FUNTASTICIQ

A PROJECT REPORT

Submitted by

Roll Number	Registration Number	Student Code	Student Name
22010301133	22013002523	BWU/BCA/22/144	Pritha Ghosh
22010301136	22013002526	BWU/BCA/22/147	Somosree Dey
22010301127	22013002516	BWU/BCA/22/137	Shrashtha Saha
22010301138	22013002528	BWU/BCA/22/149	Mousumi Paul
22010301116	22013002502	BWU/BCA/22/123	Soumita Karmakar
22010301140	22013002530	BWU/BCA/22/151	Subhadip Sahu

in partial fulfillment for the award of the degree

of

BACHELOR OF COMPUTER APPLICATION

IN

Department of Computational Sciences

BRAINWARE UNIVERSITY

398, Ramkrishnapur Road, Barasat, North 24 Parganas, Kolkata - 700 125

JULY & 2025

FUNTASTICIQ

Submitted by

Roll Number	Registration Number	Student Code	Student Name
22010301133	**22013002523**	**BWU/BCA/22/144**	**Pritha Ghosh**
22010301136	**22013002526**	**BWU/BCA/22/147**	**Somosree Dey**
22010301127	**22013002516**	**BWU/BCA/22/137**	**Shrashtha Saha**
22010301138	**22013002528**	**BWU/BCA/22/149**	**Mousumi Paul**
22010301116	**22013002502**	**BWU/BCA/22/123**	**Soumita Karmakar**
22010301140	**22013002530**	**BWU/BCA/22/151**	**Subhadip Sahu**

in partial fulfillment for the award of the degree

of

BACHELOR OF COMPUTER APPLICATION

in

Department of Computational Sciences

BRAINWARE UNIVERSITY

398, Ramkrishnapur Road, Barasat, North 24 Parganas, Kolkata - 700 125

BRAINWARE UNIVERSITY

398, Ramkrishnapur Road, Barasat, North 24 Parganas, Kolkata - 700 125

Department of Computational Sciences

BONAFIDE CERTIFICATE

Certified that this project report **FUNTASTICIQ** is the Bonafide work of "**Pritha Ghosh** (Roll No:22010301133), **Somosree Dey** (Roll No:22010301136), **Shrashtha Saha** (Roll No:22010301127), **Soumita Karmakar** (Roll No:22010301116), **Mousumi Paul** (Roll No:22010301138), **Subhadip Sahu** (Roll No:22010301140**)**" who carried out the project work under my supervision.

SIGNATURE
Dr. Jayanta Aich
HEAD OF THE DEPARTMENT
Department of Computational Sciences
398, Ramkrishnapur Road, Barasat,
North 24 Parganas, Kolkata - 700 125

SIGNATURE
Dr. Kaushik Chanda
SUPERVISOR
Associate Professor
Department of Computational Science

ACKNOWLEDGEMENT

Project Title: FUNTASTICIQ

Project Group ID: BCA22C003

We, the undersigned project members, would like to express our heartfelt gratitude to [Dr. Kaushik Chanda, Associate Professor], Department of Computational Sciences, Brainware University, for their invaluable guidance, support, and motivation throughout the course of this project. Their expert advice and insightful suggestions were instrumental in the successful completion of our work.

We are also immensely thankful to Dr. Jayanta Aich, Head of the Department of Computational Sciences, for providing the required resources and encouragement necessary for this project.

We acknowledge the cooperation and support of all faculty members and staff of the Department of Computational Sciences. Their input helped us remain on track and improve the quality of our research.

We extend our special thanks to our families, friends, and peers for their continuous encouragement and support during this academic journey and for motivating us to achieve our goals.

Lastly, we express our sincere gratitude to Brainware University for giving us the opportunity to undertake this project as a part of our academic curriculum.

Project Members:

1. BWU/BCA/22/144- Pritha Ghosh
2. BWU/BCA/22/147- Somosree Dey
3. BWU/BCA/22/149- Mousumi Paul
4. BWU/BCA/22/137- Shrashtha Saha
5. BWU/BCA/22/123- Soumita Karmarkar
6. BWU/BCA/22/151- Subhadip Sahu

Date:19.05.2025
Department of Computational Sciences

Brainware University

ABSTRACT

FuntasticIQ is a fun and interactive quiz game application built using the MERN stack, which includes MongoDB, Express.js, React, and Node.js. This user-friendly app lets players pick their favorite subject and dive into exciting quizzes that test their knowledge in a challenging yet enjoyable way. After finishing a quiz, users can see their scores, compare them with others, and find out who the top scorer of the week is. The app also keeps track of each player's highest scores, encouraging them to keep practicing and aim for the top spot. With its mix of education and entertainment, FuntasticIQ creates a lively and motivating space where users can learn new things, have fun, and compete with friends, making it a great way to boost knowledge while enjoying a game.

TABLE OF CONTENTS

List of Tables

List of Figures

Chapter 1. Introduction

FuntasticIQ is a cutting-edge quiz game application built to deliver a rich, interactive experience using the MERN stack. It allows users to register, log in, select quizzes from various subjects, track their scores, and compete on leaderboards—all within a responsive and intuitive interface. The application's primary mission is to fuse education with entertainment, offering a platform where users can both enjoy themselves and expand their knowledge.

Motivation and Educational Context

The inception of FuntasticIQ stems from the growing need to make learning more engaging in an increasingly digital world. Traditional educational methods, such as rote memorization or passive reading, often fail to captivate modern learners who thrive on interactivity and instant gratification. Quiz games bridge this gap by offering an active learning experience that challenges users while providing immediate feedback—a proven method for improving retention and understanding.

Research highlights that gamified learning boosts motivation, encourages critical thinking, and fosters a sense of achievement. FuntasticIQ capitalizes on these principles by presenting knowledge in a fun, competitive format. For example, answering a series of questions under a time constraint sharpens quick thinking, while leaderboards tap into users' competitive instincts, driving them to improve. Beyond individual benefits, the application promotes social learning by allowing users to compare scores and challenge peers, creating a community of learners.

The integration of technology further amplifies these benefits. By leveraging a web-based platform, FuntasticIQ eliminates barriers such as physical location or access to specialized resources, making education universally accessible. Its responsive design ensures that users can engage with it anytime, anywhere—whether on a laptop during a study session or a smartphone during a commute.

Choice of Technology: The MERN Stack

The decision to use the MERN stack was deliberate and strategic, driven by its efficiency and cohesion. Here's a deeper look at each component:

- **MongoDB**: As a NoSQL database, MongoDB excels in handling unstructured data, which is ideal for storing diverse quiz questions, user profiles, and performance records. Its schema-less nature allows for rapid iteration and adaptation as the application evolves.
- **Express.js**: This minimalist framework simplifies the creation of RESTful APIs, enabling smooth communication between the frontend and backend. Its middleware capabilities enhance security and request handling, crucial for a user-facing application.

- **React**: Known for its efficiency and flexibility, react powers the frontend with reusable components and a virtual DOM, ensuring fast rendering and a seamless user experience. Its ecosystem, including libraries like Redux, supports complex state management.
- **Node.js**: By running JavaScript on the server side, Node.js ensures a consistent development experience and supports high concurrency—key for handling multiple users taking quizzes simultaneously.

The synergy of these technologies creates a full-stack JavaScript environment that accelerates development, reduces context-switching between languages, and supports scalability. Additionally, the MERN stack's active community and extensive documentation provided valuable resources throughout the project.

Target Audience

FuntasticIQ caters to a broad demographic, including students seeking supplementary learning tools, educators looking to engage their classes, and casual users who enjoy trivia and competition. Its subject variety—from academic disciplines to general knowledge—ensures inclusivity, while its accessibility across devices accommodates diverse lifestyles and preferences.

Purpose and Goals

At its core, FuntasticIQ seeks to redefine how people interact with educational content. By offering a space where users can learn through play, it encourages lifelong learning and intellectual curiosity. The application's technical design prioritizes scalability and performance, ensuring it can grow with its user base while maintaining a high-quality experience.

Chapter 2. Objectives

FuntasticIQ was developed with clear, actionable objectives to guide its design and functionality. These goals ensure the application meets both user needs and technical standards.

- **Subject Diversity**: The application offers quizzes spanning multiple domains—in programming such as programming language (e.g. C, C++, Python, java), Software Engineering, Web Design, Computer Architecture etc. This variety appeals to users with different interests and expertise levels, making the platform versatile and inclusive.
- **User Experience**: A clean, intuitive interface is paramount. Navigation is streamlined with menus for subjects, quizzes, leaderboards, and profiles, while responsive design ensures usability across devices. Features like progress indicators and clear feedback enhance accessibility and reduce frustration.
- **Performance Tracking**: Users earn points for correct answers, with scores tracked per quiz and subject. High scores are stored in user profiles, providing a record of progress and a benchmark for improvement. This system motivates users to revisit challenging topics and refine their skills.
- **Competitive Environment**: Weekly leaderboards showcase the top 10 performers, updated dynamically based on quiz completions. This fosters a sense of rivalry and achievement, encouraging users to participate regularly and aim for top ranks.
- **Scalability**: The application leverages cloud-based hosting and a modular architecture to accommodate growth. MongoDB's horizontal scaling and Node.js's event-driven model ensure performance remains stable as user numbers increase.
- **Security**: Robust measures protect user data, including JWT-based authentication, password hashing with crypt, and HTTPS encryption. These safeguards build trust and comply with best practices for privacy.
- **Continuous Improvement**: The team commits to ongoing updates, incorporating user feedback and emerging technologies. This includes adding new subjects, refining the UI, and enhancing backend efficiency over time.

