play quizzes dynamically and track their results through a personalized dashboard. The system uses the Fisher-Yates Shuffle algorithm to randomize the order of quiz questions, ensuring a fair and unpredictable quiz experience every time. Users can customize their quiz by choosing the number of questions they want to attempt and selecting specific categories that interest them. The system then presents the randomized questions accordingly, and upon completion, users can immediately view their scores and detailed results in the dashboard. Developed using Node.js for the backend and ReactJS for the frontend, the system provides a seamless and interactive interface accessible on both web and mobile platforms. This allows quiz enthusiasts to engage with quizzes that fit their preferences while enabling administrators to manage quiz content efficiently. The system has demonstrated effectiveness in delivering engaging quizzes with accurate result tracking, and future improvements will include features such as timed quizzes, user leaderboards, and advanced analytics for deeper insights. Overall, the Quiz Management System offers an enjoyable, flexible, and user-centered platform for quiz playing and performance monitoring.

Keywords: Fisher-Yates Shuffle

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

JSON	JavaScript Object Notation
Node.js	JavaScript runtime for server-side apps
CSV	Comma-Separated Values
ML	Machine Learning
API	Application Programming Interface
SVM	Support Vector Machine
React JS	Framework for building web apps

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1. Introduction

In today's fast-growing digital learning and entertainment landscape, quizzes have become a popular tool for both education and engagement. As more users look for interactive ways to test their knowledge or have fun, providing personalized and dynamic quiz experiences is essential. The **Quiz Management System** is designed to address this need by offering a customizable and intelligent platform where users can select their quiz preferences, play randomized quizzes, and instantly view their results through a user-friendly dashboard.

The system allows users to choose their desired number of questions and select specific categories, making the experience tailored to individual interests. To ensure each quiz session is unique and fair, the platform utilizes the **Fisher-Yates Shuffle algorithm**, which efficiently randomizes the order of questions before presenting them to the user. This prevents repetition and ensures that no two quiz sessions are exactly the same.

Built using **Node.js** for the backend and **ReactJS** for the frontend, the application offers a smooth and responsive interface suitable for both web users. It simplifies quiz management for administrators by allowing easy categorization, and performance tracking, while offering users feedback on their quiz results through the dashboard.

The demand for personalized quiz platforms is increasing as users seek meaningful and engaging experiences. This Quiz Management System not only reduces manual effort in selecting and organizing quizzes but also supports better user engagement through customization and real-time scoring. It serves as an effective solution for educational institutions, training platforms, or general entertainment, and sets the foundation for future features like time-based quizzes, performance analytics, and leaderboards.

In essence, this system streamlines the process of quiz creation and participation, ensuring an enjoyable, efficient, and personalized quiz experience for all users.

1.2. Problem Statement

The Quiz Management System tries to solve the following problems:

- Most systems offer limited personalization, failing to cater to different user preferences like topic or question count.
- Fairness and variety in question delivery is often lacking, especially when users attempt the same quiz multiple times.
- Manual quiz creation and curation is time-consuming and inefficient, especially for large sets of quizzes and users.

1.3. Objectives

The following are the objectives of the project:

- To build a Quiz Management System that suggests quizzes based on what the user likes.
- To use the Fisher-Yates Shuffle algorithm to compare different quizzes and recommend the ones that are most similar to the user's interests.
- To provide users with instant results and a dashboard that displays performance data such as scores and attempted questions.

1.4. Scope and Limitations

1.4.1. Scope

The following are the scopes of the project:

- The system is designed to recommend suitable quizzes to users based on their interests.
- It uses the Fisher-Yates Shuffle algorithm to compare quiz features and user preferences.
- The system will be available as a simple and user-friendly application to help quizers easily find matching quiz options.

1.4.2. Limitations

The following are the limitations of the project:

- The system is limited to presenting only multiple-choice questions (MCQs).
- It does not currently support features like timed quizzes, negative marking, or user leaderboards.

- Quiz suggestions are based on pre-entered question banks; newly added quizzes may not appear unless the data is updated.

1.5. Development Methodology

The **Prototype Model** is a software development method that emphasizes creating a simplified version of the system early in the process. This early model or **prototype** allows users and stakeholders to see how the system might work, provide feedback, and refine it before full-scale development begins. It's especially helpful when requirements are unclear at the start. In the context of the **Quiz Management System**, this model ensures that the system aligns with user expectations through continuous iterations.

Requirement Gathering: In the first phase of the Prototype Model, initial requirements for the Quiz Management System are gathered. At this point, the focus is on understanding the basic expectations of users, such as the need for a login system, the ability to select quiz categories, answer questions, and receive scores at the end. Admin-side requirements may include managing users, quiz categories, and questions.

Quick Design: Once the basic requirements are identified, a quick design of the system is created. This design doesn't go into deep technical details but instead focuses on how the system will appear and function from the user's perspective. For the Quiz Management System, this may include wireframes or simple screens for login, dashboard, quiz interface, and an admin panel. It also outlines the basic navigation between pages and how users will interact with the system.

Build Prototype: Using the quick design, a working prototype is built with limited functionalities. This early version of the Quiz Management System may allow users to select a quiz category, answer a few sample questions, and view a mock score at the end. It also includes a basic admin interface to add or view quiz questions. While the prototype may not be connected to a database or include features like a timer or authentication at this point, it gives a feel of the system's core idea.

User Evaluation: After building the prototype, it is presented to end-users and stakeholders for feedback. In the case of the Quiz Management System, students, teachers, or supervisors interact with the prototype and provide their opinions on its usability and functionality. For example, they might suggest adding features like time limits for quizzes, feedback on wrong answers, mobile responsiveness, or a progress bar.

Refining Prototype: Based on the feedback collected, the prototype is revised and improved. This stage may go through multiple cycles of feedback and enhancement to meet

user expectations. For the Quiz Management System, this might involve implementing navigation between questions, real-time score calculation, better UI/UX design, integration with a sample database, and refining user roles (like admin, student).

Implement and Maintain: After the system is deployed, it enters the maintenance phase. During this phase, the system is monitored for bugs, performance issues, and changing user needs. Updates may include adding new quiz categories, fixing glitches in score calculations, enhancing security measures, or including new features like leaderboards, certificates, or analytics. Regular maintenance ensures that the Quiz Management System stays relevant, secure, and efficient over time as user expectations and technologies evolve.

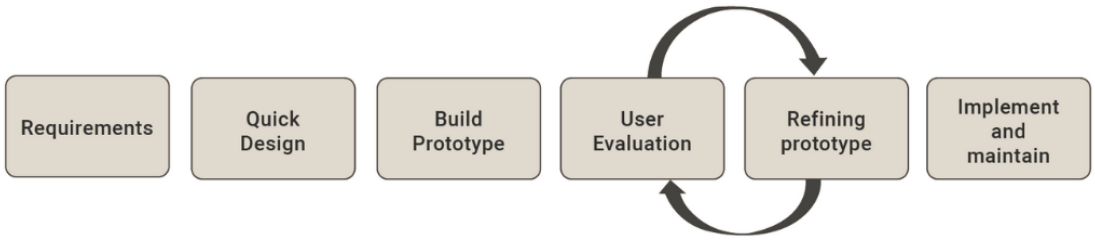


Figure 1.1: Prototype Model [9]

1.6. Report Organization

The report on the Quiz Management System is organized into five chapters, each addressing a crucial aspect of the project. This structured approach provides a comprehensive overview of the system's development, implementation, and evaluation.

Here's a summary of each chapter:

The first chapter introduces the Quiz Management System, offering a detailed overview of its purpose, scope, and importance. It answers key questions such as what the system aims to achieve, how it works, and why it is useful in the context of quizzes. The chapter sets the foundation for understanding the project's goals and its impact on improving the user quiz experience through smart management.

Likewise, in the second chapter, the report reviews related work in the area of quiz Management systems and the use of machine learning. It presents a study of existing systems and research, highlighting commonly used methods, algorithms, and their outcomes. This literature review helps to identify gaps and explains how the proposed system contributes new value to the field of quiz management.

Chapter 3 provides a detailed analysis and design of the Quiz Management System. It includes system requirements, feasibility study, and various design diagrams such as use case, class, sequence, activity, and state diagrams. This chapter also explains the use of the Fisher-Yates Shuffle algorithm for providing personalized quiz suggestions. Additionally, it evaluates technical and operational feasibility and outlines the system architecture.

The fourth chapter focuses on the development and testing of the Quiz Management System. It describes the tools and technologies used, such as ReactJS for frontend, Node.js for backend, and MongoDB for database. The chapter explains how the system processes user inputs and recommends quizzes using the algorithm. It also includes code snippets, data visualization, and test cases to validate the performance and accuracy of the system.

