

QR-BASED CAMPUS EXIT MANAGEMENT SYSTEM

Submitted in partial fulfillment of the requirements

For the award of degree of

BACHELORS OF TECHNOLOGY IN COMPUTER SCIENCE & ENGINEERING

Submitted by:

N BHAGYESWARI

(20NR1A0572)

R RAHUL

(20NR1A0595)

N LALITHA LAYA DEVI

(20NR5A0575)

D BHAGYA LAKSHMI

(20NR1A05A7)

**Under the guidance of
Mr. S. Durga Prasad
Associate Professor**



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
BABA INSTITUTE OF TECHNOLOGY & SCIENCES**

(Accredited by NAAC | Approved by AICTE, New Delhi)

Affiliated to JNTU – GV, Vizianagaram

2020 – 2024



BABA INSTITUTE OF TECHNOLOGY AND SCIENCES

(Accredited by NAAC | Approved by AICTE, New Delhi)

Affiliated to JNTU – GV

BONAFIDE CERTIFICATE

This is to certify that the project work entitled as “**QR – BASED CAMPUS EXIT MANAGEMENT SYSTEM**”. The project work is done by **N BHAGYESWARI (20NR1A0572), R RAHUL (20NR1A0595), N LALITHA LAYA DEVI (21NR5A0575), D BHAGYA LAKSHMI (20NR1A05A7)** in partial fulfilment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering during final viva voce examination held on _____.

Project Guide

Head of the department

External Examiner

DECLARATION

We hereby declare that this report for a project entitled “**QR – BASED CAMPUS EXIT MANAGEMENT SYSTEM**” has been developed by us under the supervision of **Mr. S. Durga Prasad**, Associate Professor, Department of Computer Science & Engineering, Baba Institute of Technology and Sciences, Visakhapatnam in the fulfilment of requirements. It is our own and not submitted to any other college/University any time before.

Place: BITS – VIZAG

Date:

By:

N BHAGYESWARI

(20NR1A0572)

R RAHUL

(20NR1A0595)

N LALITHA LAYA DEVI

(20NR5A0575)

D BHAGYA LAKSHMI

(20NR1A05A7)

ACKNOWLEDGEMENT

We would like to express our deep gratitude to our project guide MR. S. DURGA PRASAD, Associate Professor, Department of Computer Science and Engineering, BITS Vizag, for his guidance unsurpassed knowledge, and immense encouragement.

We are grateful to MRS. U. Padma Mohan, Head of the Department, Department of Computer Science and Engineering, for providing us with the required facilities for the completion of the project work.

We are very much thankful to the Principal Dr. D Poorna Sathyanarayana and Management of BITS, PM Palem, for their encouragement and cooperation in carrying out this work.

We express our thanks to all teaching faculty of the Department of CSE, whose suggestions during reviews helped us in the accomplishment of our project. We would like to thank all nonteaching staff of the Department of CSE, BITS for providing great assistance in the accomplishment of our project.

We would like to thank our parents, friends, and classmates for their encouragement throughout our project period. At last, but not the least, we thank everyone for supporting us directly or indirectly in completing this project successfully.

N BHAGYESWARI	(20NR1A0572)
R RAHUL	(20NR1A0595)
N LALITHA LAYA DEVI	(20NR5A0575)
D BHAGYA LAKSHMI	(20NR1A05A7)

ABSTRACT

In today's educational institutions, ensuring the safety, security & surveillance of students and faculty as well within the campus premises is of paramount importance. This project introduces a novel system aimed at simplifying the exit process for college students and faculty during designated hours, enhancing campus security, and maintaining attendance records efficiently. The system leverages QR (Quick Response) code technology to generate unique identifiers for students and faculty, which are embedded onto their college identification cards.

Once integrated into the student/faculty identification system, the QR code serves as a digital passport granting authorized access for both students and faculty to exit the college campus during designated hours. To ensure proper monitoring and regulation, the system requires students/faculty to present their ID cards with the embedded QR code to the designated security personnel stationed at the exit points.

The security personnel utilizes a QR code scanner mobile application to verify the authenticity of the code and subsequently grants permission for the student/faculty to exit the premises. By implementing this technology-driven solution, the college administration can effectively monitor student and faculty's movements, thereby enhancing campus security and safety protocols.

Moreover, the system facilitates the automatic recording of student/lecturer's attendance, providing management & higher authorities with valuable data for attendance tracking and analysis. This feature not only simplifies administrative tasks but also ensures accurate attendance records, promoting accountability and transparency within the academic environment.

Overall, the integration of QR-code technology into the college campus exit system offers a robust solution for enhancing security measures while simultaneously optimizing administrative processes. This project represents a significant step forward in leveraging technology to create safer and more efficient educational environments.

INDEX

S.No.	Title	Pg.No
01.	Introduction	1-6
	1.1 Feasibility Study	2-3
	1.2 Existing System	4-11
	1.3 Proposed System	5-6
02.	Software Requirements	7-
	2.1 Introduction	8
	2.2 External Interface Requirement	8
	2.3 Functional Requirements	9
	2.4 Non-Functional Requirements	11-18
03.	Analysis	12-18
	3.1 Use Case Diagrams	13
	3.2 Sequence Diagrams	14
	3.3 Collaboration Diagrams	15
	3.4 State Chart Diagrams	16-17
	3.5 Activity Diagrams	18
04.	Design	19-28
	4.1 Architecture	20-28
	4.2 Flow Chart	21-22
	4.3 Database Design/Models	23-24

S.No	Title	Pg.No
	4.4 Class Diagram	25
	4.5 Component Diagram	26
	4.6 Deployment Diagram	27-28
05.	Implementation	29-49
	5.1 Introduction to Technology	30-36
	5.2 Sample Code	37-43
	5.3 Screenshots	44-49
06.	Testing	50-54
	6.1 Introduction to Testing	51-52
	6.2 Sample Test Cases	53-54
07.	Conclusion	55
08.	Future Enhancements	56
09.	References	57

CHAPTER-01
INTRODUCTION

1.1 **Feasibility Study:**

- **Purpose:**

The purpose of this is to define that, In traditional college campus settings, managing student/faculty's movements during college hours poses significant challenges for administrators and security personnel. The absence of a simplified exit system often leads to inefficiencies, potential security breaches, and difficulties in monitoring student/faculty's presence.

The system aims to track the exit of individuals from the campus by requiring them to scan their QR code upon leaving. This ensures that campus security can monitor departures effectively.

The feasibility of the QR-Based Campus Exit Management System project can be evaluated based on several factors:

- **Technical Feasibility:**

- The chosen technologies (React.js for web app, Node.js for backend, MongoDB for database, React Native for mobile app) are widely used and well-supported, indicating technical feasibility.
- The use of REST APIs for communication between the web and mobile applications is a common practice, ensuring compatibility and interoperability.
- Local deployment of the system may suffice for initial development and testing but may require additional considerations for scalability and reliability in a production environment.

- **Operational Feasibility:**

- The system aims to streamline campus exit management by automating the process using QR codes, which could improve efficiency and reduce manual efforts.
- Integration with existing campus infrastructure, such as access control systems or student databases, may require careful planning and coordination to ensure seamless operation.
- Training for security personnel and administrative staff on using the mobile application and interpreting system data may be necessary to ensure effective implementation.

- **Economical Feasibility:**

- The project's financial feasibility depends on factors such as the cost of development, deployment, and maintenance.
- Open-source technologies like React.js, Node.js, and MongoDB offer cost-effective solutions for development.
- However, ongoing costs associated with the server hosting, maintenance, and potential future enhancements should be considered in the project's budget.

- **Legal and Ethical Feasibility:**

- Clear policies and procedures should be established to address data security, access control, and user privacy concerns.
- As it is own college management application & data storage, it is legally & Ethically feasible.

In conclusion, the QR-Based Campus Exit Management System appears technically feasible with the chosen technologies and operational concept, thorough consideration of operational, financial, legal, ethical, and it is necessary to ensure successful implementation and adoption. Conducting a detailed feasibility study encompassing these factors would provide a comprehensive assessment of the project's viability.

1.2 **Existing System:**

The existing system for campus exit management typically relies on manual methods such as sign-out sheets or physical tokens like ID cards. These methods are prone to errors, time-consuming, and lack real-time monitoring capabilities.

Here are the key drawbacks of the existing system:

- **Manual Sign-out Sheets:**

Students or faculty members leaving the campus are required to manually sign out in a logbook, which can be easily manipulated or forgotten.

- **Limited Tracking:**

With manual systems, it's difficult to track the exact time and location of each exit, leading to inefficiencies in campus security and management.

- **Dependency on Human Resources:**

The existing system heavily relies on security personnel or security personnel to manually verify the exit records, leading to potential errors and delays.

- **Bias in Time Entry:**

Differences in work habits or approaches among team members may lead to variations in time entry bias.

1.3 **Proposed System:**

The proposed QR-Based Campus Exit Management System aims to address the limitations of the existing system by introducing an automated and efficient solution.

Here's how the proposed system improves upon the existing system:

- **QR Code Technology:**

Each student and faculty member is assigned a unique QR code, which they scan using a mobile app while exiting the campus. This QR code contains essential information such as user ID, name, and timestamp.

- **Real-time Monitoring:**

The system provides real-time monitoring of campus exits, allowing administrators to track the entry and exit of individuals accurately.

- **Simple & Efficient :**

Simple to understand & Efficient support system for addressing user queries and issues.

- **Enhanced Security:**

The use of QR codes enhances campus security by providing a secure and tamper-proof method of identification.

- **Integration with Mobile App:**

The mobile app for scanning QR codes is user-friendly and can be easily deployed on security personnel's devices. It streamlines the exit process and enhances operational efficiency.

- **Centralized Database:**

All exit records are stored in a centralized MongoDB database, facilitating easy access and retrieval of data for administrative purposes.

- **Scalability:**

The system is scalable to accommodate future growth and can be easily integrated with existing campus management systems.

- **Accuracy:**

Accurate verification of user identity & generation of reports on campus exit activities.
It also measures data accuracy and consistency in reports.

CHAPTER-02
SOFTWARE REQUIREMENTS
SPECIALIZATION (SRS)

2.1 Introduction:

The QR-Based Campus Exit Management System aims to revolutionize the monitoring process of student and faculty exits from the campus premises. By leveraging modern technologies, including React.js for web application development and React Native for mobile application development, along with Node.js for the backend and MongoDB for database management, the system provides a comprehensive solution for tracking exits in real-time. The primary objective of this system is to enhance campus security, automate exit monitoring processes, and provide administrators with accurate data for effective decision-making.

2.2 External Interface Requirement:

- **Hardware Interfaces:**

- The system requires devices with camera functionality for QR code scanning, such as smartphones or tablets, to interact with the mobile application.
- A server environment is necessary to host the web application and backend services, equipped with adequate processing power and storage capacity.

- **Software Interfaces:**

- **Web Application:**

- The web application interface is accessible through standard web browsers such as Google Chrome, Mozilla Firefox, or Safari.

- **Mobile Application:**

- The mobile application interface is compatible with iOS and Android operating systems, accessible through smartphones or tablets.

- **Database:**

- The system interacts with the MongoDB database management system for storing and retrieving data.

- **User Interfaces:**

- **Web Application Interface:**

- The web interface provides administrators with functionalities for user management, monitoring exits, generating reports through graphs, and managing system settings.

- **Mobile Application Interface:**

- The mobile interface allows security personnel or security personnel to scan QR codes, view exit records, and entry statuses in real-time.

2.3 System Features (Functional Requirements):

- **User Registration and Management:**

- Administrators can register and manage student and faculty profiles, assigning unique QR codes to each individual.
- Security personnel can access the mobile application with designated credentials for scanning QR codes and updating exit statuses.

- **QR Code Generation:**

- The system generates unique QR codes for each registered user, containing essential identification information.

- **QR Code Scanning:**

- Security personnel use the mobile application to scan QR codes when students or faculty exit the campus premises.
- Scanned QR codes are validated against the database to verify user identity and record exit timestamps.

- **Real-time Monitoring:**

- Administrators can monitor exits in real-time through the web application, viewing updated exit records and statistics.

- **Database Management:**

- The system manages user profiles, exit records, and system settings in the MongoDB database, ensuring data integrity and reliability.

- **Reporting:**

- Administrators can analyse and can generate comprehensive reports on exit activities, including exit frequency, peak hours, and user trends, to facilitate decision-making and security planning.