These objectives collectively aim to create a platform that is engaging, reliable, and adaptable, serving as both an educational resource and a source of entertainment.

Chapter 3. Planning

The development of FuntasticIQ followed a meticulous planning process to ensure timely delivery and quality outcomes. This phase laid the groundwork for efficient collaboration and execution.

Project Management Methodology

The team adopted Agile, a flexible and iterative approach suited to software development. Key practices included:

- **Sprints**: Two-week cycles focused on specific deliverables, such as completing the authentication module or designing the quiz interface. Each sprint ended with a review and retrospective to assess progress and adjust plans.
- **Daily Stand-Ups**: Brief meetings where team members shared updates, discussed challenges, and planned their day. This kept everyone aligned and responsive to issues.
- **Sprint Planning**: At the start of each sprint, tasks were prioritized and assigned based on the project backlog, ensuring focus on high-impact features.
- **Retrospectives**: Post-sprint reflections identified what worked well (e.g., effective pair programming) and what needed improvement (e.g., better estimation of task complexity).

Planning Activities

- **Scope Definition**: The team outlined essential features—user management, quiz functionality, scoring, leaderboards, and profiles—while leaving room for future enhancements. This clarity prevented overextension and maintained focus.
- **Milestone Setting**: Key deadlines included completing frontend wireframes (Week 2), backend APIs (Week 4), database integration (Week 5), and end-to-end testing (Week 7). These milestones provided structure and measurable progress points.
- **Resource Allocation**: Tools like Git ensured version control, Slack facilitated communication, Trello tracked tasks, and Visual Studio Code served as the primary IDE. Roles were distributed based on expertise—e.g., frontend developers focused on React, while backend specialists handled Node.js and MongoDB.
- **Risk Assessment**: Potential risks included integration issues between stack components, delays from learning new libraries, and server overload during testing. Mitigation strategies involved regular code reviews, documentation study sessions, and load testing early in the cycle.
- **Communication Plan**: Beyond daily stand-ups, weekly stakeholder updates via email or zoom ensured transparency. A shared Google Drive housed documentation, fostering collaboration.

Tools and Collaboration

- **Version Control**: Git with GitHub enabled branching, pull requests, and code reviews, maintaining a clean and collaborative codebase.

- **Task Management**: Trello boards visualized workflows with columns for “To Do,” “In Progress,” and “Done,” enhancing task visibility.
- **Communication**: Slack channels separated technical discussions, general updates, and urgent alerts, streamlining information flow.

This structured yet adaptable planning approach ensured the project stayed on track, with room to pivot as needed based on testing and feedback.

Chapter 4. Requirement Analysis

Requirement analysis was a critical step in defining FuntasticIQ functionality and performance criteria. This phase involved gathering insights from stakeholders and users to create a comprehensive blueprint.

Functional Requirements

These specify the application's core behaviors and features:

Category	Requirement	Description
User Management	• Register with username, email, password. • Login/logout. • Password reset via email.	Users can create accounts, access them securely, and recover access if needed.
Quiz Management	• Admins create/edit/delete quizzes. • Quizzes have titles, subjects, and multiple-choice questions	Admins maintain a dynamic quiz library, ensuring content variety and relevance.
Quiz Taking	• Select subjects/quizzes • Answer questions with timer • Track responses	Users engage with quizzes interactively, with time pressure adding challenge.
Scoring	• Award points per correct answer • Display score post-quiz • Store subject-specific highs	Performance is quantified and preserved, motivating users to improve.
Leaderboard	• Show top 10 weekly scores • Allow rank comparison	A competitive element drives engagement through visibility and rivalry.
User Profile	• Display high scores and history • Update personal info	Users can track their progress and customize their experience.

Table 1: Functional Requirement Analysis (Source: Self-Created)

Non-Functional Requirements

These define quality attributes and operational constraints:

Category	Requirement	Description
Responsive Design	• Seamless operation across devices	Ensures accessibility on desktops, tablets, and phones with consistent UI/UX.
Performance	• Page load <2s • Handle100+ concurrent users	Fast and reliable under varying loads, enhancing user satisfaction.
Security	• HTTPS, encrypted data • JWT authentication	Protects user privacy and prevents unauthorized access.
Scalability	• Horizontal scaling • Efficient large data handling	Supports growth in users and content without performance loss.
Usability	• Intuitive navigation • WCAG-compliant accessibility	Easy to use for all, including those with disabilities.
Reliability	• 99.9% uptime Automated backups	Dependable service with minimal downtime and data loss risk.

Table 2: Non-Functional Requirement Analysis (Source: Self-Created)

Requirement Gathering Process

- **Stakeholder Interviews**: Engaged educators, students, and trivia enthusiasts to pinpoint desired features (e.g., diverse subjects) and pain points (e.g., complex UIs).
- **User Surveys**: Distributed online questionnaires to 50 potential users, revealing preferences like timed quizzes (80% approval) and leaderboards (75% interest).
- **Market Research**: Analyzed competitors like Quizlet and Kahoot, identifying gaps (e.g., limited subject variety) that FuntasticIQ could fill with broader content and competition.

Validation and Prioritization

Requirements were validated through mockups and discussions, ensuring feasibility. High-priority items (e.g., user authentication, quiz functionality) were tackled first, while lower-priority enhancements (e.g., profile customization) were scheduled for later iterations.

This thorough analysis ensured FuntasticIQ aligned with user expectations and technical realities, forming a solid foundation for development.

By thoroughly analyzing these requirements, the FickFinder project aims to address the needs of its users effectively while maintaining high standards for performance, security, and scalability. This analysis serves as the cornerstone for the successful execution of the project.

Chapter 5. System Flow

Architecture Diagram:

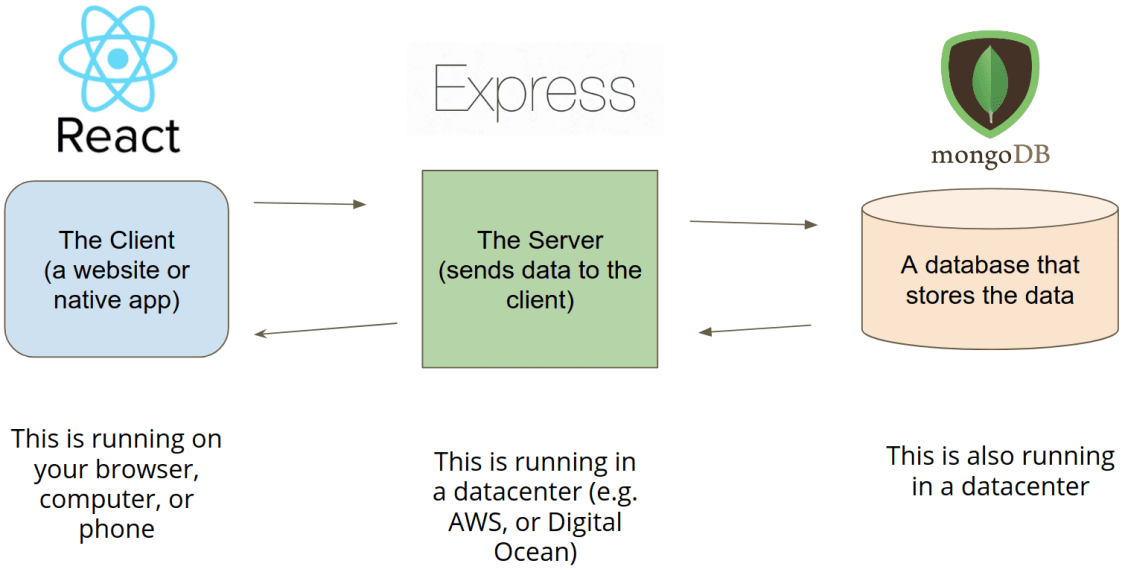

Figure 1: Architecture Diagram (Source: Iteachrecruiter,2025)

UML DIAGRAMS

UML INTRODUCTION

The Unified Modeling Language (UML) is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. The UML is a very important part of developing objects-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

GOALS OF UML

The primary goals in the design of the UML are:

Provide users with a ready-to-use, expressive visual modeling language so they can develop and exchange meaningful models.

1. Provide extensibility and specialization mechanisms to extend the core concepts.
2. Being independent of particular programming languages and development processes.
3. Provide a formal basis for understanding the modeling language.
4. Encourage the growth of the OO tools market.
5. Support higher-level development concepts such as collaborations, frameworks, patterns and components.

TYPES OF UML DIAGRAM

Each UML diagram is designed to let developers and customers view a software system from a different perspective and in varying degrees of abstraction. UML diagrams commonly created in visual modeling tools include:

A.USE CASE DIAGRAM

Display the relationship among actors and Use Cases. A use case is a set of scenarios that describing an interaction between a user and a system. A use case diagram displays the relationship among actors and uses cases. The two main components of a use case diagram are use cases and actors.