Chapter 5 summarizes the overall findings of the project. It evaluates the effectiveness of the Management system and the role of the selected algorithm in improving quiz suggestions. The chapter provides insights into the system's success and offers suggestions for future improvements, such as using more advanced algorithms, expanding the dataset, and adding features like live tracking and real-time management.

This report structure ensures a complete presentation of the Quiz Management System, from concept and research to design, implementation, and future development. Each chapter is designed to provide detailed, clear information to help readers understand the full scope and results of the project.

CHAPTER 2

BACKGROUND STUDY AND LITERATURE REVIEW

2.1. Background Study

The increasing integration of digital platforms in education has led to the development of intelligent systems for managing and evaluating student performance. Among these, **Quiz Management Systems (QMS)** have emerged as effective tools for conducting assessments, providing immediate feedback, and monitoring learner progress. These systems allow educators to create, store, and manage quizzes while enabling students to access and attempt them conveniently through web or mobile interfaces.

One of the critical aspects of an efficient QMS is **randomization**—the ability to present questions in a unique order for each attempt or user. This feature is essential for minimizing cheating, maintaining fairness, and ensuring that assessments reflect individual understanding. Traditional static quiz formats can lead to repetitive patterns and memorization, which may compromise the integrity of evaluations. Therefore, implementing a robust algorithm for randomizing questions and answer choices is crucial.

The **Fisher-Yates Shuffle Algorithm**, also known as the **Knuth Shuffle**, provides an optimal solution for this requirement. It is a well-known algorithm for generating a random permutation of a finite sequence—in this case, quiz questions or options. The algorithm operates in **linear time complexity (O(n))**, ensuring efficiency even for large datasets. Its unbiased nature ensures that every possible ordering of elements is equally likely, making it ideal for educational applications where fairness is critical.

The integration of the Fisher-Yates Shuffle algorithm within the QMS architecture ensures a dynamic and fair assessment environment. This algorithm works well across various programming platforms and can be easily implemented on both the client-side (e.g., JavaScript) and server-side (e.g., Python, PHP, Node.js).

Moreover, combining the Fisher-Yates Shuffle with backend logic and user session tracking allows the system to maintain quiz integrity while also recording and analyzing student performance metrics. These features collectively support personalized learning paths and adaptive assessments, aligning with modern educational goals.

In conclusion, the Fisher-Yates Shuffle algorithm plays a vital role in enhancing the functionality and fairness of a Quiz Management System. Its ability to efficiently and

uniformly randomize quiz components makes it a preferred choice in educational software development. By ensuring variability and reducing predictability, it helps maintain academic integrity and supports a more accurate evaluation of student understanding.

2.2. Literature Review

This study used the android media application with the Multimedia Development Life Cycle (MDLC) research method and the Fisher-Yates Shuffle algorithm to perform arandom process of problem training. This study only has once randomization and this method of software testing that examines the functionality of an application based on the specifications (Black Box Testing). [1]

The objective of this paper to examine about AEP improvement and to develop AEP in UiTM Perlis Branch. Subsequently, Ad-Hoc Question Paper Application (AQPA) has been created utilizing Fisher-Yates calculation to produce inquiries for test paper in the college. This framework just can randomize onetime and master test by 20 Lecturer. [2]

The FYSA applied to choose shuffle questions r from the questions bank in the database. Text Mining Algorithm aids in duplicity removal from the paper. The generated question paper will be in Word Format. In this study, one class sample when end semester examination. [3]

It is proved that the game is well performed and the pieces of pictures arrangement can be randomized in accordance with the method. -Onetime randomization -Black box test with small sample size test. [4]

This application can reduce cheating in the process of admission examination new students and help activities the process of admitting new students. Randomization process only once and Black Box Testingonly. [5]

The challenges and obstacles linked to the manual creation of question papers by educators, proposing a remedy in the form of an Automatic Question Paper Generating System. Developed with the ASP.Net programming language, this system generated question papers using an existing question repository in the database. The process involved the utilization of fuzzy logic and the apriori algorithm. The research outcome yielded a functional web application that underwent testing and demonstrated successful operation. [6]

Automatic Question Generation System comprising various modules such as user administration, subject selection, difficulty level specification, question entry, question management, paper generation, and paper management. Through a meticulous design process, the system adeptly scrutinized and composed examination papers with a high success rate. Their approach employed a shuffling algorithm for randomization purposes. Within this algorithm, users specified the subject, question type, and difficulty level, and the system automatically generated the examination paper accordingly. The editing of questions was facilitated through a Word processor, with the final paper being storable in ".doc" file format. Noteworthy attributes of the system included user-friendly operation, a well-designed interface, optimal usability, robust security, and high stability, along with reliability. [7]

The Automatic Question Generator designed for crafting Basque language test questions. This generator relies on linguistically analyzed real corpora, represented in Extensible Markup Language (XML), as its information source. ArikIturri employs Natural Language Processing (NLP) tools and operates with a data bank containing morphologically and syntactically analyzed sentences, identifying phrase chunks. Input for ArikIturri is given in XML format, while the generated outputs are instances of questions defined using XML markup language. The system automatically generates question instances, utilizing both NLP tools and specialized linguistic information for question formulation. [8]

CHAPTER 3

SYSTEM ANALYSIS AND DESIGN

3.1. System Analysis

3.1.1. Requirement Analysis

3.1.1.1. Functional Requirements

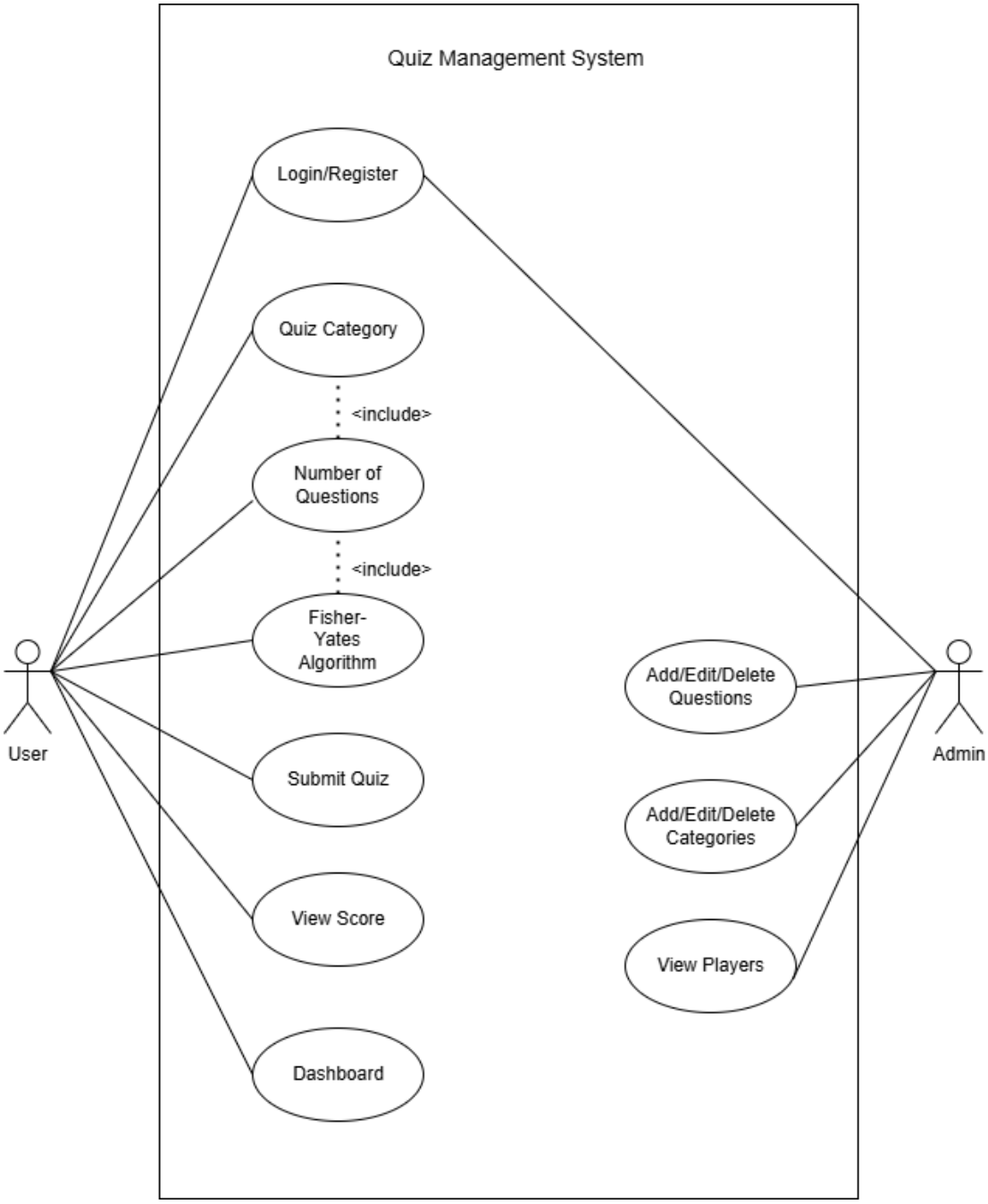


Figure 3.1: Use Case diagram for Quiz Management System

1. Login/Register

This use case allows users to create an account or log into the system. Registration captures essential details to identify users, and login ensures that only authorized users can access quiz functionalities. It also sets up the foundation for personal tracking of performance and secure access.