2.4 **Other Non-Functional Requirements:**

- **Performance:**

- The system must exhibit high performance, with minimal latency in scanning QR codes and updating exit records.
- It should be capable of handling concurrent user interactions and large volumes of exit data efficiently.

- **Security:**

- The system must adhere to industry-standard security practices to protect user data and prevent unauthorized access.
- Admin authentication mechanisms should be robust, utilizing encryption and secure communication protocols for the users data.

- **Reliability:**

- The system should be reliable, with minimal downtime and data loss.
- Backup and recovery mechanisms should be in place to ensure data integrity and availability.

- **Scalability:**

- The system should be scalable to accommodate future growth in user base and exit monitoring requirements.
- It should support horizontal scaling of server resources and database capacity expansion.

- **Usability:**

- The user interfaces of both the web and mobile applications should be intuitive and user-friendly, requiring minimal training for administrators and security personnel.

- **Accessibility:**

- The system should be accessible to users with disabilities, complying with accessibility standards and guidelines.

- **Compatibility:**

- The web and mobile applications should be compatible with a wide range of devices and screen sizes, ensuring seamless user experience across different platforms.

- **Documentation:**

- Comprehensive documentation, including user manuals and system architecture guides, should be provided to facilitate system understanding, maintenance, and troubleshooting.

- **Deployment:**

- The system should support easy deployment on local servers, cloud platforms, or hybrid environments, with minimal configuration requirements.

Thus, Software Requirements Specification (SRS) outlines the introduction, external interface requirements, functional requirements, and non-functional requirements of the QR-Based Campus Exit Management System in detail.

CHAPTER - 03

ANALYSIS

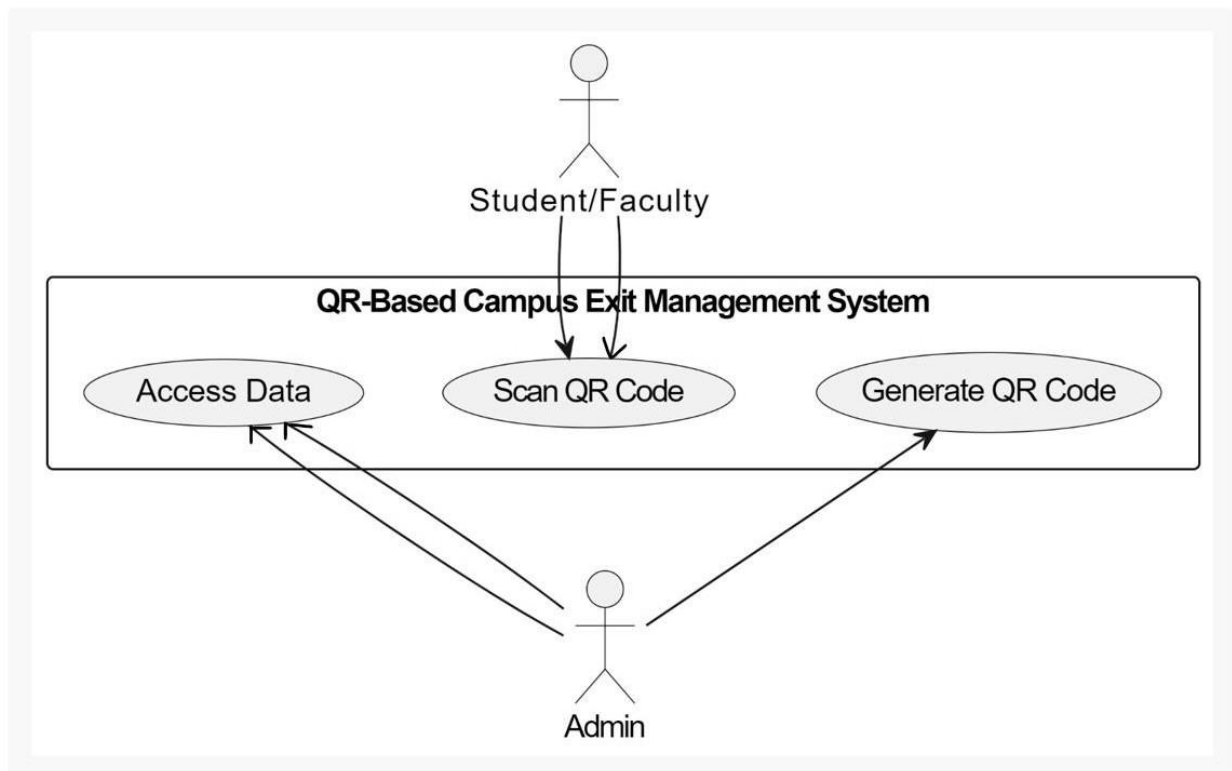
3.1 Use Case:

Use cases describe interactions between users (actors) and the system to accomplish specific goals.

Explanation:

In the context of the QR-Based Campus Exit Management System, use cases could include:

- **Scan QR Code:** Student/Faculty scans their QR code when leaving the campus.
- **Verify Exit:** Security personnel scans the QR code to verify the exit.
- **View Exit History:** Admin can view the exit history of students/faculty.
- **Benefits:** Use cases help in understanding the system's functionality from a user's perspective and defining the system's requirements clearly.



USE CASE DIAGRAM

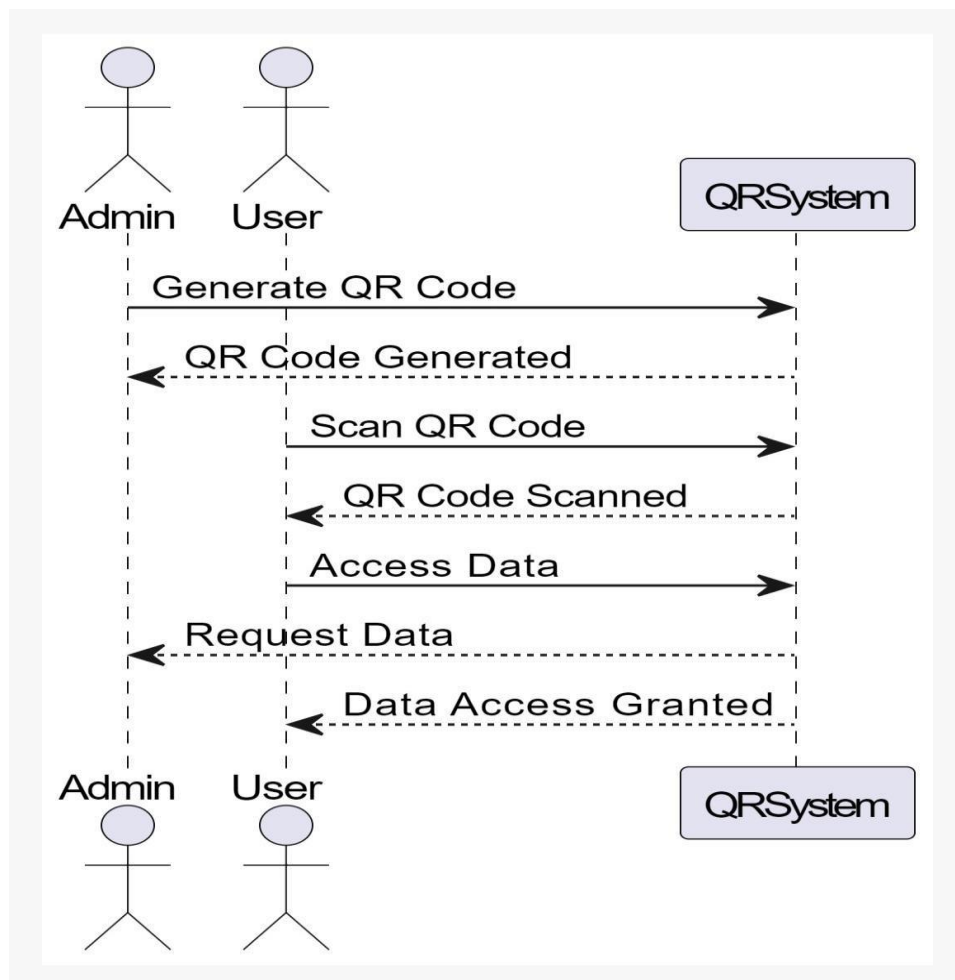
3.2 Sequence Diagram:

Sequence diagrams illustrate the interactions between objects or components in a system over time.

Explanation:

Sequence diagrams for the system could depict:

- **Student/Faculty Exit Sequence:** The sequence of actions between the user and the system when exiting the campus, including QR code scanning and verification.
- **Security personnel Verification Sequence:** The sequence of actions between the security personnel's mobile app and the backend server when verifying the QR code.
- **Benefits:** Sequence diagrams provide a visual representation of the system interactions, helping to understand the flow of operations and identify potential bottlenecks.



SEQUENCE DIAGRAM

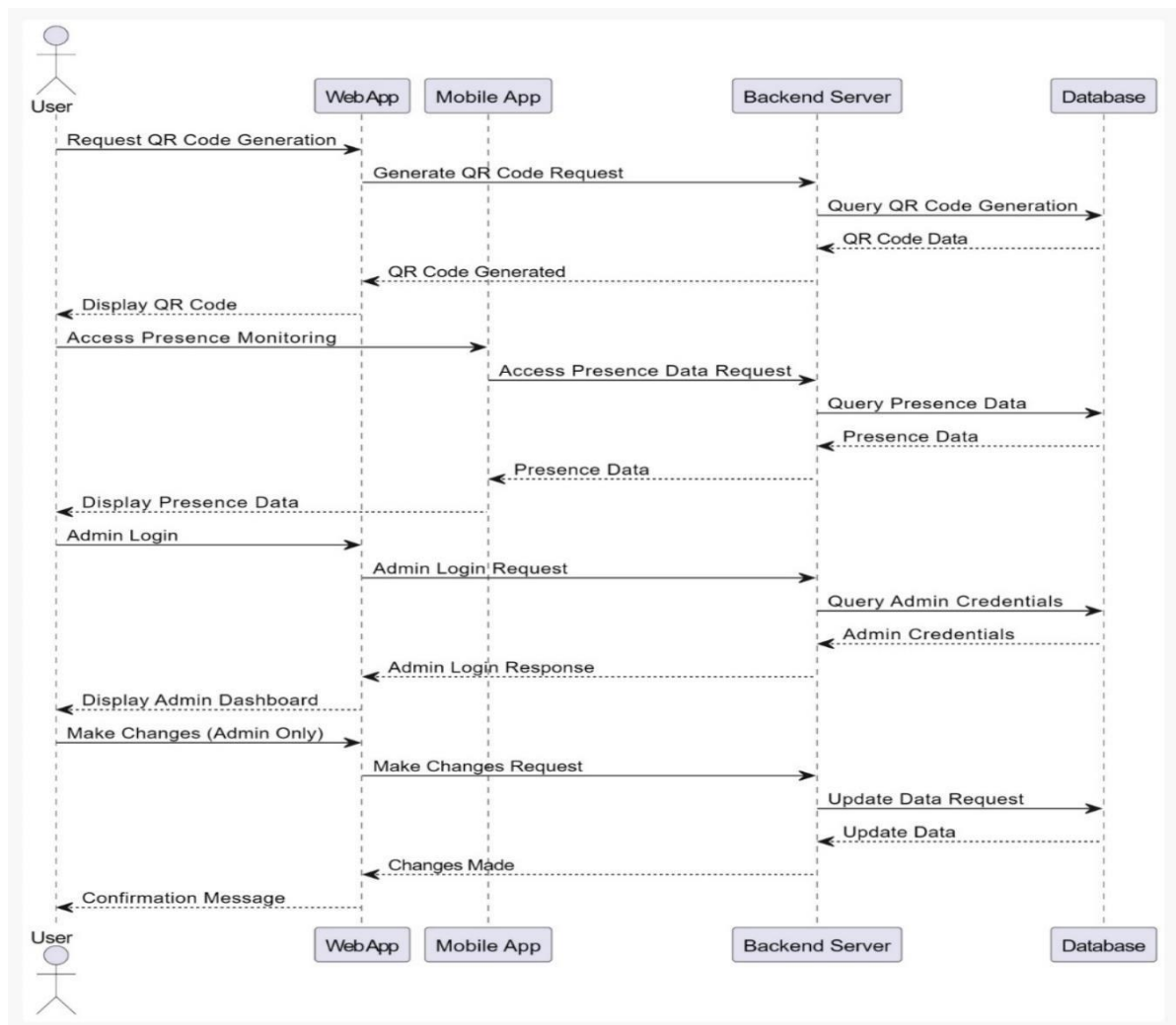
3.3 Collaboration Diagram:

Collaboration diagrams, also known as communication diagrams, illustrate the interactions between objects or components within a system.