Use Cases (Functionalities):

1. **Sign in**
    - Allows existing users to log in to their FUNTASTICIQ account using credentials.
2. **Signup**
    - Enables new users to register and create an account on FUNTASTICIQ.
3. **Play Quiz**
    - Users can participate in different types of quizzes that test their intelligence and knowledge in a fun way.
4. **View Leaderboard**
    - After attempting quizzes, users can view their ranking and compare scores with others.
5. **Take Test**
    - Users can take structured tests or mock quizzes, possibly for practice or assessment purposes.
6. **Manage Quizzes**
    - Users can view, create, or manage their attempted quizzes. This may also include saving or retaking them.

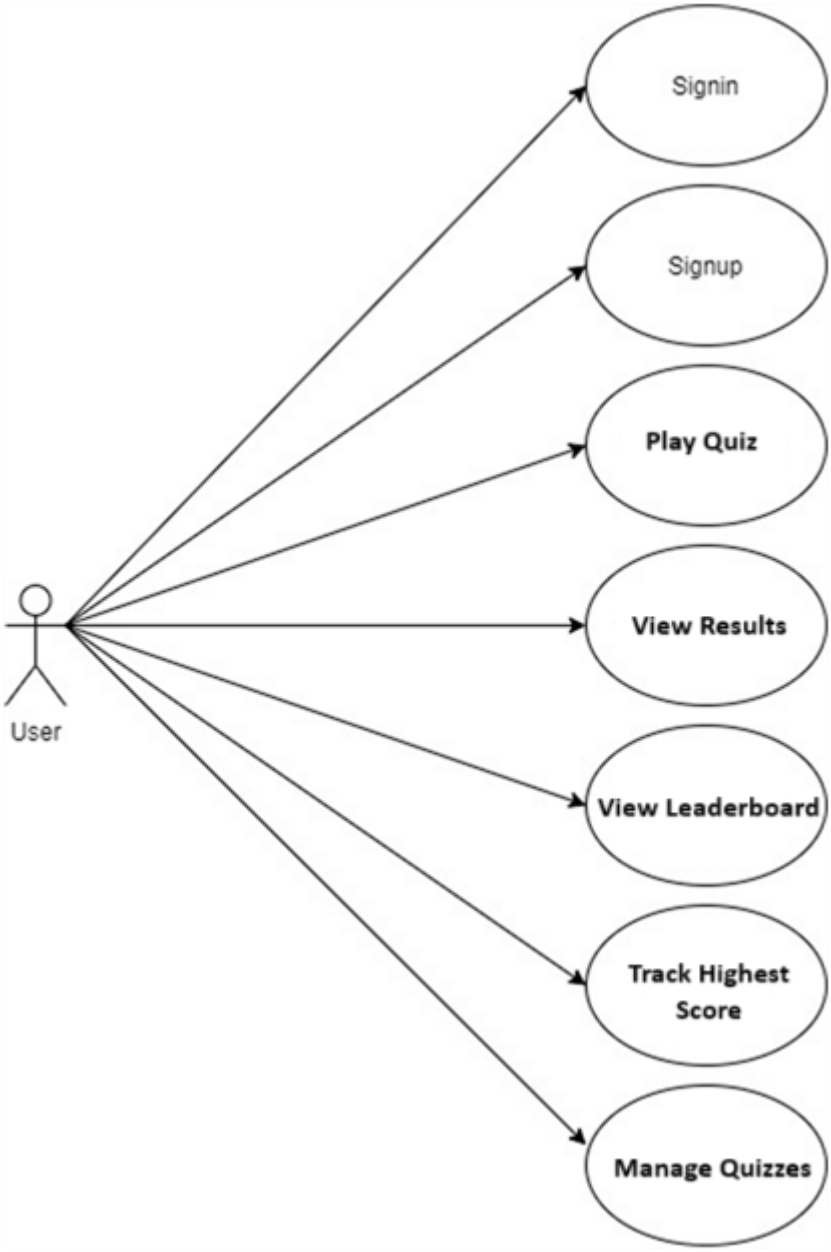

Figure 2: Use case diagram (Source: Self-Created)

B. CLASS DIAGRAM

Class diagrams are widely used to describe the types of objects in a system and their relationships. Class diagrams model class structure and contents using design elements such as classes, packages and objects. Class diagrams describe three different perspectives when designing a system, conceptual, specification, and implementation. These perspectives become evident as the diagram is created and help solidify the design.

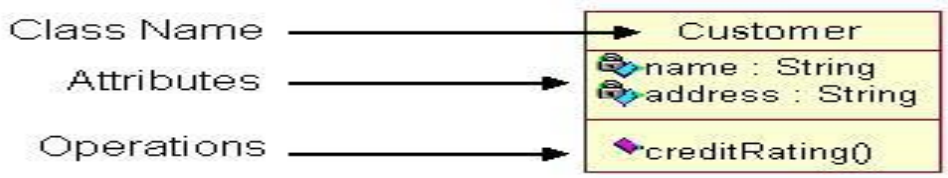

Figure3: Class diagram (Source: Self-Created)

Class diagrams also display relationships such as containment, inheritance, associations and others. The association relationship is the most common relationship in a class diagram. The association shows the relationship between instances of classes. Another common relationship in class diagrams is a generalization

C. SEQUENCE DIAGRAM

Purpose:

To show how different components of the FUNTASTICIQ system interact when a student takes an online quiz.

Actors/Entities Involved:

1. **Student** – The end user who takes the quiz.
2. **Online Examination System (FUNTASTICIQ platform)** – The interface and logic that manages user interaction.
3. **Database** – Stores user credentials and system data.
4. **Question Bank** – Contains a pool of quiz/test questions.
5. **Answer Key** – Contains correct answers used for evaluating the user's responses.

Process Flow:

1. **Student logs in** to the FUNTASTICIQ platform.
2. The **Online Examination System** sends credentials to the **Database**.
3. The **Database verifies** and **confirms** the student's credentials.
4. Once verified, the **system grants access** to begin the quiz.
5. Student **starts the exam**.
6. The system **requests questions** from the **Question Bank**.
7. The **Question Bank returns** the relevant questions.

8. The system **displays the questions** to the student.
9. Student **submits answers** after completing the quiz.
10. The system **sends answers to the Answer Key** for evaluation.
11. **Scores are calculated** and returned.
12. The system **displays the scores** to the student.
13. Student **logs out**.

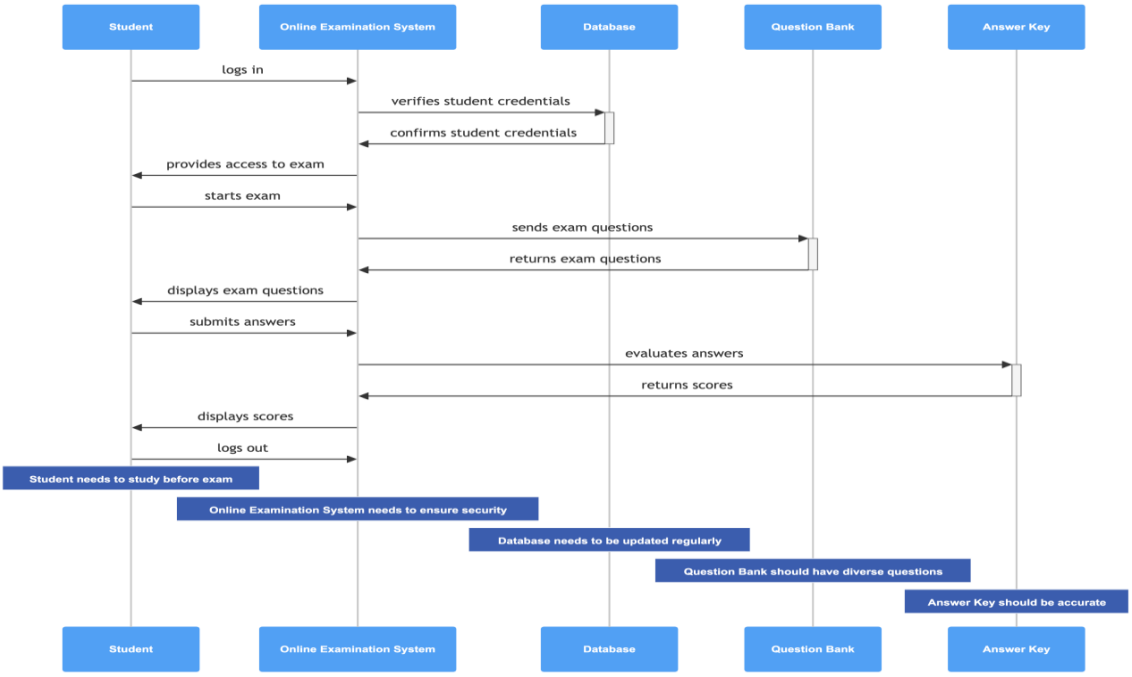

Figure 4: Sequence diagram (Source: Self-Created)

E. ACTIVITYDIAGRAM

Purpose:

To depict how different users (Admin, Lecturer, Student) interact with the FUNTASTICIQ platform, and how the system processes those interactions to manage exams and generate results.

Roles & Responsibilities:

1. Admin:

- **Run Server**: Initializes the system and makes it ready for use.
- **Set Timer**: Defines the duration for each quiz or exam.
- **Start the Exam**: Activates the exam for students to participate.
- **Print Final Degree**: Receives and prints the final score/report after evaluation.