2. Quiz Category

Once logged in, users can select a quiz category of their interest. This helps them take quizzes in specific domains such as Science, History, Technology, or General Knowledge. Category selection filters the question bank accordingly, making the quiz relevant and personalized.

3. Number of Questions

After selecting a category, users are asked to choose how many questions they want in the quiz. This adds flexibility, allowing users to engage in short quizzes (like 5 questions) or longer ones, depending on their preference or time availability.

4. Fisher-Yates Algorithm

This represents the internal process that shuffles the questions randomly. The Fisher-Yates algorithm ensures that no two quiz attempts are exactly the same, even if a user selects the same category and number of questions again. It adds fairness, unpredictability, and variety to the user experience.

5. Submit Quiz

Once the user finishes answering all the questions, they can submit the quiz. This use case sends the user's responses to the system, which then evaluates the answers and prepares the final score. It acts as the transition between answering and reviewing performance.

6. View Score

After submission, the user can instantly view their total score. This module also provides a breakdown of correct and incorrect answers, helping users assess their strengths and weaknesses in the chosen category.

7. Dashboard

The dashboard presents a summary of the user's quiz activity. It may display stats such as total quizzes taken, average score, best performance, and history of past attempts. The dashboard helps users track their learning progress over time.

Admin-Side Use Cases

8. Login

The admin uses this functionality to securely access the backend of the system. Only verified admins are allowed to manage the question bank, categories, and view user data. It protects sensitive system features from unauthorized access.

9. Add/Edit/Delete Questions

This core administrative feature allows the admin to maintain the quiz database. They can add new questions, modify existing ones if there are errors, or delete outdated or irrelevant questions. It ensures the question set stays updated and accurate.

10. Add/Edit/Delete Categories

Admins can also manage the classification of questions through categories. They can create new categories to introduce new quiz topics, rename existing ones, or remove unused categories. This helps in organizing the quiz content efficiently.

11. View Players

This use case enables the admin to monitor quiz participants. It may include viewing usernames, the number of attempts, scores, and user activity logs. This information can be used for analytics, identifying active users, or improving the quiz experience.

3.1.1.2. Non-Functional Requirements

1. User Friendly:

The Quiz Management System must provide a smooth and intuitive user experience. Both users and admins should be able to navigate the system without confusion. Clear layouts, straightforward buttons, and easy-to-read content will enhance overall usability.

2. Simple and Easy to Use:

The system should be simple enough for users of all technical backgrounds to operate

without any training. The quiz-taking process, category selection, and question submission should all follow a clear and logical flow.

3. Easy Access:

The system should be accessible from any standard web browser without requiring complex installations or configurations. Users should be able to access the system from desktops, laptops, tablets, or smartphones with a stable internet connection.

4. Responsive Design:

The system must be fully responsive, adapting to various screen sizes and resolutions. Whether accessed from a mobile device or a desktop, the layout and functionality should remain consistent and effective.

5. Performance and Speed:

The system should load quickly and perform smoothly during all operations—whether loading questions, submitting answers, or displaying scores. Efficient use of resources and optimized queries are essential for maintaining good performance.

3.1.2. Feasibility Analysis

3.1.2.1. Technical Feasibility

The Quiz Management System is technically feasible due to its simple architecture, efficient algorithms, and use of widely adopted web technologies. It integrates the Fisher-Yates Shuffle algorithm, which is a lightweight and optimal method for randomizing quiz questions, ensuring unique and fair quiz attempts. The backend is developed using Node.js, with MongoDB as the database to store questions, categories, and quiz results. These technologies are well-suited for real-time data handling and can efficiently manage dynamic quiz content. On the frontend, ReactJS is used to build an interactive, responsive, and user-friendly web interface. The system operates through RESTful APIs, enabling smooth communication between the frontend and backend. The use of modular, open-source tools ensures the system performs effectively on standard machines or cloud servers, making it scalable and adaptable for increasing numbers of users and quiz data.

3.1.2.2. Operational Feasibility

The Quiz Management System ensures operational feasibility through its intuitive design and simple navigation flow, making it easy for users of all backgrounds to take quizzes and view results. Admins can seamlessly manage quiz content, including adding, editing, or deleting questions and categories. The system supports instant quiz submission and result

generation, ensuring minimal delays in the user experience. Built using JavaScript-based technologies (Node.js and ReactJS), the system is modular and easy to maintain. Additionally, the system can be expanded to include advanced features like user authentication, leaderboard tracking without requiring a major overhaul.

3.1.2.3. Economic Feasibility

The Quiz Management System is cost-effective to build and maintain due to its reliance on **open-source technologies** like Node.js, Express.js, MongoDB, and ReactJS, eliminating licensing costs. These tools require only basic hardware and can be hosted on budget-friendly cloud platforms such as Heroku, Vercel, or MongoDB Atlas. For academic or small-to-medium scale deployments, the infrastructure cost remains minimal. Moreover, the lightweight Fisher-Yates algorithm does not require computationally intensive processing, reducing the need for high-end servers. In terms of return on investment (ROI), the system adds significant value by enhancing learning experiences, supporting self-assessment, and providing a platform that can be monetized or integrated into larger educational services or training tools.

3.1.2.4. Schedule Feasibility

- **Brainstorming**: The initial brainstorming phase is scheduled from March 18 to March 24 (7 days). During this period, core system objectives, user stories, and requirements were identified. The timeline is suitable for team discussions and requirement gathering.

- **Design**: The design phase runs from March 25 to April 10 (17 days). This includes preparing UI/UX wireframes, database schema designs, and outlining system flowcharts. The time allocated is appropriate for laying a strong foundation before development.

- **Development**: Development is planned from April 11 to May 5 (25 days). This phase involves building frontend components, setting up backend routes and database integration, and implementing the Fisher-Yates Shuffle logic. The duration is realistic for a focused scope with agile iteration.

- **Testing**: Testing is scheduled from May 6 to May 9 (4 days). This period focuses on unit testing, usability testing, bug fixing, and performance checks. With prior integration of testing during development, the schedule is feasible.

- **Deployment:** Deployment will occur between May 10 to May 13 (4 days). It involves deploying the system to a live environment, ensuring proper configuration, and monitoring real-time behavior. This schedule supports a smooth and controlled release.

- **Documentation**: Documentation will be created from May 14 to May 25 (12 days). This includes technical documentation, user manuals, admin guides, and final project reporting. The time is sufficient to produce quality, maintainable documentation.

Table 3.1: Schedule Feasibility

Task	Start Date	End Date	Days to Complete
Brainstorm	18 March	24 March	7
Design	25 March	10 April	17
Development	11 April	5 May	25
Testing	6 May	9 May	4
Deployment	10 May	13 May	4
Documentation	14 May	25 May	12