Explanation:

Collaboration diagrams for the QR-Based Campus Exit Management System could show:

- **Interactions between Components:** Communication between frontend components (web app), backend components (Node.js server), and mobile app components (React Native).
- **Data Flow:** Exchange of data between the system's components during QR code generation, scanning, and verification processes.
- **Benefits:** Collaboration diagrams provide insights into how different system components collaborate to achieve specific functionalities, aiding in system design and implementation.



COLLABORATION DIAGRAM

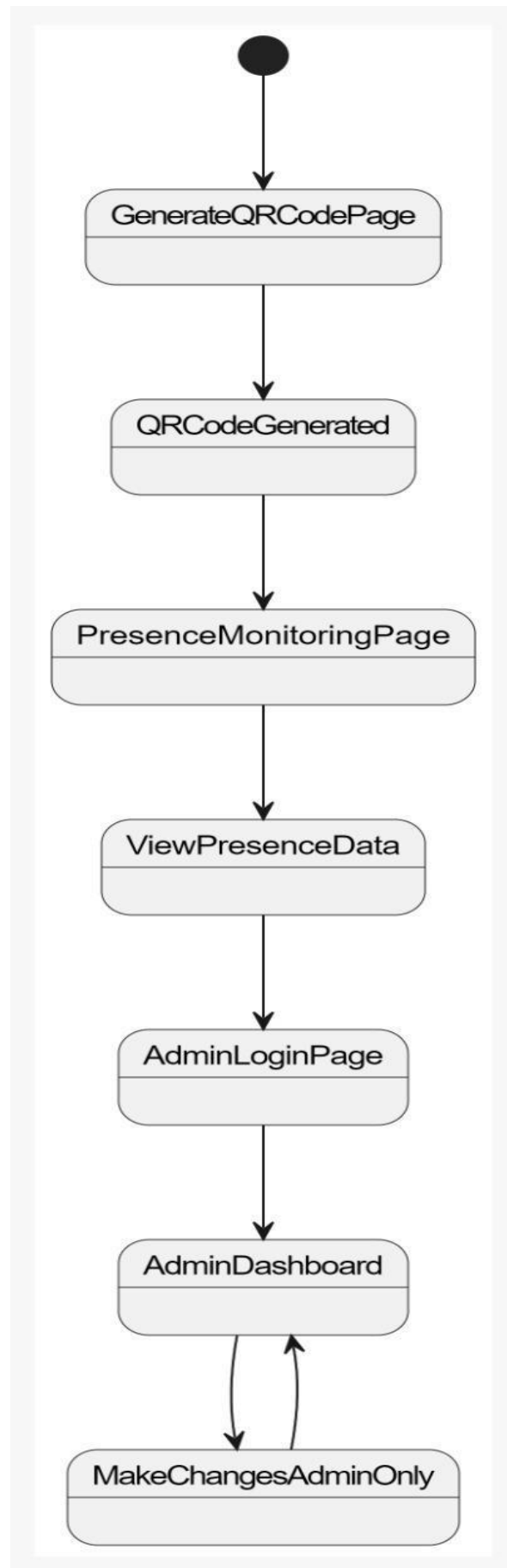
3.4 State Chart Diagram:

State chart diagrams depict the various states that an object or component can go through during its lifecycle and the transitions between these states.

Explanation:

State chart diagrams for the QR-Based Campus Exit Management System could illustrate:

- **QR Code State:** The states of a QR code (e.g., Valid, Invalid, Scanned).
- **User State:** The states of a user (e.g., Logged In, Logged Out).
- **Security personnel App State:** The states of the security personnel's mobile app (e.g., Idle, Scanning, Verifying).
- **Benefits:** State chart diagrams provide a clear understanding of the system's behaviour in different states, aiding in designing robust and efficient system logic.



STATE CHART DIAGRAM

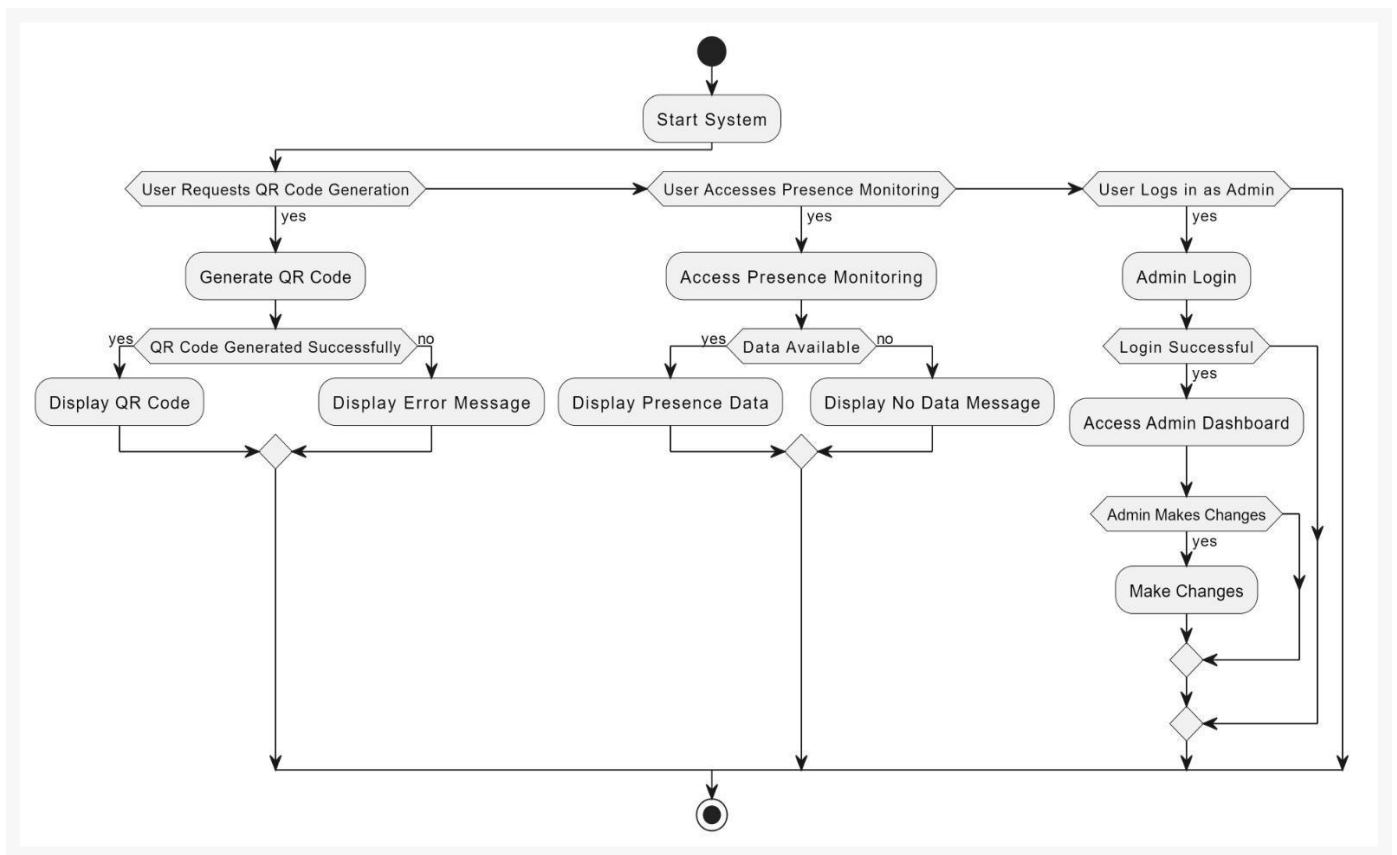
3.5 Activity Diagram:

Activity diagrams depict the flow of activities or actions within a system or process.

Explanation:

Activity diagrams for the QR-Based Campus Exit Management System could represent:

- **User Exit Process:** The sequence of activities involved when a user exits the campus, including QR code scanning and verification.
- **Security personnel Verification Process:** The activities performed by the security personnel when verifying the QR code.
- **Benefits:** Activity diagrams provide a detailed view of the system's workflow, helping to identify potential inefficiencies or areas for improvement.



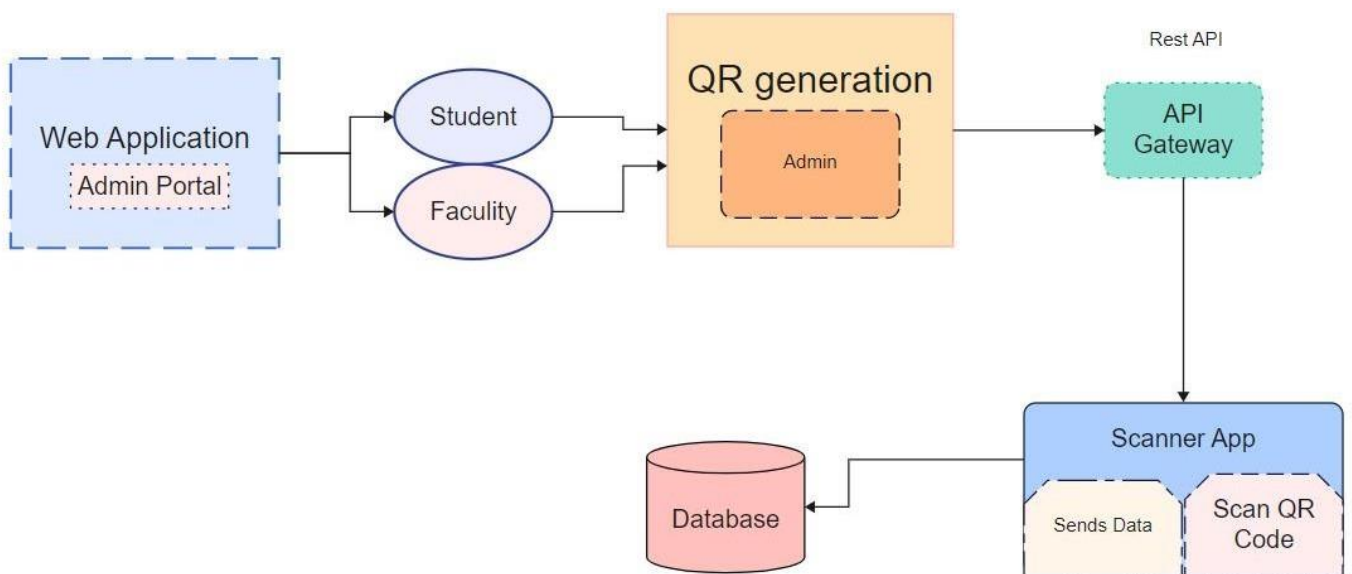
ACTIVITY DIAGRAM

CHAPTER - 05
DESIGN

4.1 Architecture:

The architecture of your system likely follows a client-server model, where clients interact with the server to generate and scan QR codes. Here's a basic overview:

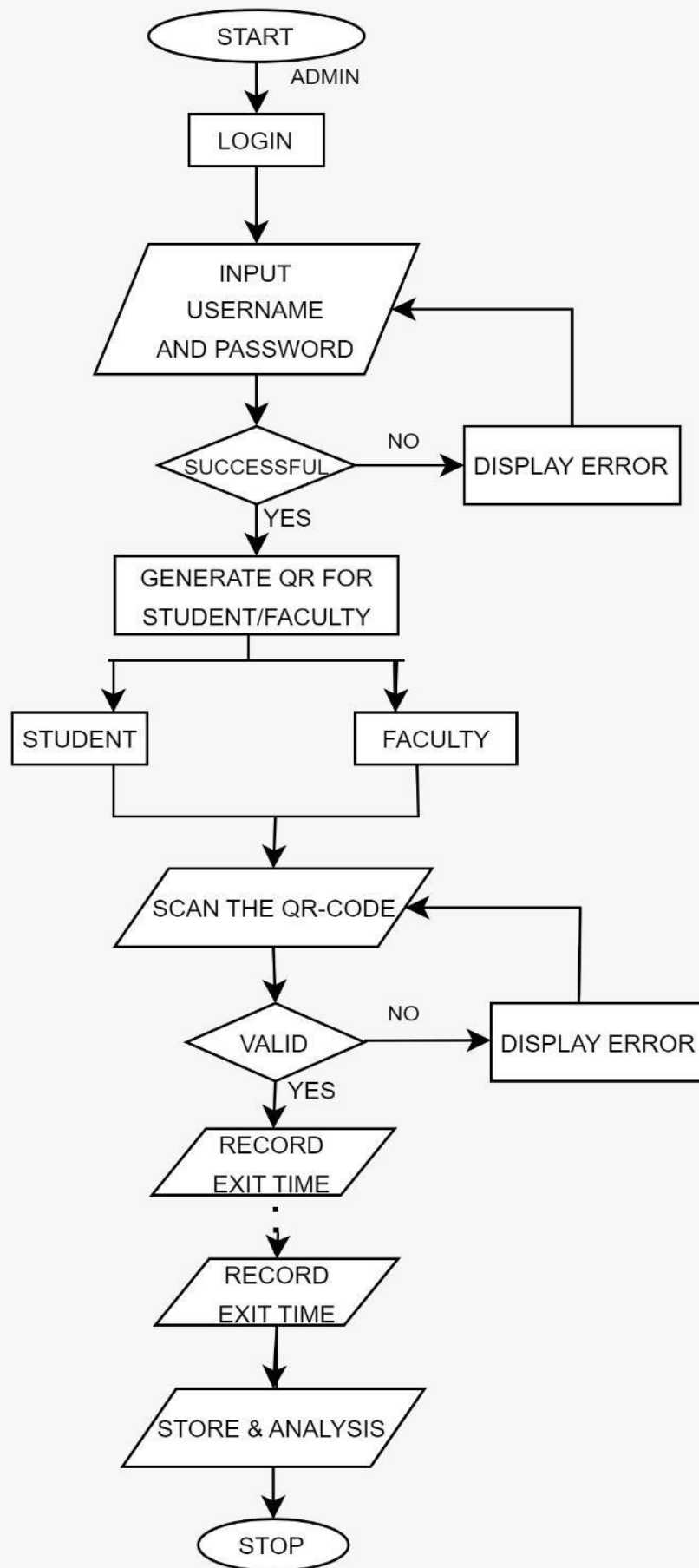
- **Client (Web):** ReactJS for frontend. Generates QR codes and sends requests to the server for data.
- **Server:** Node.js for backend. Receives requests from clients, interacts with the database, and sends responses back.
- **Database:** MongoDB. Stores information about students, faculty, QR codes, and exit logs.
- **Mobile App:** React Native. Scans QR codes and sends scanned data to the server for processing.



ARCHITECTURE

4.2 Flow Chart:

1. **User Registration:** Students and faculty register with their details.
2. **QR Code Generation:** The server generates unique QR codes for each registered user.
3. **Exit Management:**
 - Users scan their QR code when exiting the campus.
 - The mobile app scans the QR code and sends the data to the server.
 - The server records the exit event in the database.



FLOW CHART

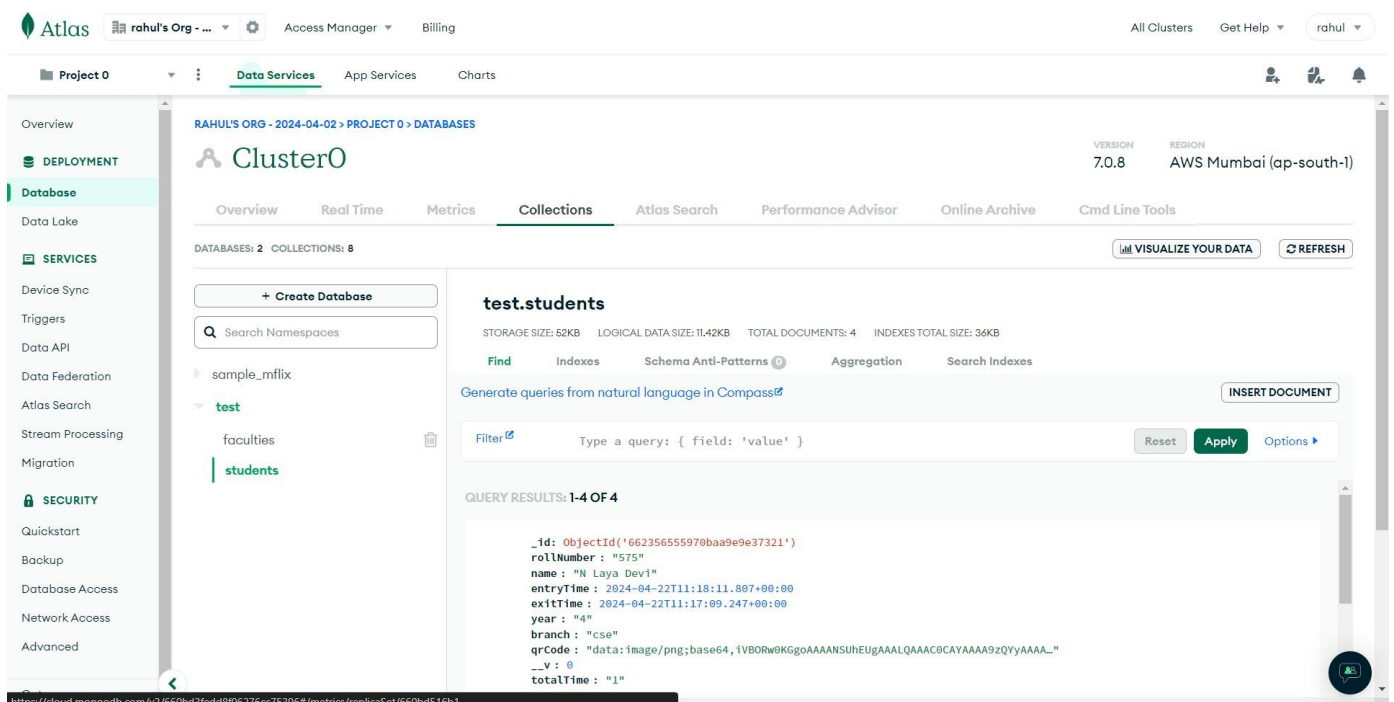
4.3 Database Design/Model:

In Database, we've stored the information about users (both faculty and students), their QR codes, and exit logs. The data will be stored in the collections(clusters) of the database in Json format.

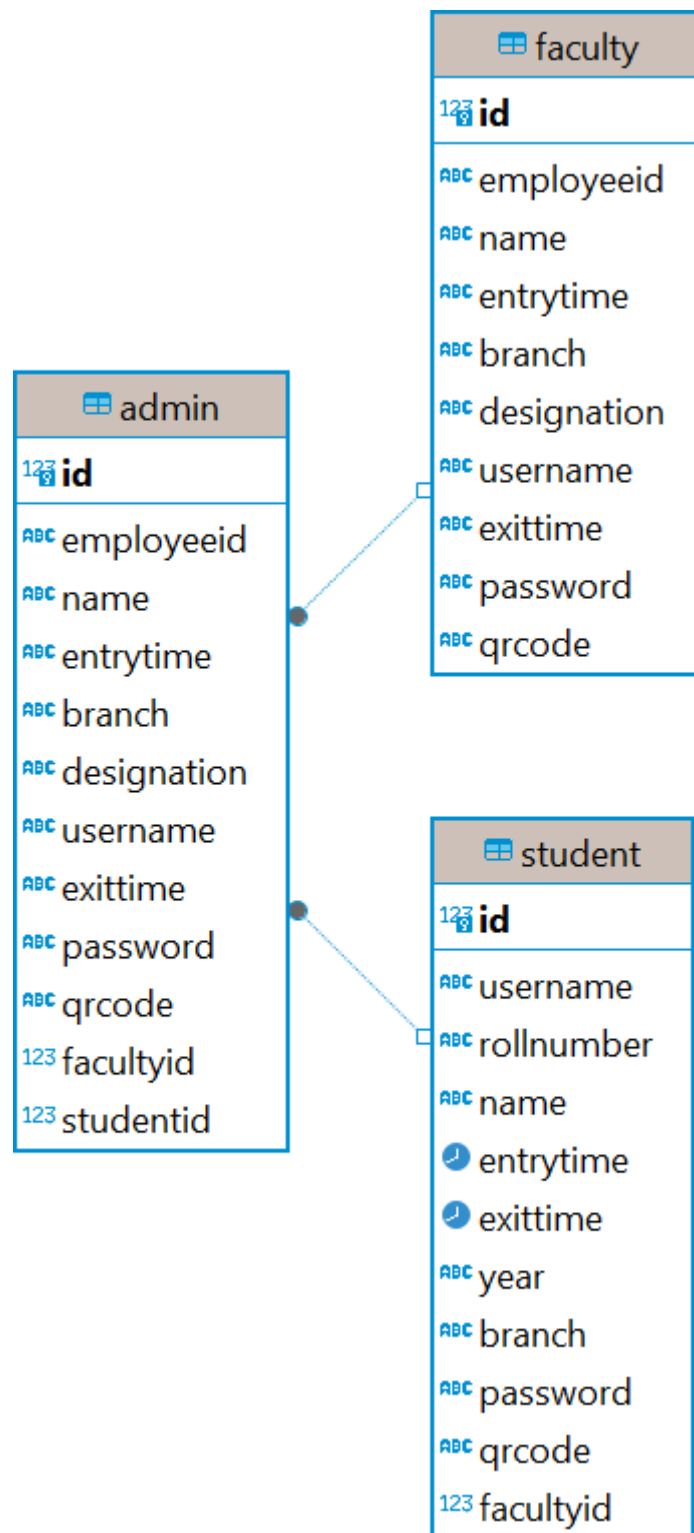
Here's the basic model of the MongoDB database:

Collection: Users

- `_id`: ObjectId (Automatically generated unique identifier)
- `username`: String (Username for login)
- `password`: String (Hashed password for security) – only for admin
- `role`: String (Admin, Faculty, Student)
- `profile`: Object (Details of the user)
- `name`: String (Name of the user)
- `id`: String (Unique ID of the user)
- `department`: String (Department of the user)
- other relevant user details such as Timings



FORMAT OF THE DATABASE (Representing NoSQL)



SCHEMA OF THE DATABASE

4.4 Class Diagram:

- **Admin Class:**

Represents the administrator of the system, who plays a typical role in the Qr-generation & monitoring of the student/faculty

- **User Class:**

Represents users of the system. It includes attributes like ID, username, password, role, and profile. Each user can have one associated QR code and multiple exit logs.

- **QR-Code Class:**

Represents QR codes associated with users. It contains an ID, the ID of the user it belongs to, the QR code itself, and the creation timestamp.

- **Mobile App Class:**

This is used for the scanning of the user's data & timings & to validating the details which are scanned.