2. Lecturer:

- **Set Instruction**: Adds guidelines or rules for the quiz/exam.
- **Set Questions**: Inputs the questions into the FUNTASTICIQ question bank for the exam.

3. Student:

- **Login**: Authenticates and accesses the platform.
- **Take Exam**: Attempts the exam based on available instructions and questions.

4. System (FUNTASTICIQ Platform):

- **Check Answers**: Evaluates the student's responses based on the answer key.
- **Calculate the Final Degree**: Processes the total score or grade.
- **Upload the Final Degree**: Updates the result to the system so it can be accessed or printed.

Workflow Summary:

1. **Admin** starts the system and sets up the exam environment.
2. **Lecturer** provides necessary instructions and uploads questions.
3. **Student** logs in and takes the exam.
4. The **System** automatically checks answers, calculates results, and uploads the final score.
5. **Admin** prints the final report or certificate (Final Degree).

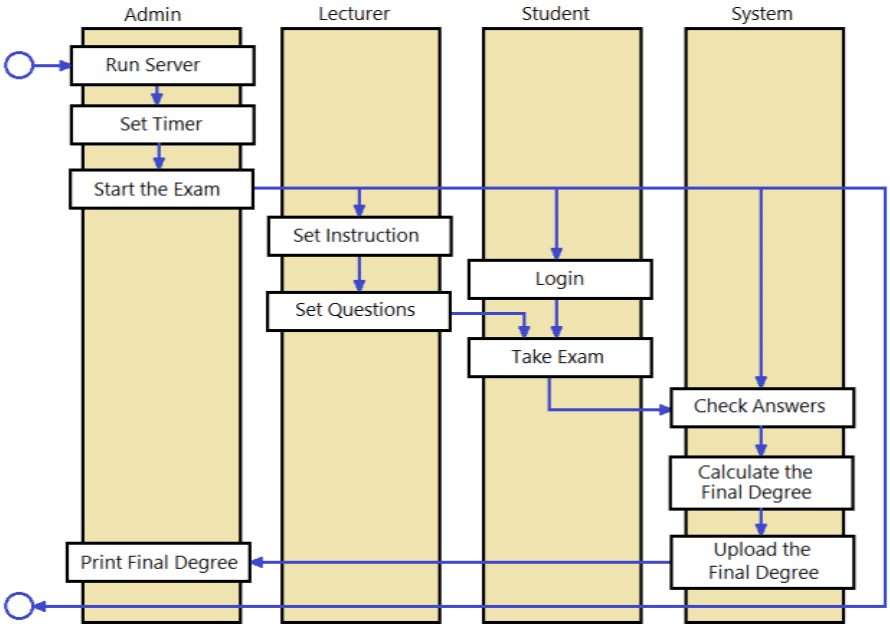

Figure 5: Activity diagram (Source: Self-Created)

Purpose:

To illustrate the architecture of the FUNTASTICIQ application and show the interaction between key software modules, including users, requests, database, and persistence layers.

Components and Their Roles:

1. Management (<<application>>)

- **Role:** Acts as the main controller of the application logic.
- **Function:** Coordinates user and request handling and communicates with other modules.

2. User Component

- **Role:** Represents the user-related operations.
- **Function:** Handles functionalities like login, profile management, and user-specific requests in FUNTASTICIQ.

3. Request Component (Request implementing IRequest)

- **Role:** Manages student or admin requests like quiz submissions, score retrieval, etc.
- **Function:** Facilitates communication between user actions and backend processes.

4. Persistence Layer (<<infrastructure>> Persistence)

- **Role:** Acts as the data handler.
- **Function:** Connects the logic layer (user/request) with the database and ensures data is stored and retrieved properly.

5. Database (JDBC)

- **Role:** Stores all persistent data (e.g., user info, quiz questions, answers, scores).
- **Function:** Back-end storage system accessed through JDBC (Java Database Connectivity).

Interactions:

- The Management application component communicates with both User and Request components.
- User and Request send data or commands to the Persistence layer.
- Persistence interfaces with the Database using JDBC.
- All communication is modular, showing a decoupled and scalable system design.

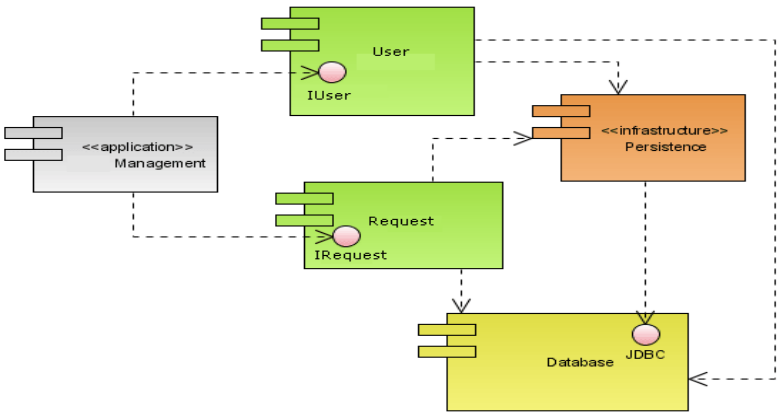

Figure 6: Component diagram (Source: Self-Created)

- This architecture supports real-time quiz operations, secure result generation, and scalable backend integration.
- Using Node.js with MongoDB ensures fast performance, asynchronous processing, and easy data handling.
- Follows MVC architecture, making the system maintainable and modular.

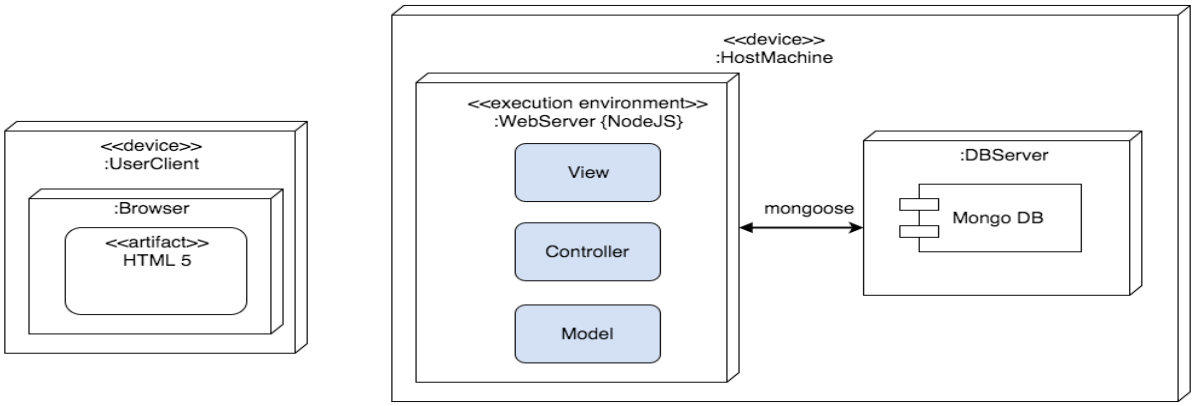

Figure 7: Deployment diagram (Source: Self-Created)

Chapter 6. Proposed Design

Login

```
import React, { useState } from "react";
import { useNavigate } from "react-router-dom";
import axios from "axios";
import "./Login.css";
const Login = () => {
  const navigate = useNavigate();
  const [user, setUser] = useState({
    email: "",
    password: "",
  });
  const handleChange = (e) => {
    setUser({ ...user, [e.target.name]: e.target.value });
  };
  const handleSubmit = async (e) => {
    e.preventDefault();

    try {
      const response = await axios.post("http://localhost:5000/api/login", user);
```

```
      if (response.data. success) {
        alert ("Login Successful!");
        localStorage.setItem("token", response.data. token);
        navigate("/dashboard");
      } else {
        alert (response.data. message);
      }
    } catch (error) {
      alert ("Invalid credentials, please try again.");
    }
};
return (
  <div className="login-container">
    <div className="login-box">
      <h2>Login to FuntasticIQ</h2>

      <input
        type="email"
        name="email"
```

```
    placeholder="Enter your email"
    value={user.email}
    onChange={handleChange}
    required
  />

  <input
    type="password"
    name="password"
    placeholder="Enter your password"
    value={user.password}
    onChange={handleChange}
    required
  />
  <button onClick={handleSubmit}>Login</button>
  <p>
    Don't have an account?{" "}
    <span className="signup-link" onClick={() => navigate("/signup")}>
      Sign Up
    </span>
```

```
                    </p>
                  </div>
                </div>
              );
            };
```