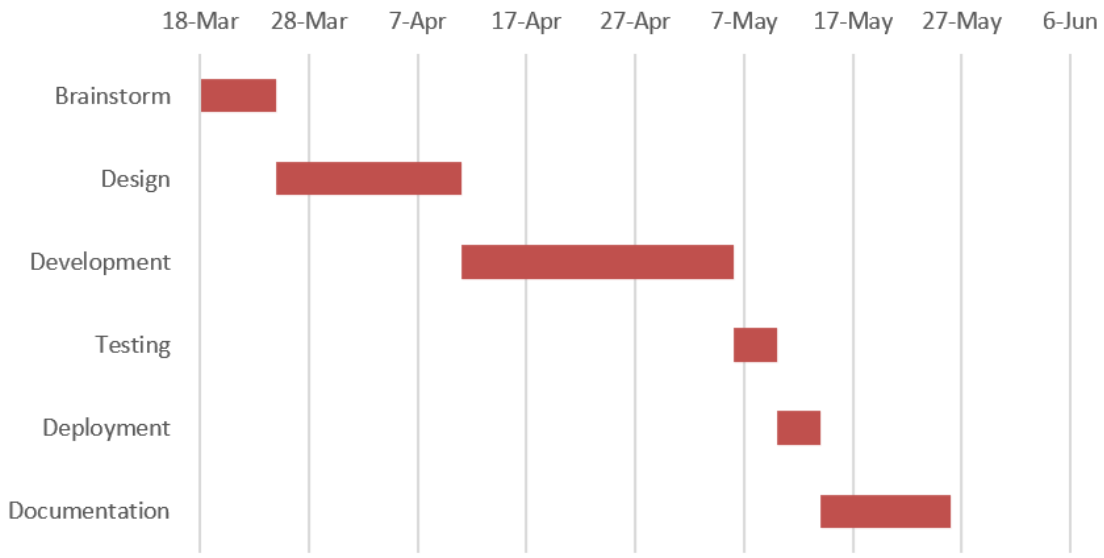


Figure 3.2: Gantt Chart for Quiz Management System

3.1.3. Object Modeling using Class Diagram

The class diagram illustrates a well-structured Quiz Management System composed of several core components, each with a distinct responsibility. At the center is the QuizResult

class, which records the outcome of quiz attempts, including scores, selected answers, and accuracy. Supporting this are the User, Question, and Category classes, where User manages player information and authentication, Question stores quiz content along with correct answers, and Category groups questions by topic. The system captures detailed data for each quiz attempt, linking users to their performance and responses. This modular structure ensures efficient quiz tracking, performance analysis, and organized content management, making the system scalable, maintainable, and easy to use.

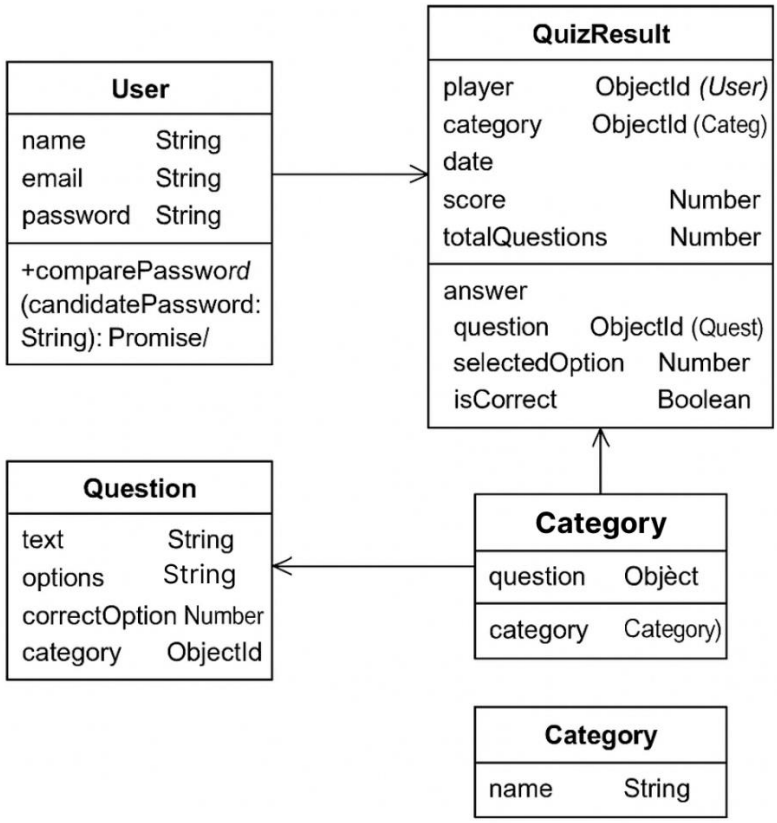


Figure 3.3: Class Diagram for Quiz Management S System

Figure 3.3 represents a Quiz Application System with four main entities: User, Question, Category, and QuizResult. Users have basic authentication details and can take multiple quizzes, which are recorded in the QuizResult class along with score, date, and selected answers. Each quiz result links to a specific category and stores individual question responses, including whether the selected option was correct. Questions belong to

categories and include the question text, multiple options, and the correct answer. The design ensures organized quiz management, performance tracking, and category-based question grouping.

3.1.4. Dynamic Modelling using State and Sequence Diagrams

The state diagram of the Quiz Management System illustrates a dynamic workflow that begins with user login or registration, followed by validation of credentials. Based on the user's role, the system branches into different states: regular users proceed to take quizzes, answer questions, and submit responses to receive their results, while admins transition to quiz management functionalities, including creating, editing, and publishing quizzes. Admins can also monitor user performance and analyze quiz outcomes. This diagram effectively demonstrates how the system adapts dynamically to user roles and actions, ensuring smooth handling of both quiz participation and administrative tasks.

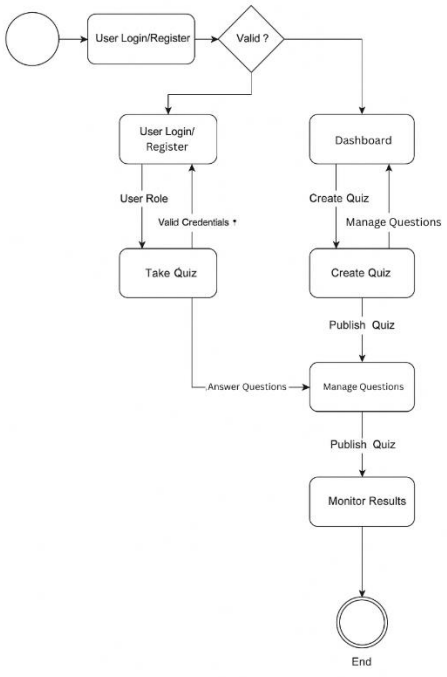


Figure 3.4: State Diagram for Quiz Management System

In figure 3.4, In the state diagram of the Quiz Management System, the process begins with the user accessing the system through a login or registration form, transitioning to a validation state where their credentials are checked. Upon successful validation, the system diverges based on user roles—regular users move to the quiz-taking flow, where they select a quiz, answer questions, and submit responses. This leads to the final state where results and scores are displayed. On the other hand, admins enter a different flow, progressing

through states such as creating quizzes, managing questions, and publishing them. The process concludes with monitoring quiz results and user performance. The diagram clearly captures how the system handles role-based transitions, input validation, and dynamic quiz lifecycle management from both the user and admin perspectives.

The sequence diagram represents the interaction flow between an application and a server within a diabetes prediction system. It visually outlines the series of steps and exchanges that occur as the system processes user input and generates predictions. This type of diagram is useful for understanding how different components of the system communicate and the sequence in which actions take place.

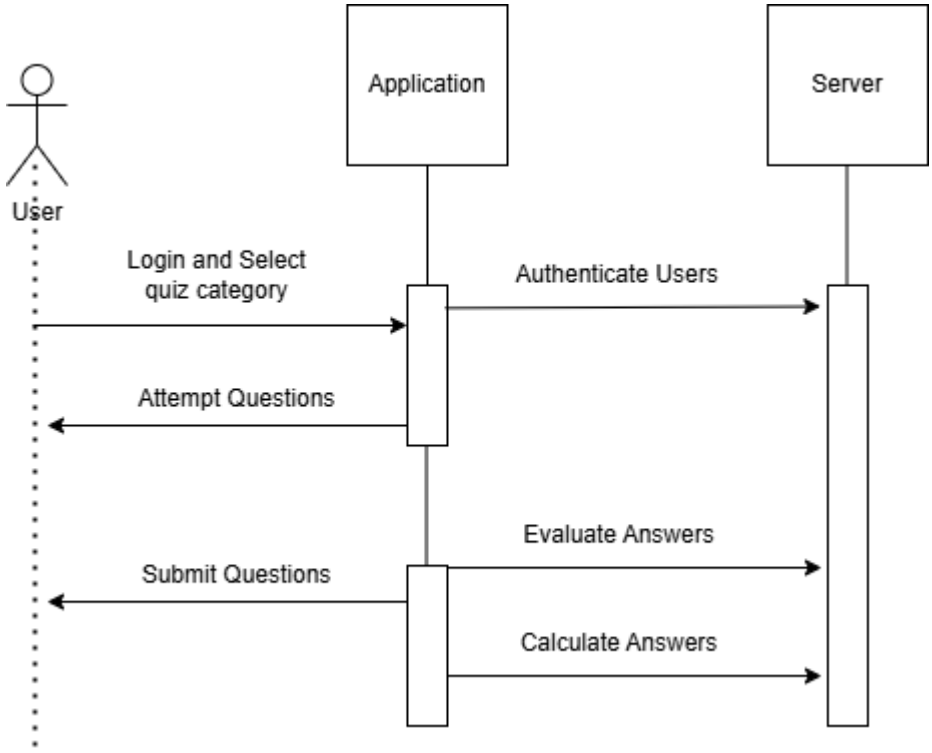


Figure 3.5: Sequence Diagram for Quiz Management System

In Figure 3.6, the process begins when the user logs into the application and selects a quiz category. The application captures this request and forwards it to the server, which authenticates the user and fetches questions related to the selected category. The server sends back a set of questions, which are then displayed to the user through the application interface.