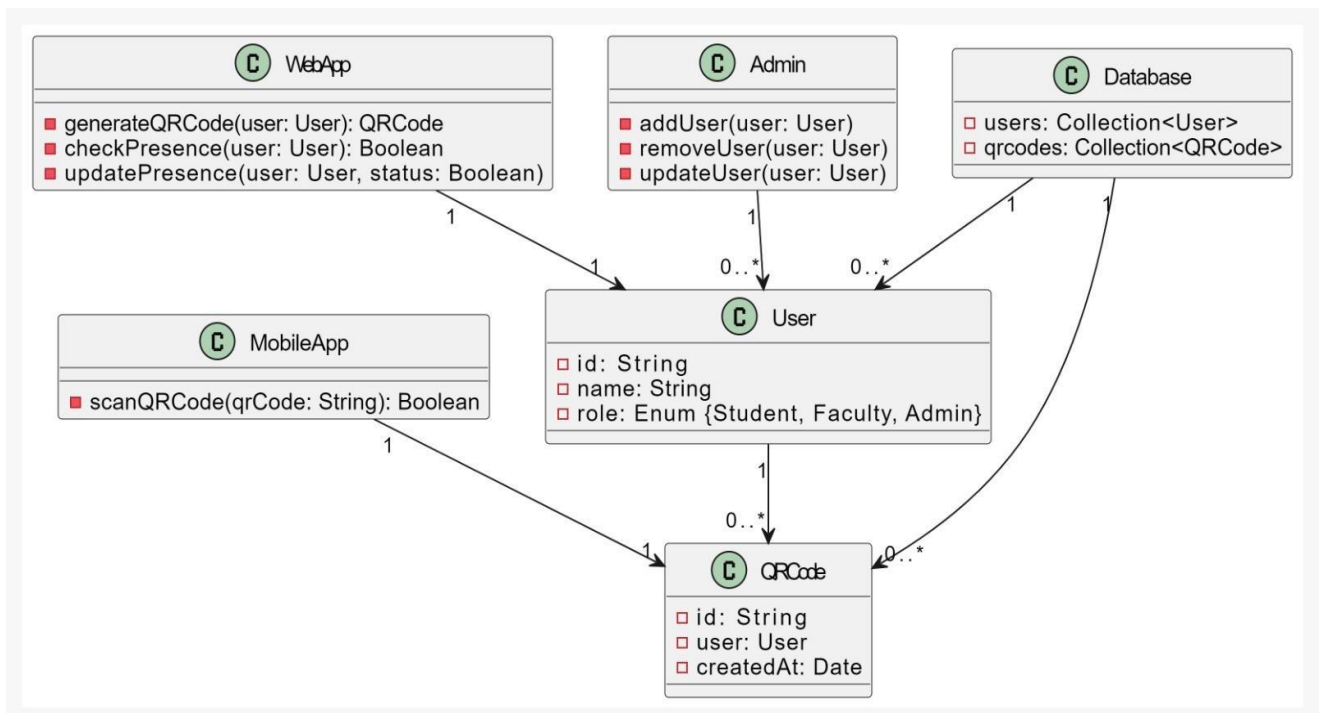
- **Database Class:**

Database class is used to store the exit & entry logs of the users during the working times of the Educational Institutions. It includes an ID, the ID of the user who exited, and the timestamp of the exit.

- **Web App Class:**

It is used to represent the storage of the users in the database including the exit & entry logs for users. It is also used to monitor the student/faculty's presence & to create analytics & reports.

This design allows for efficient management of users, their associated QR codes, and exit logs while maintaining a clear relationship between the entities. The admin can generate QR codes for users, monitor exits, and access data as required.



CLASS DIAGRAM

4.5 Component Diagram:

The component diagram represents the different components/modules of your system and how they interact with each other.

Explanation:

1. Web Client:

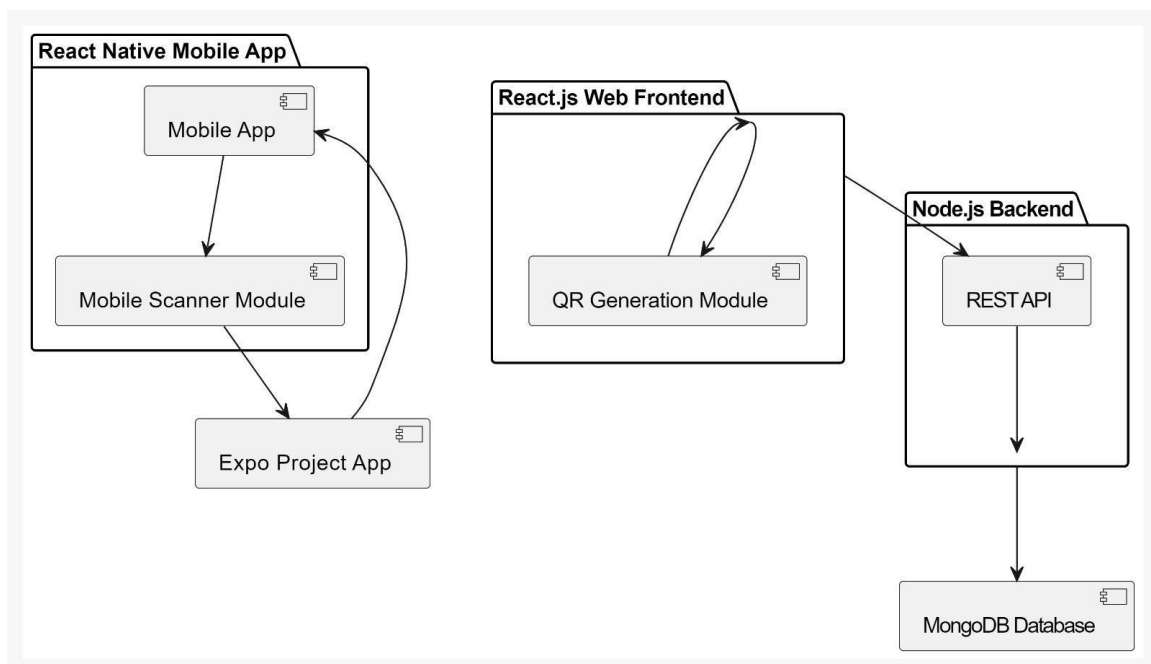
- Responsible for generating QR codes, displaying user interfaces, and sending requests to the backend.
- Uses ReactJS for QR generation and frontend UI.

2. Node.js Backend:

- Handles authentication, API endpoints, database interaction, and QR code generation.
- Uses Node.js for backend logic and MongoDB for data storage.

3. Mobile App:

- Scans QR codes, sends requests to the backend, and displays user interfaces.
- Developed using React Native for cross-platform compatibility.



COMPONENT DIAGRAM

4.6 **DEPLOYMENT:**

The deployment diagram illustrates how your system is deployed across different servers or environments.

Explanation:

1. Web Server:

- Hosts the web application built with ReactJS.
- Serves HTML/CSS/JS files for the frontend.
- Handles QR generation and user interfaces.

2. Node.js Server:

- Runs the Node.js backend.
- Utilizes Express.js for handling HTTP requests and API endpoints.
- Interacts with MongoDB for data storage.

3. Web Client:

- ReactJS is used for its ease of component-based UI development.
- QR codes are generated dynamically on the client side.
- User interfaces are rendered based on the user's role (admin, faculty, student).

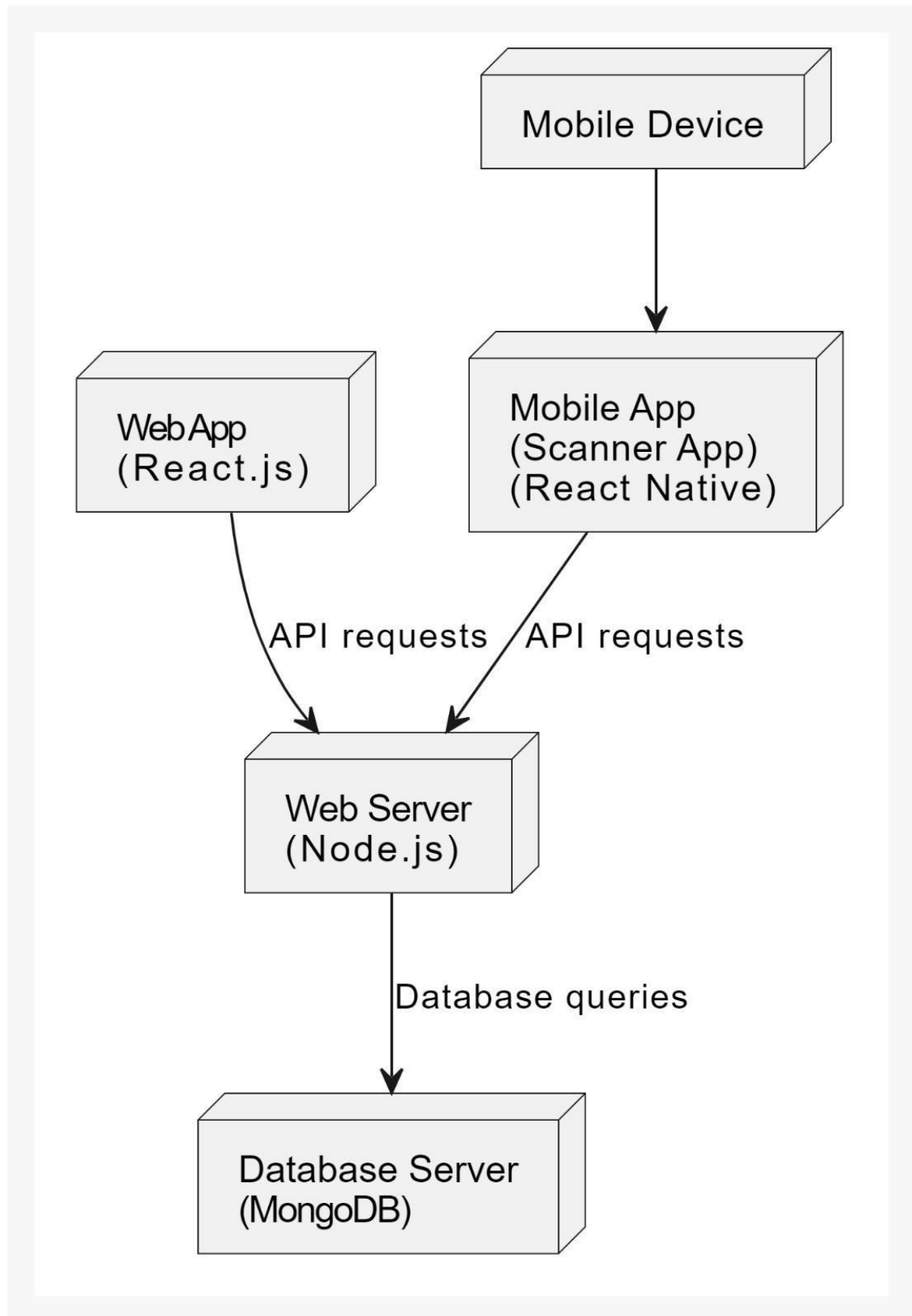
4. Node.js Backend:

- Provides RESTful APIs for communication between the frontend and the database.
- MongoDB is chosen for its flexibility and scalability.
- Authentication ensures only admins can access certain functionalities.

5. Mobile App:

- React Native offers a smooth and efficient development experience for cross-platform mobile apps.
- Expo Client simplifies the deployment and testing process for React Native apps.
- The QR scanner utilizes the device's camera to scan QR codes.

This setup ensures a seamless and efficient QR-based campus exit management system, with clear separation of concerns between frontend, backend, and mobile app components.



DEPLOYMENT DIAGRAM

CHAPTER-05

IMPLEMENTATION

5.1 Introduction to Technology:

The implementation of the QR-Based Campus Exit Management System involves the integration of various technologies to achieve the desired functionalities.

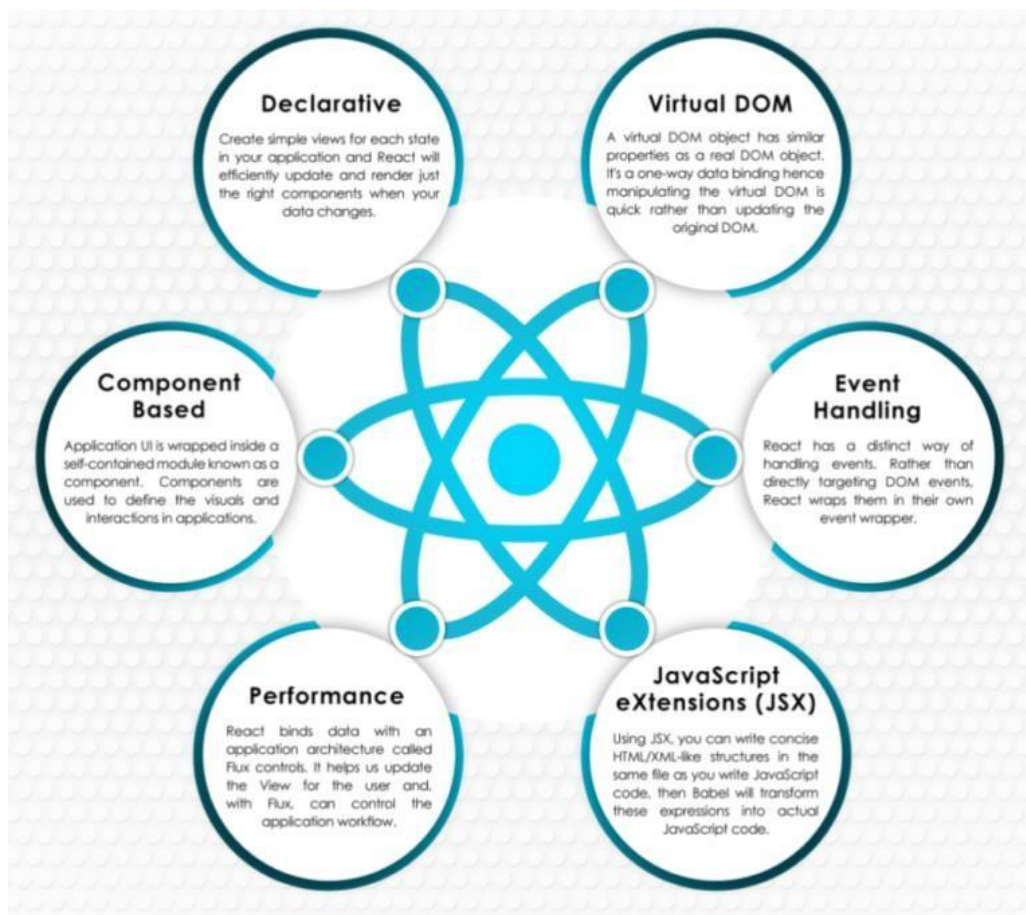
Here is an overview of the technologies used in the implementation process:

- **React.js for Frontend Development**
- **Node.js for Backend Development**
- **MongoDB Database**
- **RESTful APIs**
- **React Native for Mobile App Development**
- **Deployment**
 - **Netlify – for the web App**
 - **Local Hosting – for Mobile App**

Thus, the implementation of the QR-Based Campus Exit Management System involved the strategic utilization of React.js, Node.js, MongoDB, RESTful APIs, and React Native technologies to create a seamless and efficient solution for monitoring student and faculty exits from the campus premises.