dashoboard.jsx

```
import React, { useState, useEffect } from 'react';
import { Link } from "react-router-dom";
import DashboardNavbar from "../components/DashboardNavbar";
import "./Dashboard.css";


const Dashboard = () => {
  const [searchTerm, setSearchTerm] = useState("");
  const [animateCards, setAnimateCards] = useState(false);
  const [isFocused, setIsFocused] = useState(false);
  useEffect(() => {
    const timeout = setTimeout(() => {
      setAnimateCards(true);
    }, 100); // Wait for DOM to mount
```

```
    return () => clearTimeout(timeout);
  }, []);


   // Get the username from localStorage
   const username = localStorage.getItem("username") || "User";



  const quizSubjects = [
    { name: "C Programming", icon: "/assets/c-programming.png", link: "/select-level/c-programming" },
    { name: "Computer Architecture", icon: "/assets/computer-architecture.png", link: "/select-level/computer-architecture" },
    { name: "Web Design", icon: "/assets/web-design.png", link: "/select-level/web-design" },
    { name: "Artificial Intelligence", icon: "/assets/AI.png", link: "/select-level/artificial-intelligence" },
    { name: "Data Structures", icon: "/assets/data-structures.png", link: "/select-level/data-structures" },
    { name: "Operating Systems", icon: "/assets/operating-systems.png", link: "/select-level/operating-systems" },
    { name: "Database Management", icon: "/assets/database-management.png", link: "/select-level/database-management" },
    { name: "Computer Networks", icon: "/assets/computer-networks.png", link: "/select-level/computer-networks" },
```

```
  { name: "Machine Learning", icon: "/assets/machine-learning.png", link: "/select-level/machine-learning" },
  { name: "Cloud Computing", icon: "/assets/cloud-computing.png", link: "/select-level/cloud-computing" },
  { name: "Software Engineering", icon: "/assets/software-engineering.png", link: "/select-level/software-engineering" },
  { name: "Cybersecurity", icon: "/assets/cybersecurity.png", link: "/select-level/cybersecurity" },
  { name: "C++ Programming", icon: "/assets/c++.png", link: "/select-level/c++-programming" },
  { name: "Python Programming", icon: "/assets/python.png", link: "/select-level/python-programming" },
  { name: "Java Programming", icon: "/assets/java.png", link: "/select-level/java-programming" },
 ];
 const filteredSubjects = quizSubjects.filter((subject) =>
  subject.name.toLowerCase().includes(searchTerm.toLowerCase())
 );
 return (
  <div className="dashboard-container">
   {/* Search bar */}
   <div className="search-container">
    <span className="search-icon">🔍</span>
    <input
```

```
  type="text"
  placeholder="Search subject..."
  className="search-input"
  value={searchTerm}
  onChange={(e) => setSearchTerm(e.target.value)}
  onFocus={() => setIsFocused(true)}
  onBlur={() => setTimeout(() => setIsFocused(false), 200)} // delay to allow click
  />
</div>
<DashboardNavbar />
<div className={dashboard-blur-wrapper ${isFocused && searchTerm ? "blurred" : ""}}>
{/* Floating icons */}
<div className="floating-icons-dashboard">
    <span className="icon-dashboard">❓</span>
    <span className="icon-dashboard">🎮</span>
    <span className="icon-dashboard">🎯</span>
    <span className="icon-dashboard">💡</span>
    <span className="icon-dashboard">🎊</span>
    <span className="icon-dashboard">🎉</span>
    <span className="icon-dashboard">🏅</span>
```

```
      <span className="icon-dashboard">🔥</span>
      <span className="icon-dashboard">🎲</span>
      <span className="icon-dashboard">🧩</span>
      <span className="icon-dashboard">📚</span>
      <span className="icon-dashboard">🏆</span>
      <span className="icon-dashboard">👥</span>
      <span className="icon-dashboard">🕶️</span>
      <span className="icon-dashboard">🧠</span>
      <span className="icon-dashboard">📈</span>
      <span className="icon-dashboard">📊</span>
      <span className="icon-dashboard">🎁</span>
      <span className="icon-dashboard">🌍</span>
      <span className="icon-dashboard">🌟</span>  {/* Star */}
      <span className="icon-dashboard">✅</span>
    </div>
```

The purpose of testing is to discover errors. Testing is the process of trying to discover each and every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of tests. Each test addresses a specific testing requirement.

Black Box Testing

In this system black box testing is performed by giving random inputs to the functionalities and verified whether expected results are obtained. For Example, random inputs are given to a login page and verified whether it's accepting even the wrong credentials. When these kinds of bugs are detected then these are fortified. In this testing we verified whether our application is according to the requirement specification. Any missing functionalities can be detected in this phase.

White Box Testing

White box Testing is performed by verifying the internal code of the application. Syntactic errors, Decisions, Control Flow and statements of the code are verified line by line for all the modules involved in our system such as Employee and Admin. We could detect control flow errors such as wrong redirection using white box testing.

7.3 Component Testing

This system has 2 Modules i.e. Student Module and Admin Module. All the modules are independently tested by isolating other modules. Each functionality of the modules is verified. All the individual pages like List of Donors; Reserve etc. are verified whether the input data is forwarded to the action page.

7.4 Unit Testing

Each functionality is verified if it is according to the specified requirements or not. Sometimes input validations are missed; there are solved in unit testing. For example, student registration needs email. This input must be unique in database so that users can be restricted from using the same email.

7.5 Integration Testing

Our system has two different modules involved i.e. student and Admin. Though component testing is performed on each module sometimes error generate due to incompatibility between the interfaces. This kind of errors can be detected in Integration Testing and is resolved.

7.6. Test Cases

Test id	Test Description	Test Design	Expected output	Actual output	Status

1	Check response on login	Enter correct employee id and password	Allow access	Allow access	Pass
2	Check response for login	Enter invalid username and click on sign in	Error message should display as incorrect details	Error message should be displayed	Pass
3	Check response for login	Enter invalid password and click on and sign in	Error message should display as incorrect details	Error message should be displayed	Pass
4	Check response for login	Without giving the username press on sign in	Error message displaying give the input	Error message is displayed showing give the input	Pass

Table 3: Test Cases for Login (Source: Self-Created)

Test id	Test Description	Test Design	Expected output	Actual output	Status
1	Check response on retrieving a student data	Giving the invalid student id	Error message displaying the value doesn't exist	Error message displaying the value doesn't exist	Pass
2	Check response on retrieving a student data	Giving the invalid student id	Work details of a employee is retrieved	Work details of a employee is retrieved	Pass

Table 4: Test Cases for Student (Source: Self-Created)

Chapter 7. Experimental Result

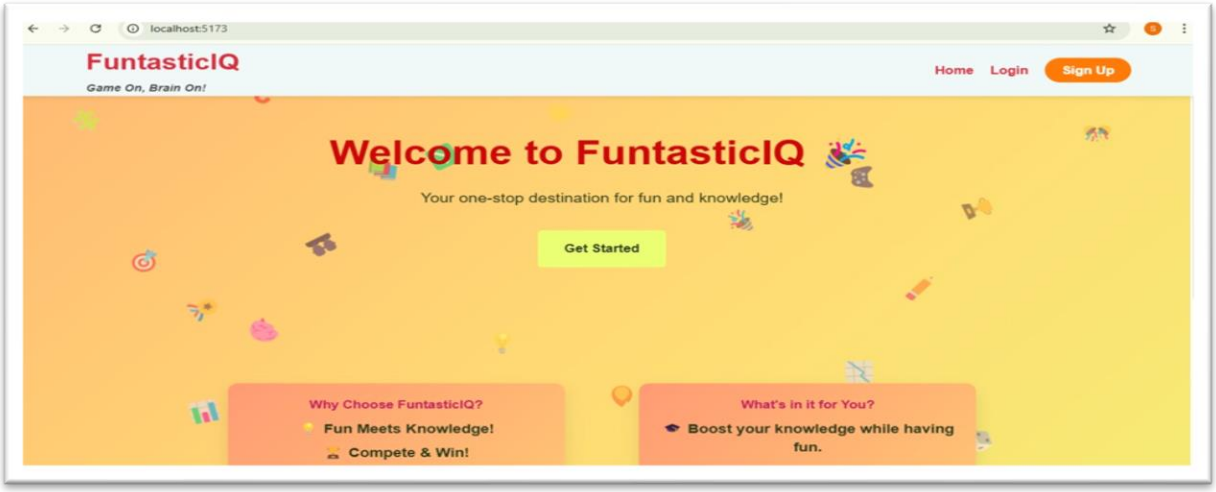

Figure 8: Home Page (Source: Self-Created)

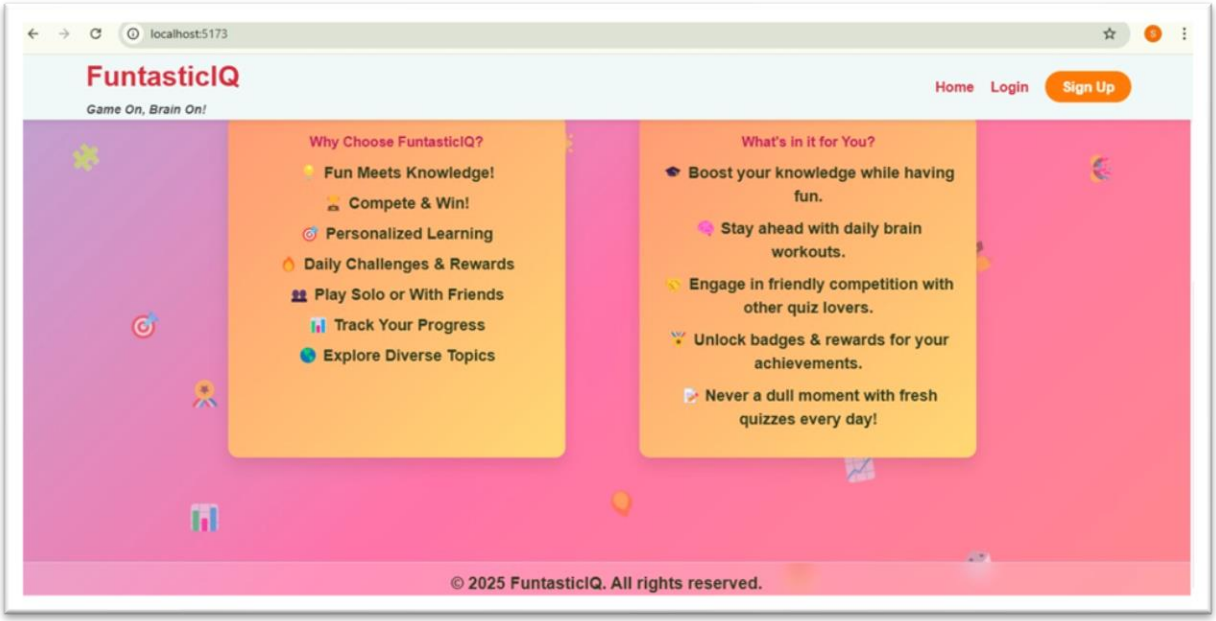

Figure 9: Home Page Contd… (Source: Self-Created)