As the user attempts the quiz, their answers are submitted question by question or all at once, depending on the design. The application transmits the responses to the server, where

the server evaluates the answers, calculates the score, and stores the result in the database. Once the evaluation is complete, the server sends the result (such as score and correct answers) back to the application, which then displays the final result to the user, completing the quiz process.

3.1.5. Process Modelling using Activity Diagrams

The activity diagram represents the workflow of a diabetes prediction system. The process starts with the user login form and the authentication of user details.

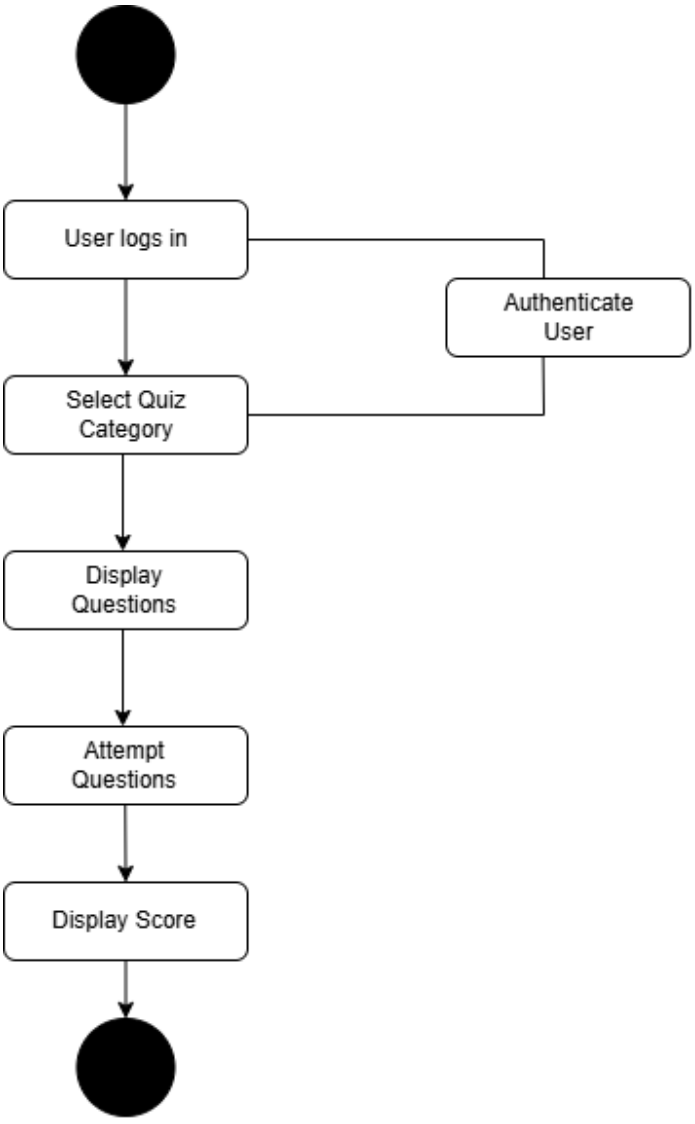


Figure 3.6: Activity Diagram for Quiz Management System

Figure 3.6 illustrates the structured flow of a quiz-based application, starting from user authentication to score display. Initially, the user logs into the system, which then authenticates the credentials to ensure secure access. Upon successful authentication, the

user proceeds to select a quiz category of interest. The system responds by retrieving and displaying the relevant set of questions for the chosen category. The user then attempts the displayed questions, after which the system evaluates the responses and calculates the final score. This score is subsequently displayed to the user. The diagram clearly outlines the sequential interactions between the user and the system, emphasizing key stages such as authentication, content delivery, user engagement, and result presentation.

3.3. Algorithm Details

3.3.1. Fisher-Yates Shuffle Algorithm

The **Fisher-Yates Shuffle** is a well-known algorithm used to generate a random permutation of elements in a list, ensuring that every possible arrangement is equally likely. In a **quiz management system**, this algorithm is essential for introducing fairness and unpredictability in the order of quiz questions. When a user selects a quiz category, the system retrieves a set of related questions. Before displaying them, the Fisher-Yates Shuffle is applied to randomize their order.

The algorithm works by traversing the list from the last index down to the first. For each index iii, it generates a random index and then swaps the elements at positions. This in-place swapping ensures that no additional memory is required and that the operation runs in linear time, making it highly efficient for real-time applications.

By shuffling questions differently for each quiz attempt, the system reduces the chances of users memorizing question sequences or collaborating based on order. This increases the **integrity, fairness, and variability** of the quiz-taking experience. Its simplicity, efficiency, and mathematical soundness make it a perfect choice for educational platforms that require random, yet fair, presentation of content.

This mathematical formula for the Fisher-Yates Shuffle Algorithm is:

$$\textit{For an array } A=[a_0, a_1, \ldots\ldots, a_{n-1}], \textit{ for each i from n-1 down to 1:} \ldots\ldots\ldots\ldots \text{(i)}$$

$$j = \textit{random integer such that } 0 \leq j \leq I \ \ldots\ldots\ldots\ldots\ldots \textit{ (equation 1)}$$

$$\textit{Swap } a_i \leftrightarrow a_j$$

Where:

n: Length of the array.

i: Current index in the reverse iteration (starting from n−1 down to 1).

j: Random index selected uniformly such that $0 \leq j \leq i$.

$a_i \leftrightarrow a_j$: *Swap the values at positions i and j.*

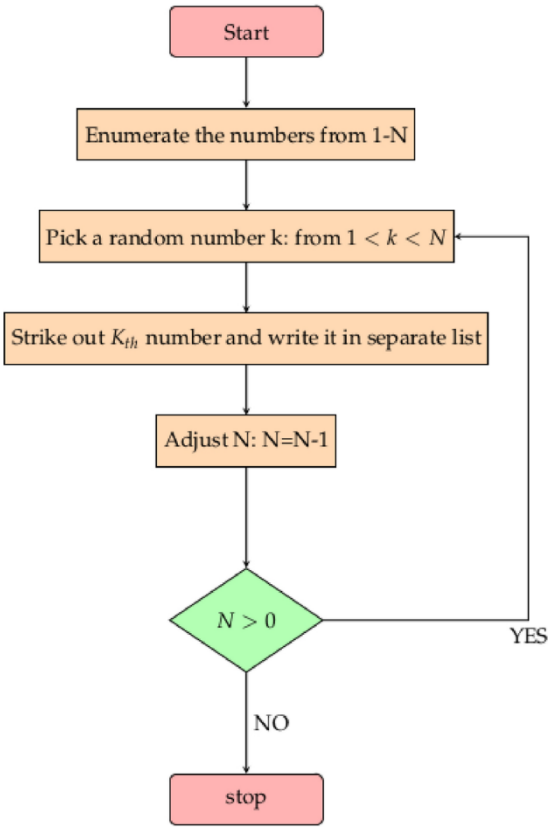


Figure 3.18: Fisher-Yates Shuffle Algorithm [8]

CHAPTER 4

IMPLEMENTATION AND TESTING

4.1. Implementation

4.1.1. Tools Used

MongoDB: MongoDB is a NoSQL, document-oriented database that stores data in flexible, JSON-like documents. It allows developers to handle large volumes of unstructured or semi-structured data with ease. In the MERN stack, MongoDB serves as the primary database, providing scalability and high performance for storing application data. Its schema-less nature offers flexibility in data modeling, making it suitable for modern web applications where data requirements may evolve over time.

Express.js: Express.js is a lightweight and flexible web application framework for Node.js that simplifies the process of building server-side logic. It provides a robust set of features for handling HTTP requests, routing, middleware, and APIs. In the MERN architecture, Express acts as the backend framework that connects the frontend (React) to the database (MongoDB), enabling efficient request handling and response delivery.

React.js: React.js is a popular JavaScript library developed by Facebook for building dynamic and responsive user interfaces. It allows developers to create reusable UI components and manage application state efficiently through its virtual DOM and declarative syntax. In the MERN stack, React handles the frontend, providing an interactive and fast user experience.

TailwindCSS: Tailwind CSS is a utility-first CSS framework used to design custom user interfaces quickly and efficiently. In a React Native context, Tailwind-like utility libraries such as nativewind are often used to style components using predefined utility classes, allowing rapid UI development with consistency.