➤ **React.js for Frontend Development:**

- React.js is a JavaScript library widely used for building user interfaces, particularly single-page applications.
- ReactJS provides developers with a powerful and efficient way to build interactive and dynamic user interfaces for web applications with the help of Material UI library.
- In our project, React.js was employed for frontend development, specifically for QR code generation and monitoring the presence of students/faculty on the web application.
- React.js provides a component-based architecture, making it easy to develop and maintain complex user interfaces efficiently.



REACT.JS & IT'S USES

➤ **Node.js for Backend Development:**

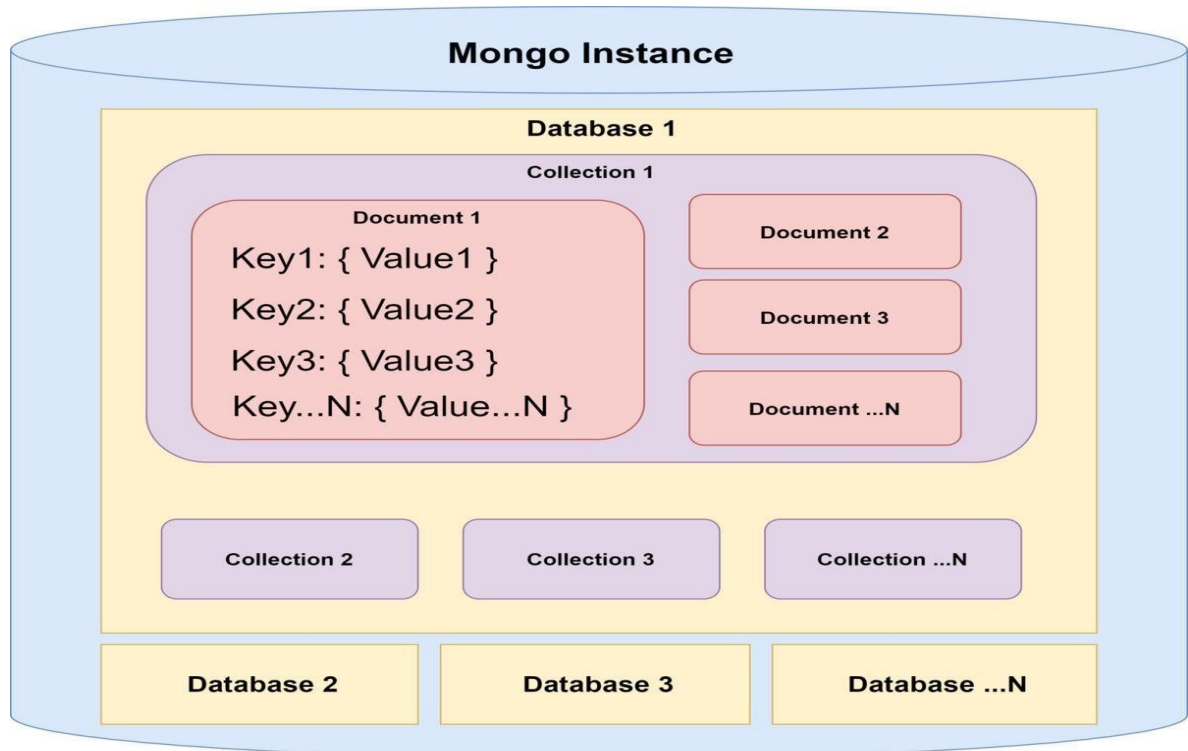
- Node.js is a server-side JavaScript runtime environment known for its non-blocking I/O model and event-driven architecture.
- We utilized Node.js for backend development to handle server-side logic, manage database operations, and serve RESTful APIs to the frontend.
- Its lightweight and scalable nature makes it suitable for building fast and efficient backend systems.
- Node.js serves as the backbone of our project, handling the server-side logic, API development, and database interactions necessary for the QR-Based Campus Exit Management System to function effectively.



NODE.JS

➤ MongoDB Database:

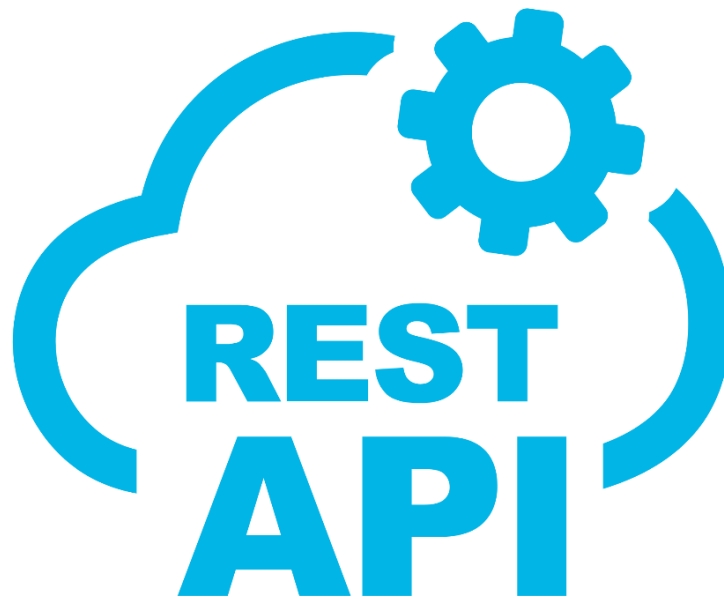
- MongoDB is a NoSQL database management system that stores data in flexible, JSON-like documents.
- In our project, MongoDB was chosen as the database solution due to its schema-less nature, which allows for easy integration with Node.js and flexibility in handling varied data types.
- The database stores user profiles, QR code data, exit records, and other relevant information required for the system's functionality.



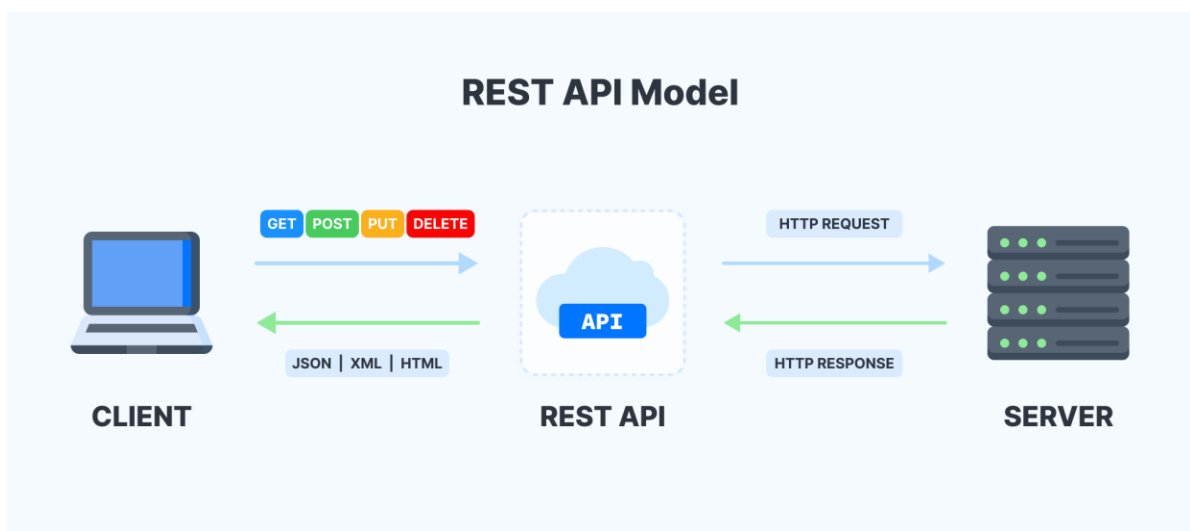
MONGODB DATABASE & IT'S STRUCTURE

➤ RESTful APIs:

- RESTful APIs (Representational State Transfer) are used to enable communication between the frontend and backend components of the system.
- These APIs define standardized endpoints and methods for performing CRUD (Create, Read, Update, Delete) operations on resources such as user profiles, exit records, and system settings.
- Node.js, was utilized to develop and manage these RESTful APIs, providing a robust foundation for handling HTTP requests and responses.



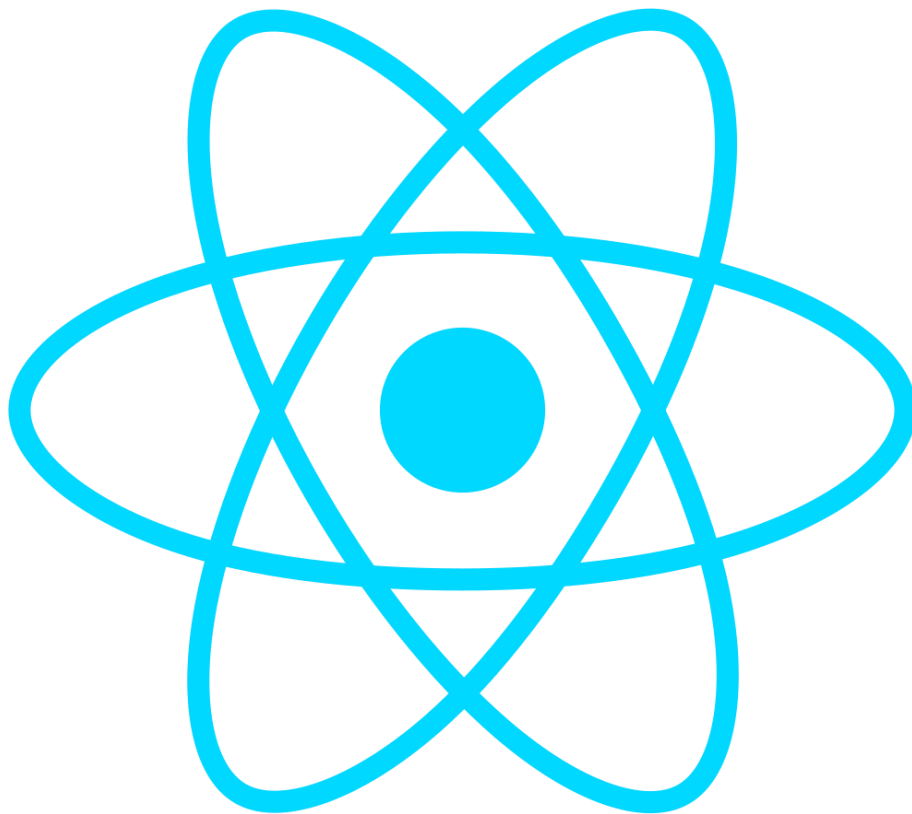
RESTful APIs



WORKING OF REST APIs

➤ **React Native for Mobile App Development:**

- React Native is a framework for building native mobile applications using JavaScript and React.
- Our project utilized React Native for developing the mobile scanner application used by security personnel to scan QR codes.
- By leveraging React Native, we were able to write cross-platform mobile code that runs on both iOS and Android devices, minimizing development efforts and ensuring consistent user experiences across platforms.
- React Native offers a robust set of developer tools, including debugging tools, performance monitoring, and testing frameworks, which aid in building, debugging, and maintaining apps efficiently.