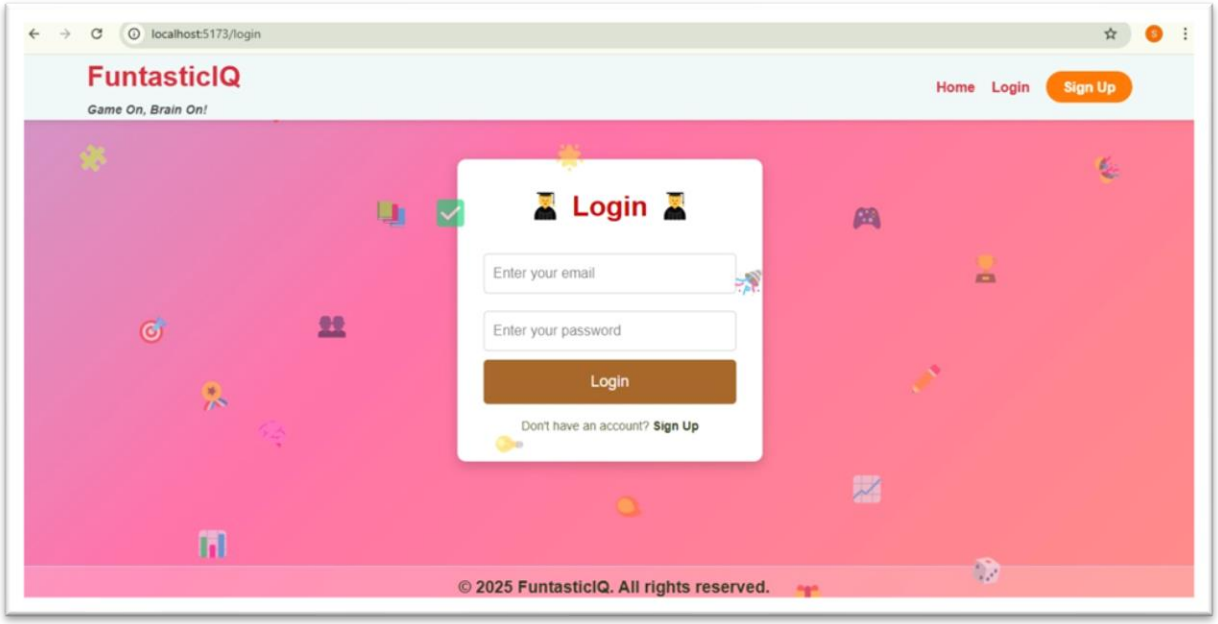

Figure 10: Login Page (Source: Self-Created)

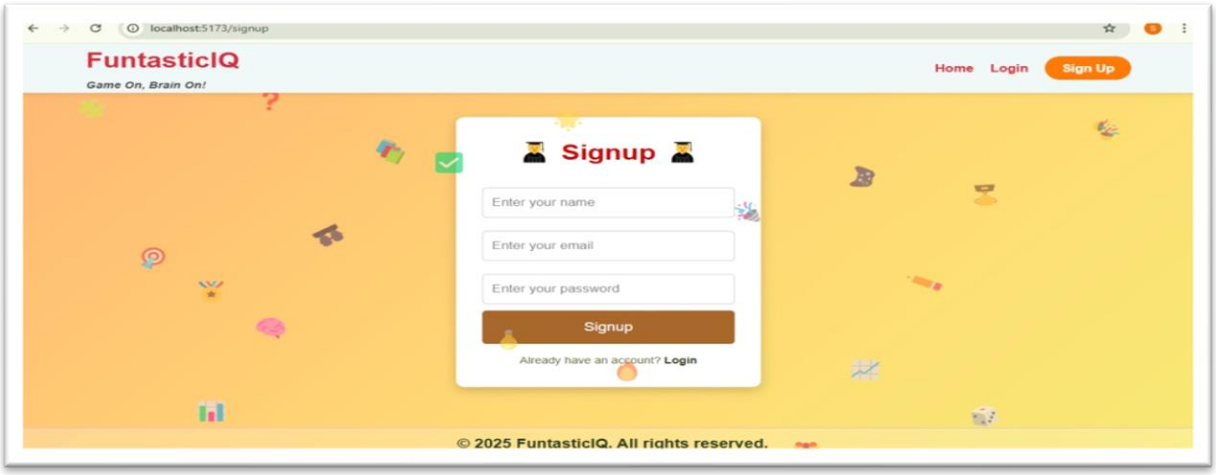

Figure 11: Signup Page (Source: Self-Created)

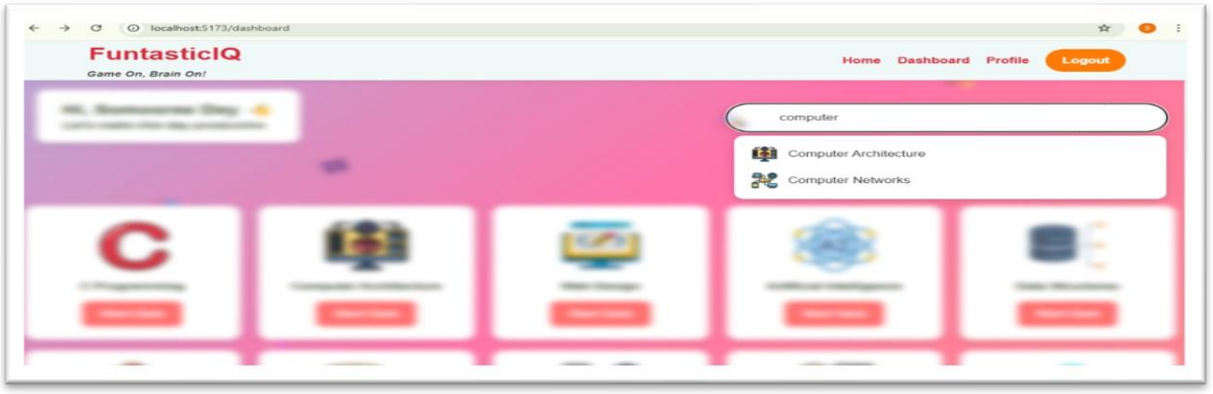

Figure 12: Dashboard Page (Source: Self-Created)

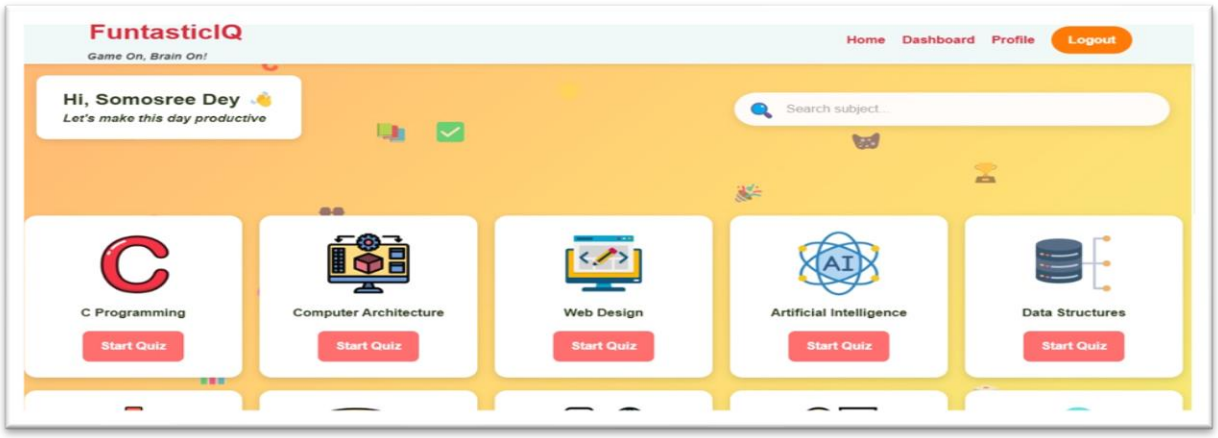

Figure 13: Dashboard Page Contd… (Source: Self-Created)

Figure 14: Dashboard Page Contd… (Source: Self-Created)