Node.js: Node.js is a server-side JavaScript runtime built on Chrome's V8 engine. It enables developers to run JavaScript on the backend, allowing for a unified development language across the entire application stack. In the MERN stack, Node.js acts as the runtime environment for Express, handling asynchronous operations and executing backend logic. Its event-driven, non-blocking I/O model makes it highly efficient and capable of handling

concurrent connections, which is crucial for modern web applications that require scalability and performance.

4.1.2. Implementation Details of Modules

The following are the modules used for the implementation of Quiz Management System.

4.1.2.1. Fisher-Yates Shuffle Algorithm:

To ensure fairness and uniqueness in each quiz attempt, the system uses the **Fisher-Yates Shuffle Algorithm** to randomize the order of questions before they are presented to the user. This prevents users from memorizing question sequences and reduces cheating. The algorithm runs in linear time and shuffles questions in place just before the quiz starts. This feature significantly enhances quiz integrity and provides a different experience in every attempt.

```
//! Fisher-Yates shuffle algorithm
function shuffleArray(array) {
  const shuffled = [...array]
  for (let i = shuffled.length - 1; i > 0; i--) {
    const j = Math.floor(Math.random() * (i + 1))
    ;[shuffled[i], shuffled[j]] = [shuffled[j], shuffled[i]]
  }
  return shuffled
}

//! Get random questions from a category
function getRandomQuestions(questions, count) {
  //! Shuffle the questions array
  const shuffled = shuffleArray(questions)

  //! Return the requested number of questions
  return shuffled.slice(0, count)
}

module.exports = {
  shuffleArray,
  getRandomQuestions,
  getRandomQuestionsWithDifficulty,
}
```

Figure 4.1: Code Snippet of Fisher-Yates Shuffle Algorithm

4.1.2.2. Registration and Login Module:

Support Vector Machine (SVM) is a supervised learning algorithm used for classification and regression tasks. It works by mapping input features into a high-dimensional space using a kernel function, where it finds the optimal hyperplane that maximizes the margin between different classes. The key idea is to identify the hyperplane that best separates the classes while minimizing classification errors. For non-linearly separable data, SVM uses kernel functions to transform the input space into a higher-dimensional space where a linear separation is possible. The algorithm is effective in high-dimensional spaces and is particularly useful for classification problems with clear margins of separation.

```
// Login user
router.post("/login", async (req, res) => {
  try {
    const { email, password } = req.body

    // Check if user exists
    const user = await User.findOne({ email })
    if (!user) {
      return res.status(400).json({ message: "Invalid credentials" })
    }

    // Check password
    const isMatch = await user.comparePassword(password)
    if (!isMatch) {
      return res.status(400).json({ message: "Invalid credentials" })
    }

    // Generate JWT
    const token = jwt.sign({ userId: user._id }, process.env.JWT_SECRET || "your_jwt_secret", { expiresIn: "7d" })

    res.json({
      token,
      user: {
        _id: user._id,
        name: user.name,
        email: user.email,
        role: user.role,
      },
    })
  } catch (error) {
    console.error(error)
    res.status(500).json({ message: "Server error" })
  }
})
```

Figure 4.2 Code Snippet of Registration and Login Module

4.1.2.3. Question Validation Module:

Before storing quiz questions, this module ensures each question meets predefined validation criteria to maintain consistency and reliability. This module performs a series of checks on each question to verify that it adheres to predefined validation rules. It ensures that the question text is not empty, there are at least two answer options, and exactly one of those options is marked as correct. Additionally, it checks for duplicate answer options and

validates that all inputs follow the correct data types and formatting standards. If any validation fails, the system generates clear and specific error messages to help administrators correct the issues promptly. By enforcing these standards, the module guarantees that only high-quality, well-structured questions are added to the quiz database, thereby enhancing the overall quiz experience and ensuring reliable assessment outcomes.

```
const validateQuestion = [
  body("text")
    .trim()
    .notEmpty()
    .withMessage("Question text is required"),

  body("options")
    .isArray({ min: 2 })
    .withMessage("Options must be an array with at least 2 items")
    .custom((opts) => opts.every(opt => typeof opt === "string" && opt.trim().length > 0))
    .withMessage("Each option must be a non-empty string"),

  body("correctOption")
    .isInt({ min: 0 }) // must be an integer >= 0
    .withMessage("Correct option must be a valid index")
    .custom((value, { req }) => {
      const options = req.body.options;
      if (!Array.isArray(options)) return false;
      return value >= 0 && value < options.length; // check index bounds
    })
    .withMessage("Correct option index must be within the range of options"),

  body("categoryId")
    .notEmpty()
    .withMessage("Category ID is required")
    .custom((value) => mongoose.Types.ObjectId.isValid(value))
    .withMessage("Invalid category ID"),
];
```

Figure 4.3 Code Snippet of Question Validation Model

4.1.2.4. Scores of Players Module:

The **Scores of Players Module** is responsible for calculating, recording, and managing user scores after each quiz attempt. When a user completes a quiz, this module evaluates the submitted answers by comparing them with the correct options stored in the database. Once the evaluation is complete, the score along with additional metrics such as the number of correct and incorrect answers, the time taken to complete the quiz, and the percentage accuracy, is stored in the database under the user's performance history. Each score record is linked to the specific quiz ID and user ID. This module also supports result retrieval for

dashboards and leaderboards, allowing users to view their past performance and enabling administrators to analyze user engagement and quiz difficulty levels. By maintaining a detailed log of user scores, this module enhances transparency.

```
router.get("/leaderboard", [auth, adminAuth], async (req, res) => {
  try {
    const results = await QuizResult.find().populate("player", "name").populate("category", "name")

    const playerStats = {}

    results.forEach((result) => {
      const playerId = result.player._id.toString()
      const playerName = result.player.name
      const categoryId = result.category._id.toString()
      const categoryName = result.category.name
      const score = result.totalQuestions > 0 ? (result.score / result.totalQuestions) * 100 : 0

      if (!playerStats[playerId]) {
        playerStats[playerId] = {
          _id: playerId,
          playerName,
          quizzesTaken: 0,
          totalScore: 0,
          bestScore: 0,
          categories: {},
        }
      }

      playerStats[playerId].quizzesTaken++
      playerStats[playerId].totalScore += score
      playerStats[playerId].bestScore = Math.max(playerStats[playerId].bestScore, score)

      if (!playerStats[playerId].categories[categoryId]) {
        playerStats[playerId].categories[categoryId] = {
          categoryId,
          categoryName,
          quizzesTaken: 0,
          totalScore: 0,
          bestScore: 0,
        }
      }

      playerStats[playerId].categories[categoryId].quizzesTaken++
      playerStats[playerId].categories[categoryId].totalScore += score
      playerStats[playerId].categories[categoryId].bestScore = Math.max(
        playerStats[playerId].categories[categoryId].bestScore,
        score
      )
    })

    const leaderboard = Object.values(playerStats).map((player) => {
      player.averageScore = player.quizzesTaken > 0 ? player.totalScore / player.quizzesTaken : 0

      Object.values(player.categories).forEach((category) => {
        category.averageScore = category.quizzesTaken > 0 ? category.totalScore / category.quizzesTaken : 0
      })
```

Figure 4.4 Code Snippet of Scores of Players Module

4.2. Testing

4.2.1. Test Cases for Unit Testing

Table 4.1 Test Cases for Unit Testing of Quiz Management System

SN	Action	Input	Expected Outcomes	Actual Outcomes	Test Results
1	Launch Application	http://localhost:3000/login	Quiz landing page loads	Quiz landing page loads	Pass
2	User Registration	Valid user details	User created, token generated	User created, token issued	Pass
3	Quiz Creation (Admin)	Valid quiz data	Quiz successfully created	Quiz successfully created	Pass
4	Question Validation	Question with a missing correct option	Error message returned	Error message returned	Pass
5	Attempt Quiz	User submits quiz with answers	Score calculated and saved	Score returned and saved	Pass
6	View Leaderboard	GET /api/leaderboard	List of top scores displayed	Top scores displayed	Pass

Table 4.2: Test Cases for Unit Testing of Fisher Yates Shuffle Algorithm

SN	Action	Input	Expected Outcomes	Actual Outcomes	Test Results
1	Shuffle a list of quiz questions	Array of distinct questions	Questions in a new random order	Same questions in a order	Pass
2	Shuffle an empty list	Null	Return empty list without error	Returned empty list	Pass
3	Shuffle a list with one element	[“Q1”]	Same list returned (no change)	Same list returned	Pass
4	Shuffle maintains all elements	Array of n unique questions	All elements present after shuffle (no loss/duplication)	All elements retained	Pass
5	Multiple shuffles produce variations	Same input list shuffled multiple times	Different output orders across runs	Output varied in each shuffle	Pass