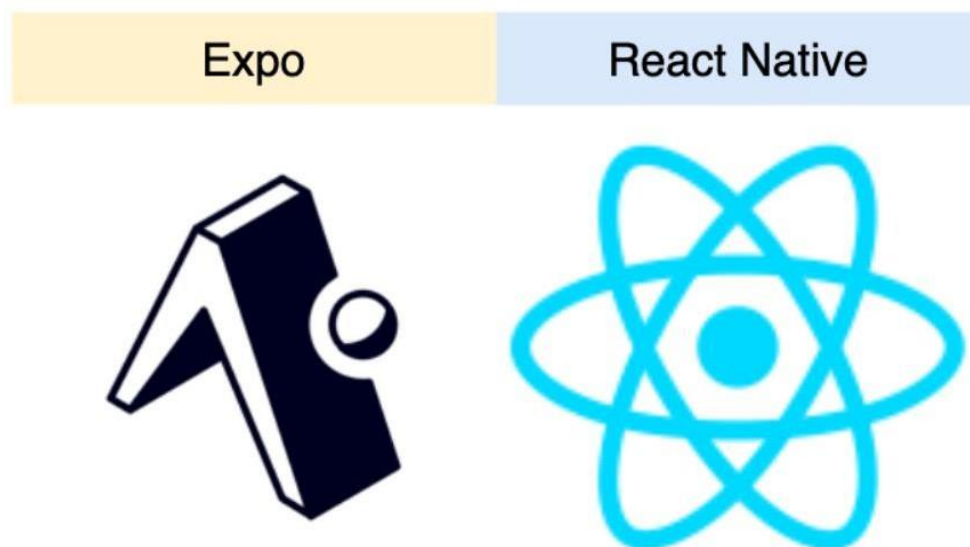
React Native

➤ **Deployment:**

- Deployment of the web application and backend components was done locally by running the server on a designated machine within the network & deployed in Netlify app as well.
- Similarly, the mobile scanner application was deployed locally using the Project-Expo app, which facilitates the development and testing of React Native applications on physical devices.



NETLIFY APP – FOR WEB APPLICATION



PROJECT EXPO – TO RUN THE MOBILE APPLICATION

5.2 Sample code:

WEB – APP IMPLEMENTATION

```
import { useSelector } from "react-redux";

import { ThemeProvider } from "@mui/material/styles";
import { CssBaseline, StyledEngineProvider } from "@mui/material";
import Routes from "routes";
import themes from "themes";
import NavigationScroll from "layout/NavigationScroll";
import Login from "views/pages/authentication/authentication3/Login3";
import { MyContext } from "store/useContext";
import { useContext, useEffect, useState } from "react";
import { useNavigate } from "react-router";
import { ToastContainer, toast } from "react-toastify";
import "react-toastify/dist/ReactToastify.css";

const App = () => {
  const [login, setLogin] = useState(false);
  const [facultyData, setFacultyData] = useState([]);
  const [studentData, setStudentData] = useState([]);

  const handleGetApi = () => {
    const studentFetch = fetch(
      "https://student-monitoring-backend.onrender.com/api/students/getAllStudents"
    )
      .then((response) => {
        if (!response.ok) {
          throw new Error("Network response was not ok");
        }
        return response.json();
      })
      .catch((error) => {
        console.error("Error fetching student data:", error);
        throw error; // Re-throw the error to be caught by Promise.all()
      });

    const facultyFetch = fetch(
      "https://student-monitoring-backend.onrender.com/api/faculty/getAllFaculty"
    )
      .then((response) => {
        if (!response.ok) {
          throw new Error("Network response was not ok");
        }
        return response.json();
      })
      .catch((error) => {
        console.error("Error fetching faculty data:", error);
        throw error; // Re-throw the error to be caught by Promise.all()
      });

    Promise.all([studentFetch, facultyFetch])
      .then(([studentData, facultyData]) => {
        // Both requests succeeded, update the state with the data
      });
  };
};
```

```

        setStudentData(studentData);
        setFacultyData(facultyData);
    })
    .catch((error) => {
        // At least one request failed, handle the error here
        console.error("Error fetching data:", error);
    });
};

useEffect(() => {
    handleGetApi();
    const localStorageLogin = localStorage.getItem("login");
    if (localStorageLogin) {
        setLogin(true);
    }
}, []);

useEffect(() => {
    if (login === true) {
        localStorage.setItem("login", "true");
    } else {
        localStorage.removeItem("login");
    }
}, []);

const customization = useSelector((state) => state.customization);

return (
    <StyledEngineProvider injectFirst>
        <ThemeProvider theme={themes(customization)}>
            <CssBaseline />
            <NavigationScroll>
                <MyContext.Provider
                    value={{
                        login,
                        setLogin,
                        facultyData,
                        studentData,
                        setFacultyData,
                        setStudentData,
                    }}
                >
                    {(!login && <Login setLogin={setLogin} />) || <Routes />}
                </MyContext.Provider>
                <ToastContainer />
            </NavigationScroll>
        </ThemeProvider>
    </StyledEngineProvider>
);
};

export default App;

```

MOBILE – APP IMPLEMENTATION

```
import React from "react";
import { NavigationContainer } from "@react-navigation/native";
import { createStackNavigator } from "@react-navigation/stack";

import LoginScreen from "../screens/LoginScreen";
import RegisterScreen from "../screens/RegisterScreen";
import HomeScreen from "../screens/HomeScreen";

const Stack = createStackNavigator();

const App = () => {
  return (
    <NavigationContainer>
      <Stack.Navigator initialRouteName="Login">
        <Stack.Screen name="Login" component={LoginScreen} />
        { /* <Stack.Screen name="Register" component={RegisterScreen} /> */ }
        <Stack.Screen name="Home" component={HomeScreen} />
      </Stack.Navigator>
    </NavigationContainer>
  );
};

export default App;
```

API'S IMPLEMENTATION FOR QR-CONTROLLERS

```
const qr = require("qrcode");
const { Buffer } = require("buffer");
const Student = require("../models/Student");
const Faculty = require("../models/Faculty");

const generateQRCode = async (data) => {
  try {
    const qrCodeDataUri = await qr.toDataURL(JSON.stringify(data));
    return qrCodeDataUri;
  } catch (error) {
    console.error("Error generating QR code:", error);
    throw new Error("Error generating QR code");
  }
};

module.exports = {
  scanQR: async (req, res) => {},
  generateNewQr: async (req, res) => {
    try {
      const {
        userType,
        rollNumber,
        emplyoeId,
        branch,
        name,
        year,
        designation,
        entryTime,
        exitTime,
      } = req.body;

      if (!userType) {
        return res.status(400).json({
          success: false,
          message: "User type required",
        });
      }

      const data = {
        userType,
        rollNumber,
        emplyoeId,
        branch,
        name,
        year,
        designation,
        entryTime,
        exitTime,
      };
    }
  }
};
```

```

// Generate QR code
const qrCodeDataUri = await generateQRCode(data);

let user;

// Check userType and update/create user
if (userType === "student") {
  let existingUser = await Student.findOneAndUpdate(
    { rollNumber: rollNumber },
    { $set: { qrCode: qrCodeDataUri } },
    { new: true }
  );
  user = existingUser;
} else if (userType === "faculty") {
  let existingUser = await Faculty.findOneAndUpdate(
    { employeeId: employeeId },
    { $set: { qrCode: qrCodeDataUri } },
    { new: true }
  );
  user = existingUser;
} else {
  return res.status(400).json({
    success: false,
    message: "Invalid user type",
  });
}

if (!user) {
  return res.status(404).json({
    success: false,
    message: "User not found",
  });
}

res.status(200).json({
  success: true,
  qrCodeDataUri: qrCodeDataUri,
  user: user,
});
} catch (error) {
  console.error("Error generating QR code:", error);
  res.status(500).json({
    success: false,
    message: "Error generating QR code",
  });
}
},

// POST endpoint for faculty data
scanFaculty: async (req, res) => {
  try {
    // Extract data from request body
    const { employeeId, name, branch, designation, entryTime, exitTime } =
      req.body.faculty;

```



```

// Create new faculty object
const newFaculty = new Faculty({
  employeeId,
  name,
  branch,
  designation,
  entryTime,
  exitTime,
});

// Save faculty data to the database
const savedFaculty = await newFaculty.save();

// Send success response
res.status(201).json({
  message: "Faculty data saved successfully",
  faculty: savedFaculty,
});
} catch (error) {
  // Send error response
  res
    .status(500)
    .json({ message: "Internal server error", error: error.message });
}
},
generateQRCode: generateQRCode,
};