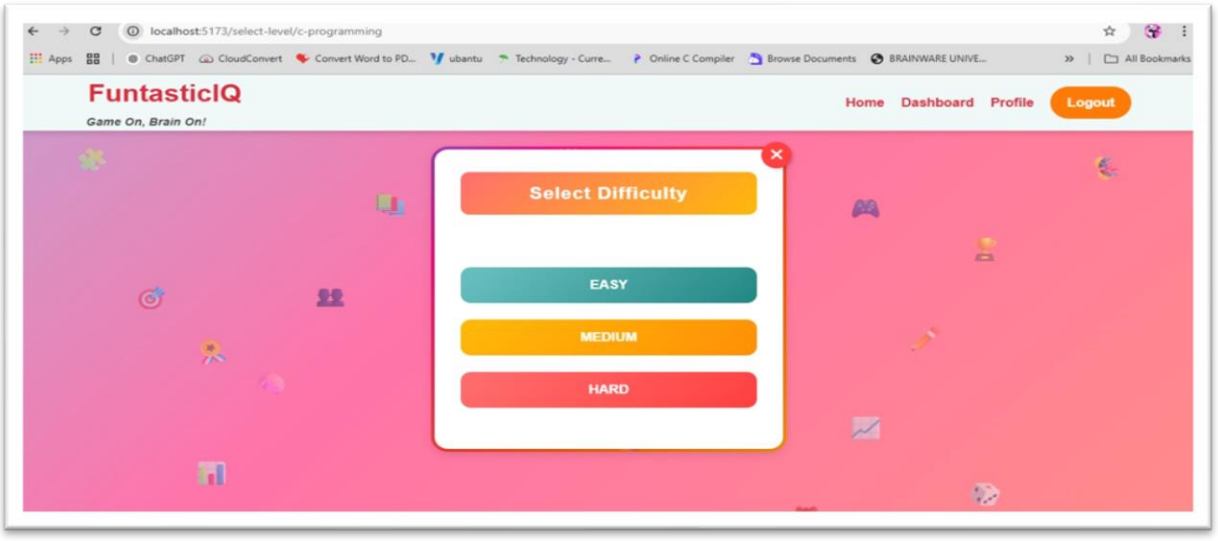

Figure 15: Select Level (Source: Self-Created)

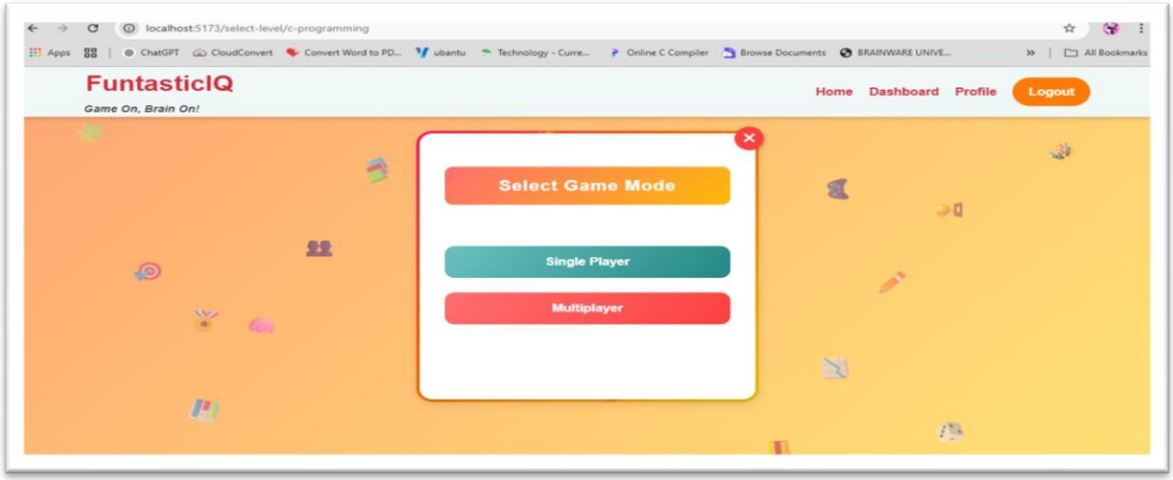

Figure 16: Select Level Contd… (Source: Self-Created)

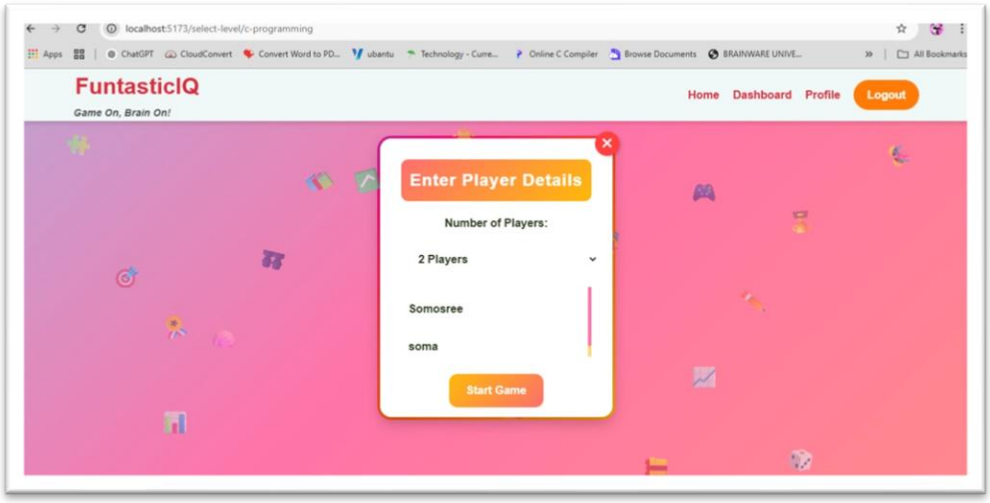

Figure 17: Select Level Contd… (Source: Self-Created)

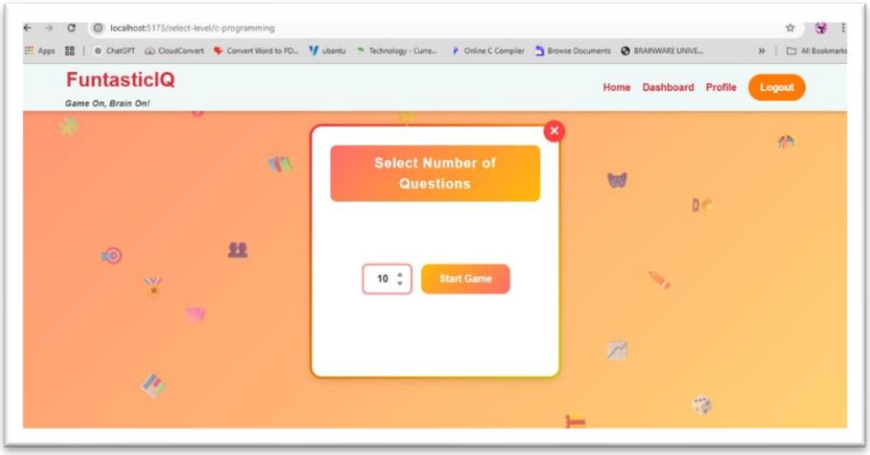

Figure 18: Select Level Contd… (Source: Self-Created)

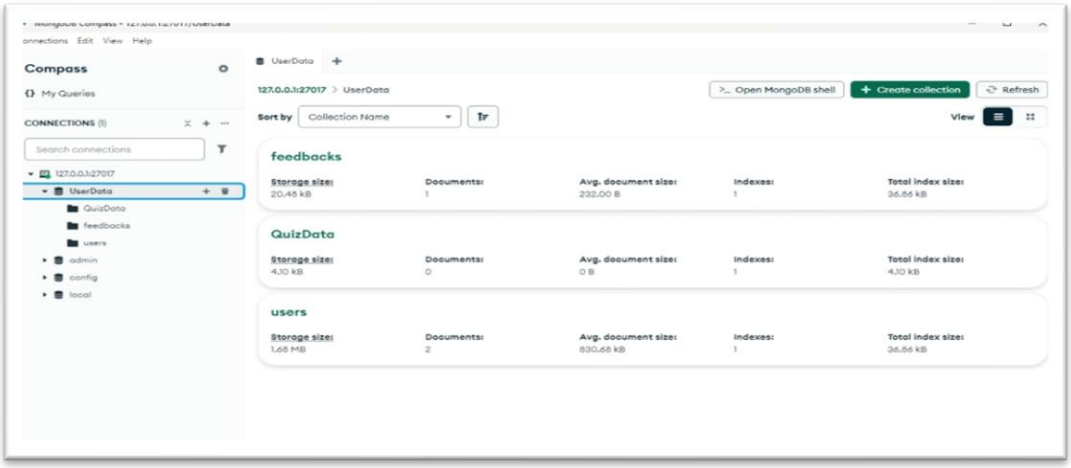

Figure 19: Back End design (Source: Self-Created)

Chapter 8. Future Scope

Future Scope for FuntasticIQ

FuntasticIQ, as an interactive and gamified learning platform, holds immense potential for growth in both educational effectiveness and user engagement. As the platform evolves, several opportunities exist for improvement and innovation. The future scope for FuntasticIQ can be categorized into six primary areas: **feature expansion, technological advancements, user engagement, business scalability, global expansion**, and **compliance considerations**.