4.2.2. Test Cases for System Testing

Table 4.3 Test Cases for System Testing of Quiz Management System

SN	Action	Input	Expected Outcomes	Actual Outcomes	Test Results

1	Launch Application	http://localhost:3000/login	Landing page displayed	Landing page displayed	Pass
2	User Registration	POST /api/register with valid name, email, and password	User registered, token issued	User registered, token issued	Pass
3	Duplicate Registration	Same email again	Error: Email already exists	Error: Email already exists	Pass
4	User Login	POST /api/login with valid credentials	User logged in, redirected to dashboard	User logged in successfully	Pass
5	Admin Adds Questions	POST /api/admin/quiz/:quizId/question/add	Questions added to quiz	Questions added successfully	Pass
6	Fisher-Yates Shuffle Check	Start quiz multiple times	Different question orders per attempt	Questions appear in random order	Pass
7	Submit Quiz with Answers	POST /api/quiz/:quizId/submit	Score calculated and saved	Score calculated and saved	Pass

CHAPTER 5

CONCLUSION AND FUTURE MANAGEMENTS

5.1. Conclusion

In summary, the **Quiz Management System** project presents a comprehensive solution for creating, managing, and evaluating online quizzes with a focus on fairness, flexibility, and efficiency. The system is designed to support a wide range of users—from administrators managing quiz content to participants attempting quizzes and viewing results. Its modular architecture ensures that core features such as registration, quiz creation, question validation, timed quiz attempts, and result tracking function seamlessly within a unified platform.

A key strength of the system lies in its use of the **Fisher-Yates Shuffle Algorithm**, which ensures each quiz attempt features randomized question order, thereby minimizing predictability and promoting fairness among participants. This, combined with structured **question validation**, ensures high-quality content integrity. The system also supports **secure user authentication** using JWT and bcrypt encryption, ensuring that user data remains safe and access is role-restricted. Real-time scoring, leaderboard generation, and result analytics add significant value by enhancing the assessment experience for learners and educators alike.

The technical implementation leverages modern technologies including **React for the frontend** and **Node.js with MongoDB** for the backend, resulting in a responsive and scalable web-based application. Functional requirements, such as dynamic quiz allocation and instant feedback, have been effectively met.

As the project moves toward deployment, it has the potential to serve educational institutions, training platforms, and certification providers by offering a smart and customizable assessment environment. Future enhancements could include support for multimedia-based questions, adaptive difficulty levels using AI, mobile application integration, and advanced analytics dashboards for educators. Overall, the Quiz Management System stands as a valuable contribution to digital learning and assessment tools, demonstrating how technology can streamline knowledge evaluation while maintaining integrity and engagement.

5.2. Future Management

In future has an immense room for improvement. The following are some of the future Managements for the project:

- **Adaptive Quiz Feature:** Add smart quizzes that change question difficulty based on how well the user is doing. This makes the quiz more personalized and fair for each user.
- **Support for Images and Videos:** Include questions with pictures, audio, or videos to make quizzes more fun and helpful, especially for learning different subjects.
- **Better Reports and Analytics:** Provide detailed results for both users and admins, showing things like scores, time taken, and areas to improve. This helps track progress and improve quiz content.
- **Gamification and Rewards:** Add features like points, badges, and levels to make quizzes more engaging. Users can earn rewards and feel motivated to keep learning.

References

[1] H. D. W. Wawan Gunawan, "An Application of Multimedia for Basic Arabic Learning Using Fisher- Yates Shuffle Algorithm on Android Based," 2019. [Online]. Available: https://scispace.com/pdf/fisher-yates-shuffle-algorithm-for-randomization-math-exam-3uebub1h5x.pdf .

[2] N. M. N. I. F. I. A. Noorfaizalfar id Mohd Noor, "The Development of Autonomous Examination Paper Application: A Case Study in UiTM Perlis Branch," 2019. [Online]. Available: https://scispace.com/pdf/fisher-yates-shuffle-algorithm-for-randomization-math-exam-3uebub1h5x.pdf .

[3] L. V. R. S. B. Vinayak Hegde, "The Framework for Web-Based Automated Online Question Paper Generator through JEE," 2018. [Online]. Available: https://scispace.com/pdf/fisher-yates-shuffle-algorithm-for-randomization-math-exam-3uebub1h5x.pdf .

[4] S. e. al, "Puzzle game “Numbers in English” Based on Android Application using The FYSA," 2018. [Online]. Available: https://scispace.com/pdf/fisher-yates-shuffle-algorithm-for-randomization-math-exam-3uebub1h5x.pdf .

[5] S. &. Z. Hasan, "Implementation of Fisher-Yates Algorithm (FYSA) to Randomize Online Exam Questions.," 2017. [Online]. Available: https://scispace.com/pdf/fisher-yates-shuffle-algorithm-for-randomization-math-exam-3uebub1h5x.pdf .

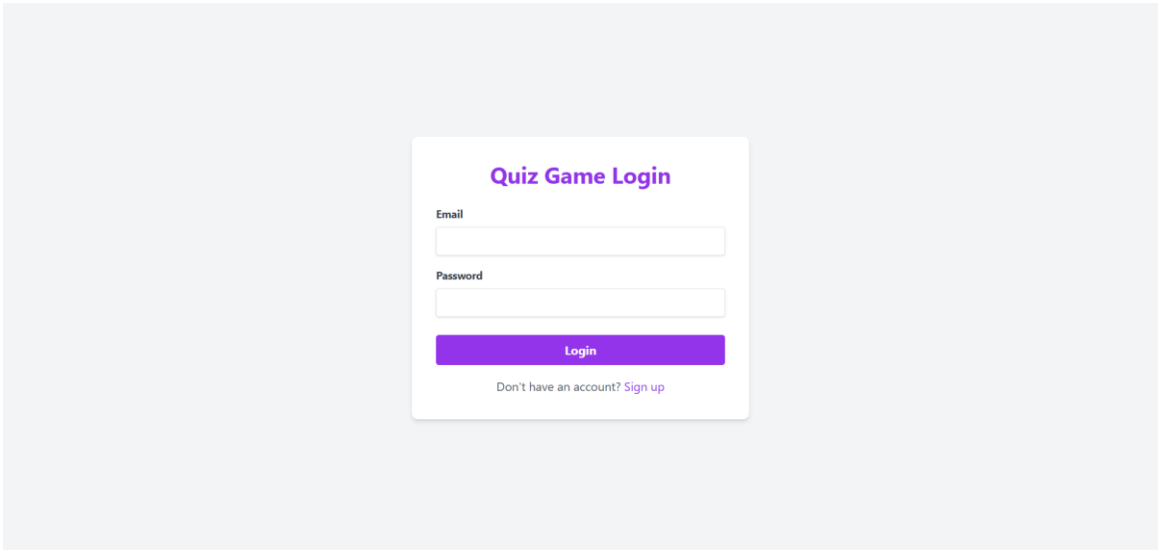
[6] T. e. al., "the challenges and obstacles linked to the manual creation of question papers by educator," 2017. [Online]. Available: https://www.researchgate.net/publication/374830859_Design_of_Question_Generator_System_QPGS_Using_Fisher-Yates_Shufling_Algorithm.

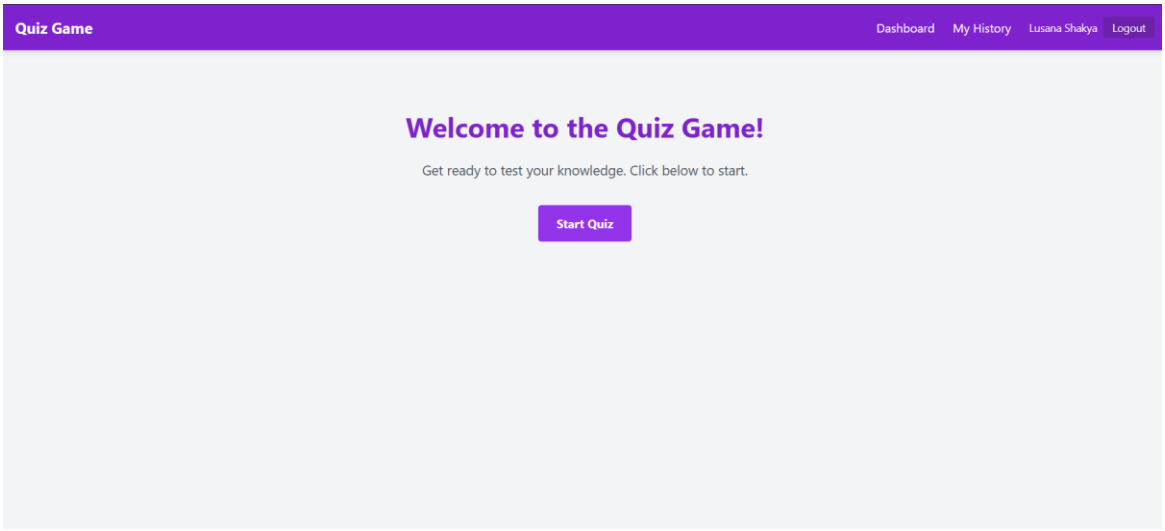
[7] N. e. al, "Automatic Question Generation System," 2014. [Online]. Available: https://www.researchgate.net/publication/374830859_Design_of_Question_Generator_System_QPGS_Using_Fisher-Yates_Shufling_Algorithm.

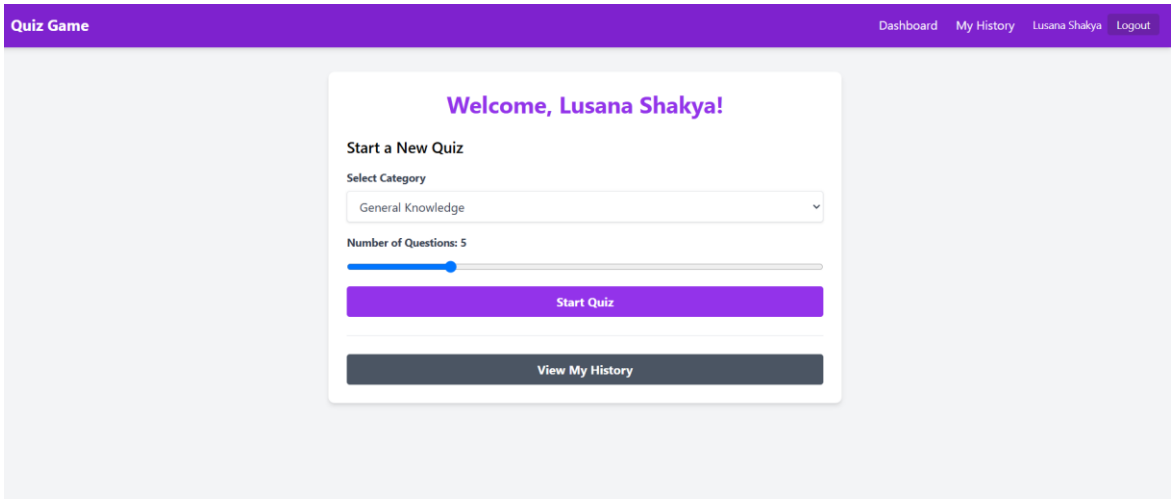
[8] A. e. al., "Automatic Question Generator designed for crafting Basque language test questions," 2006. [Online]. Available: https://www.researchgate.net/publication/374830859_Design_of_Question_Generator_System_QPGS_Using_Fisher-Yates_Shufling_Algorithm.

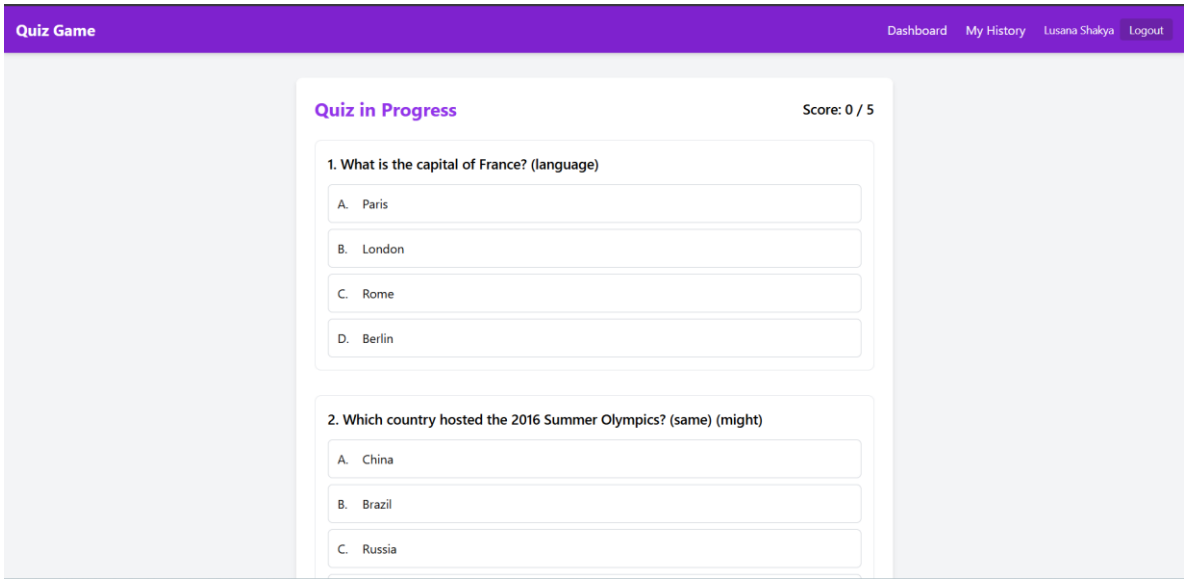
[9] P. Model. [Online]. Available: https://www.google.com/search?q=prototype+model+diagram&udm=2&sxsrf=AE3TifO1cUn3p1_spWfiPzspk3d1Bw4aUQ%3A1750485620056#vhid=tXEOTSXOkl4NrM&vssid=mosaic.

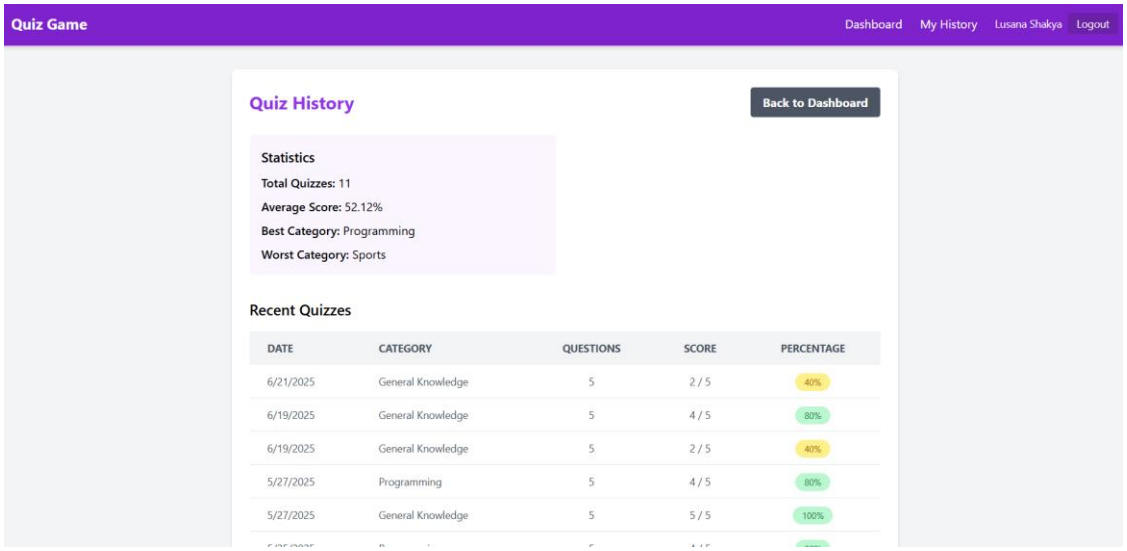
APPENDICES


Quiz Game Login
Email
Password
Login
Don't have an account? Sign up


Quiz Game
Dashboard
My History
Lusana Shakya
Logout
Welcome to the Quiz Game!
Get ready to test your knowledge. Click below to start.
Start Quiz


Quiz Game
Dashboard
My History
Lusana Shakya
Logout
Welcome, Lusana Shakya!
Start a New Quiz
Select Category
General Knowledge
Number of Questions: 5
Start Quiz
View My History


Quiz Game
Dashboard
My History
Lusana Shakya
Logout
Quiz in Progress
Score: 0 / 5
1. What is the capital of France? (language)
A. Paris
B. London
C. Rome
D. Berlin
2. Which country hosted the 2016 Summer Olympics? (same) (might)
A. China
B. Brazil
C. Russia


Quiz Game
Dashboard
My History
Lusana Shakya
Logout
Quiz History
Back to Dashboard
Statistics
Total Quizzes: 11
Average Score: 52.12%
Best Category: Programming
Worst Category: Sports
Recent Quizzes
DATE
CATEGORY
QUESTIONS
SCORE
PERCENTAGE
6/21/2025
General Knowledge
5
2 / 5
40%
6/19/2025
General Knowledge
5
4 / 5
80%
6/19/2025
General Knowledge
5
2 / 5
40%
5/27/2025
Programming
5
4 / 5
80%
5/27/2025
General Knowledge
5
5 / 5
100%