1. Feature Expansion

Currently focused on quiz-based and gamified learning experiences, FuntasticIQ can grow significantly by introducing new educational tools and content types:

- **Interactive Video Lessons:** Adding short, gamified video lessons before quizzes could enhance conceptual understanding and cater to diverse learning styles.
- **Live Quiz Competitions:** Real-time multiplayer quizzes or tournaments would allow users to compete with friends or other learners, promoting healthy competition and increased engagement.
- **Personalized Learning Paths:** Introducing AI-driven learning journeys based on quiz performance and preferences will provide tailored experiences and improve learning outcomes.
- **Discussion Forums:** Creating topic-specific community spaces where users can discuss quiz topics, share knowledge, or ask questions would enhance collaborative learning.
- **Teacher/Parent Dashboards:** Offering separate dashboards for educators and parents to track performance, assign quizzes, and monitor improvement could make FuntasticIQ more appealing in school environments.

2. Technological Advancements

To stay competitive and scalable, FuntasticIQ must embrace emerging technologies:

- **AI-Powered Adaptive Quizzing:** Using machine learning algorithms to adapt question difficulty based on user performance will create a smarter and more efficient learning experience.
- **Gamification Engines:** Upgrading the gamification model with dynamic badges, XP points, skill trees, and rank-based leaderboards can foster long-term engagement.

- **Voice and Speech Recognition:** Enabling voice-based quiz answering for younger users or accessibility purposes could broaden the platform's usability.
- **Progressive Web App (PWA):** Developing FuntasticIQ as a PWA will allow offline quiz access and better performance on low-end devices, improving accessibility.
- **Integration with Learning Management Systems (LMS):** Connecting with systems like Moodle or Google Classroom would make FuntasticIQ a useful add-on for institutional learning environments.

3. User Engagement and Personalization

Enhancing how users interact with the platform is key to increasing retention and satisfaction:

- **Customized Avatars and Profiles:** Allowing users to personalize avatars and profiles creates a sense of ownership and identity.
- **Daily Streaks and Missions:** Introducing daily challenges, login streaks, or learning missions will gamify consistency and foster habit formation.
- **Social Features:** Features like friend lists, chat rooms, and collaborative quizzes can boost engagement through peer learning.
- **In-App Achievements and Rewards Store:** A point-based rewards system linked to unlockable items (e.g., skins, stickers, bonus quizzes) will enhance motivation and interactivity.

4. Business Scalability and Monetization

To ensure sustainability, FuntasticIQ can explore multiple monetization strategies while maintaining user value:

- **Freemium Model:** Keeping core features free while offering advanced analytics, premium quizzes, and offline access through a paid plan.
- **Institutional Licensing:** Offering bulk licensing and customization options to schools and educational institutions.
- **Brand Collaborations:** Partnering with publishers, ed-tech companies, or subject-matter experts for co-branded or certified content.
- **Advertising and Sponsorships:** Carefully integrated, non-intrusive ads or sponsorships from educational brands could bring revenue without compromising user experience.
- **Merchandise and Content Sales:** Selling learning kits, printable worksheets, or educational merchandise could provide an additional revenue stream.

5. Global Expansion

To reach a wider audience, the platform can focus on expanding internationally:

- **Localization and Language Support:** Adding multiple language options and culturally relevant content will make FuntasticIQ accessible to learners from diverse regions.
- **Curriculum Mapping:** Aligning quizzes with country-specific educational standards (e.g., CBSE, GCSE, or Common Core) could help in broader adoption in formal education.
- **Local Collaborations:** Partnering with regional educators, influencers, and educational NGOs can help FuntasticIQ gain credibility and traction in specific markets.
- **Cross-Border Compatibility:** Supporting different currencies, date/time formats, and user regulations will ensure a seamless global experience.

6. Compliance and Ethical Considerations

As the platform grows and collects more user data, especially from children, adhering to legal and ethical standards will be crucial:

- **Data Protection Laws:** Ensuring compliance with COPPA (Children's Online Privacy Protection Act), GDPR, and similar data privacy regulations.
- **Parental Consent Mechanisms:** For users under 13, integrating parent/guardian consent systems and limiting certain data collection or interactions.
- **Content Moderation:** Developing automated and manual moderation systems for user-generated content to ensure a safe and inclusive environment.
- **Accessibility Standards:** Ensuring the platform meets WCAG 2.1 guidelines for web accessibility so learners with disabilities can also benefit from the platform.

Chapter 09. Conclusion

FuntasticIQ successfully automates the traditional quiz-taking process by providing an engaging and interactive digital platform. This application eliminates the need for paper-based quizzes, making it an eco-friendly solution. It can be accessed and managed remotely via web and Android devices, offering flexibility to users. FuntasticIQ reduces the manual effort required to organize quizzes, track scores, and manage leaderboards, as all processes are handled automatically through the MERN stack backend. The application ensures accurate and real-time information, such as quiz results, leaderboard rankings, and user progress, enhancing the overall user experience.

Chapter 10. Appendices

Appendix 1: Survey and User Feedback

The following is a summary of the user feedback collected through online surveys during usability testing of the FuntasticIQ platform.

- **Total Participants:** 50
- **Survey Method:** Online questionnaire with 10 questions covering usability, design, engagement, and learning effectiveness.

Key Feedback:

- **Ease of Use:** 88% of users found the platform easy to navigate, with clear instructions and intuitive quiz flows.
- **Engagement:** 92% of participants reported that the quiz format made learning more enjoyable and interactive.
- **Performance:** 87% of users experienced smooth transitions between questions with no noticeable lag.
- **User Interface:** 78% of users found the UI visually appealing and responsive across devices, but suggested adding dark mode and customizable themes.
- **Leaderboard Feature:** 81% appreciated the competitiveness introduced through the leaderboard, motivating them to retake quizzes.

Appendix 2: Test Cases and Results

This section lists the core test cases used to validate FuntasticIQ's key features.

Test Case 1: User Registration

- **Description:** Test new user registration functionality.
- **Input:** Name, Email: user1@example.com, Password: funIQ123
- **Expected Outcome:** Account created and redirected to quiz selection page.
- **Result:** Pass

Test Case 2: Quiz Attempt Flow

- **Description:** Test complete quiz-taking process with 5 questions.
- **Input:** Selected Quiz: "General Knowledge", Timer: 60 seconds per question

- **Expected Outcome:** User completes quiz, receives score summary.
- **Result:** Pass

Test Case 3: Leaderboard Update

- **Description:** Ensure scores update in real time on the leaderboard.
- **Input:** User completes quiz with high score
- **Expected Outcome:** Leaderboard reflects updated ranking.
- **Result:** Pass

Appendix 3: Load Testing Results

The load testing assessed FuntasticIQ's ability to handle concurrent users and maintain performance.

- **Testing Tool:** Apache JMeter
- **Test Scenario:** 1,000 concurrent users registering, taking quizzes, and checking leaderboards.

Key Metrics:

- **Average Response Time:** 2.1 seconds
- **Server CPU Utilization:** 78% at peak
- **Error Rate:** 0.3% (3 out of 1,000 requests failed intermittently)
- **Quiz Completion Success Rate:** 97%

Appendix 4: Data Privacy and Security

Outlined below are the security protocols implemented to safeguard user data on FuntasticIQ.

- **Encryption:** AES-256 encryption used for securing stored data.
- **Authentication:** JWT-based authentication ensures secure login and session management.
- **Data Protection:** Full compliance with GDPR and CCPA policies. Users can request access, modification, or deletion of their data through account settings.
- **Security Audits:** Regular scans and vulnerability assessments performed to prevent breaches.

Appendix 5: Project Timeline

The Gantt chart below summarizes the development lifecycle of FuntasticIQ, highlighting key phases.

- **Phase 1:** Requirement Gathering and Analysis – 2 weeks
- **Phase 2:** UI/UX Design and System Architecture – 2 weeks
- **Phase 3:** Frontend Development – 4 weeks
- **Phase 4:** Backend Development – 4 weeks
- **Phase 5:** Integration, Testing, and Bug Fixing – 3 weeks
- **Phase 6:** User Testing and Feedback Collection – 2 weeks
- **Phase 7:** Deployment and Maintenance – Ongoing

References

1. Deterding, S., Dixon, D., Khaled, R. and Nacke, L., 2011, September. From game design elements to gamefulness: defining" gamification". In *Proceedings of the 15th international academic MindTrek conference: Envisioning future media environments* (pp. 9-15).

2. Gee, J.P., 2003. What video games have to teach us about learning and literacy. *Computers in entertainment (CIE)*, *1*(1), pp.20-20.

3. Grinberg, M., 2018. *Flask web development*. " O'Reilly Media, Inc.".

4. Hoque, S., 2018. *Full-stack react projects: Modern web development using react 16, node, express, and mongodb*. Packt Publishing Ltd.

5. Iteachrecruiters, 2025.*Frontend Vs Backend Developers (for Recruiters).* Available at: https://www.iteachrecruiters.com/blog/frontend-vs-backend-developers-for-recruiters/ Accessed on: [18.05.2025]

6. Koster, R., 2013. *Theory of fun for game design*. " O'Reilly Media, Inc.".

7. Mahmood, A., Ahmed, M.U. and Ahmad, H.S., 2017. Introduction to Web Development.

8. Norman, D.A., 2013. *The Design of*. New York: Basic Books.