```

DATABASE SCHEMA FOR STUDENTS

```
const { mongoose } = require("mongoose");

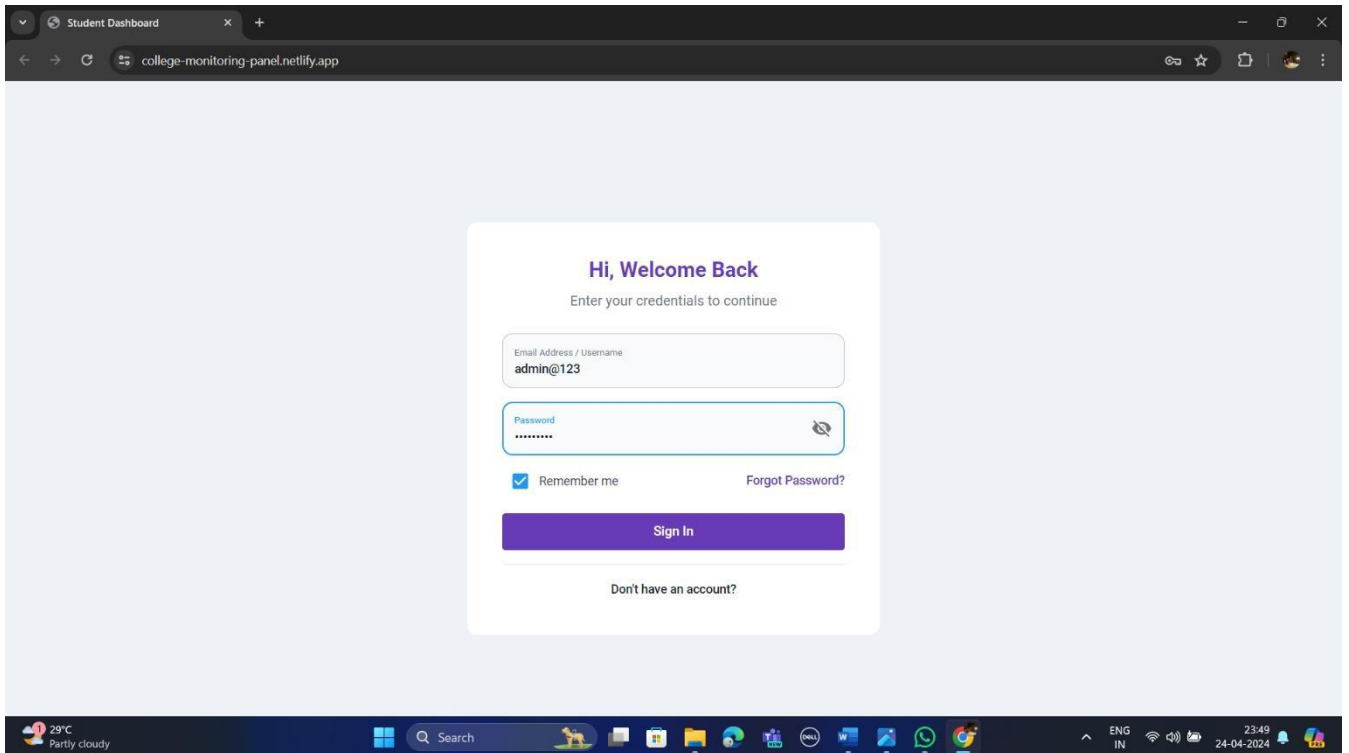
const studentSchema = new mongoose.Schema({
  userName: {
    type: String,
  },
  rollNumber: {
    type: String,
    required: true,
  },
  name: {
    type: String,
    required: true,
  },
  exit Time: {
    type: Date,
    default: Date.now,
  },
  entry Time: {
    type: Date,
    default: Date.now,
  },
  year: {
    type: String,
    required: true,
  },
  branch: {
    type: String,
    required: true,
  },
  password: {
    type: String,
  },
  qrCode: {
    type: String,
  },
});

const Student = mongoose.model("Student", studentSchema);

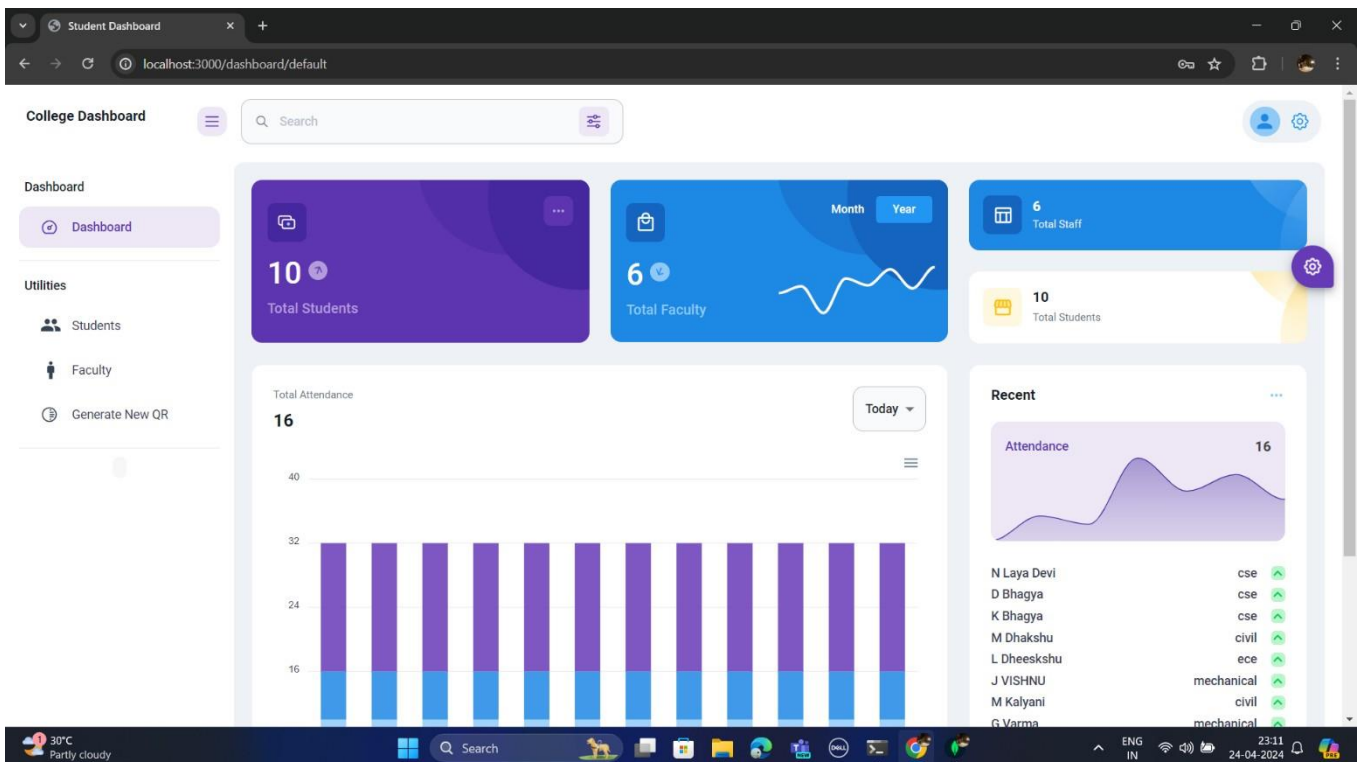
module.exports = Student;
```

5.3 Snapshots:

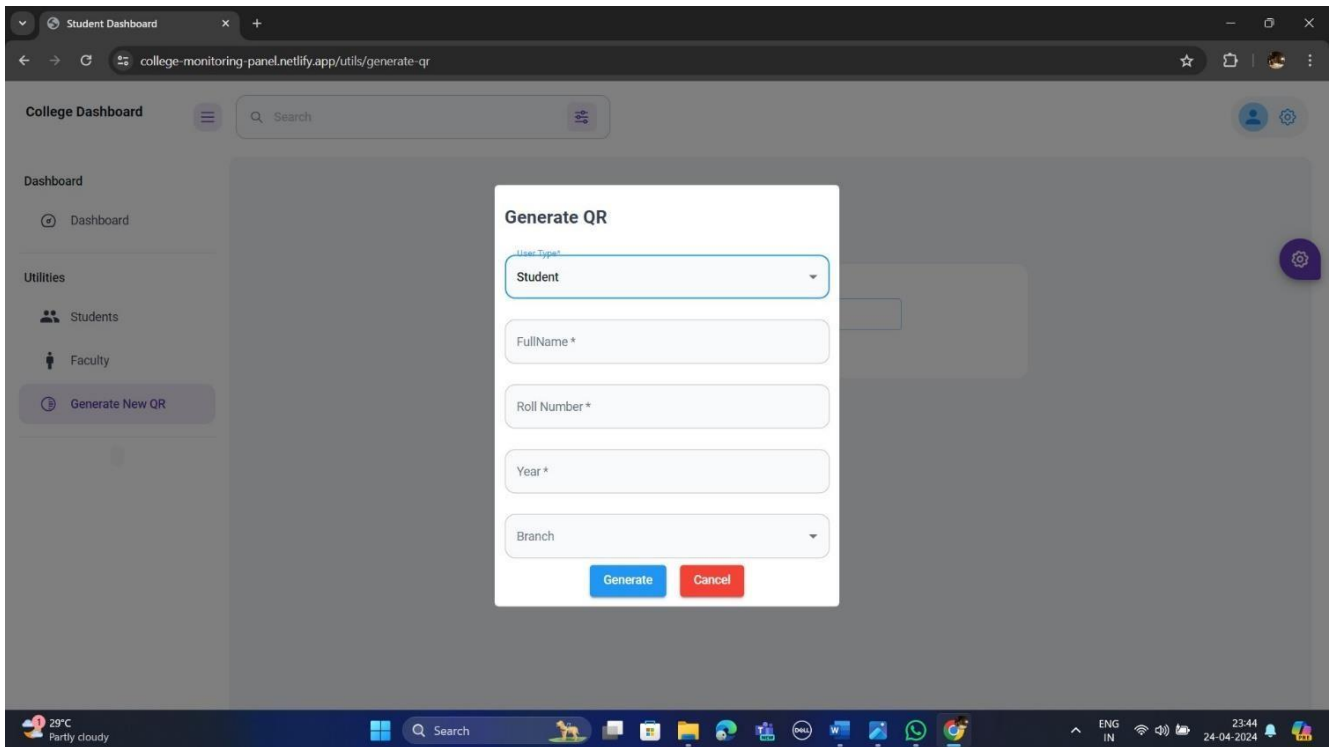
WEB-APP



WEB - APP LOGIN PAGE

















WEB - APP DASHBOARD

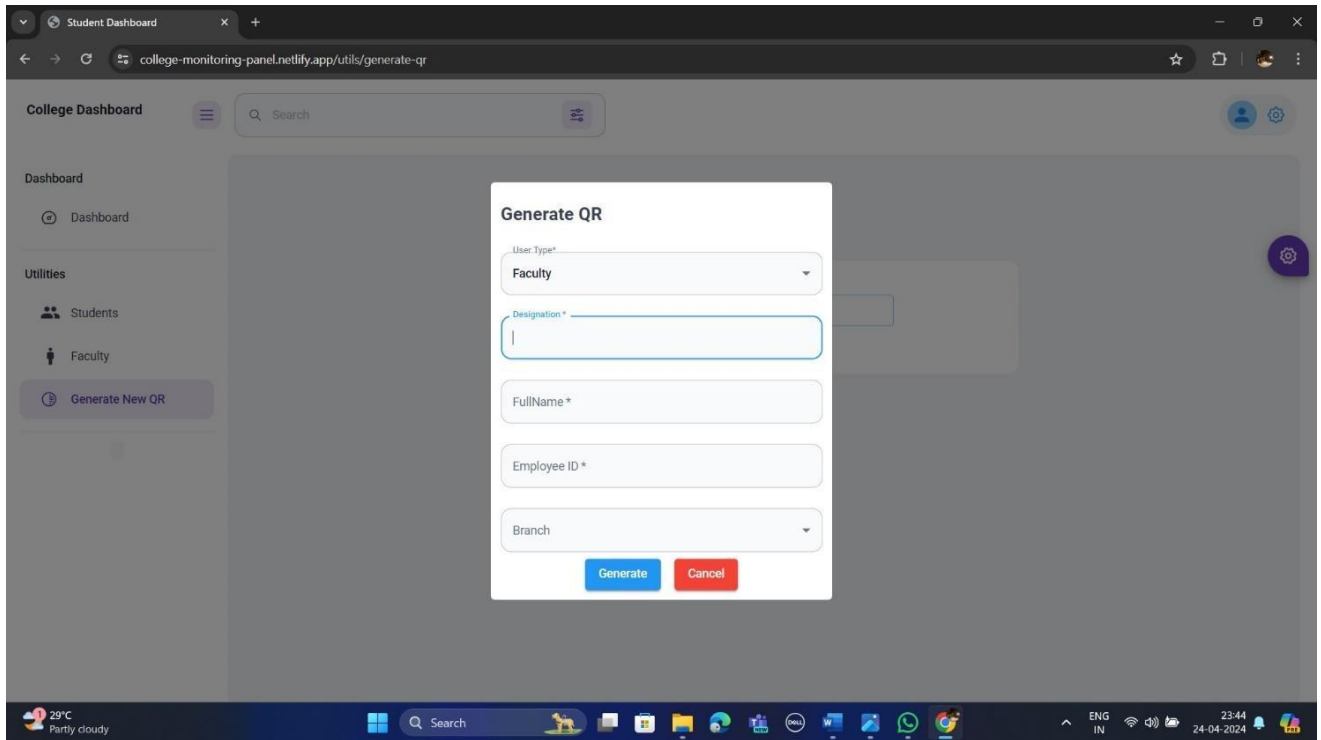


STUDENT QR-GENERATION

The screenshot shows the 'Students' table in the 'College Dashboard'. The table has 11 columns: Sl.No, Name, Roll Number, Branch, Year, QR-Code, Date, Exit Time, Entry Time, Total Time, and Actions. There are 7 rows of student data. An 'Add' button is located in the top right corner of the table area. The background shows the same sidebar and bottom status bar as the previous screenshot.

Sl.No	Name	Roll Number	Branch	Year	QR-Code	Date	Exit Time	Entry Time	Total Time	Actions
0	N Laya Devi	575	cse	4	Show Qr Code	2024-04-22	11:17:09	11:18:11	1	 
1	D Bhagya	5A7	cse	2	Show Qr Code	2024-04-22	12:39:18	12:39:41	0	 
2	K Bhagya	561	cse	4	Show Qr Code	2024-04-24	17:01:04	17:01:33	0	 
3	M Dhakshu	364	civil	3	Show Qr Code	2024-04-24	23:05:11	23:08:05	3	 
4	L Dheeskshu	465	ece	3	Show Qr Code	2024-04-24	23:05:17	23:08:14	3	 
5	J VISHNU	456	mechanical	5	Show Qr Code	2024-04-24	23:05:29	23:08:20	3	 
6	M Kalyani	167	civil	4	Show Qr Code	2024-04-24	23:05:36	23:10:35	5	 

STUDENT TABLE DETAILS



FACULTY QR-GENERATION

The screenshot shows the 'Faculty' table in the 'College Dashboard'. The table has the following columns: SI No, Faculty Name, Employee Id, Branch, Designation, QR-Code, Date, Exit Time, Entry Time, Total Time, and Actions. There are 5 rows of data. An 'Add' button is located in the top right corner of the table area.

SI No	Faculty Name	Employee Id	Branch	Designation	QR-Code	Date	Exit Time	Entry Time	Total Time	Actions
0	admin	123	admin	admin	Show Qr Code	Wed Apr 24 2024	22:52:54	22:53:17	0	
1	B Madhava Rao	247	cse	Sr Asst Prof	Show Qr Code	Wed Apr 24 2024	22:51:02	12:49:04	2	
2	Srinivas	145	ece	Lab Incharge	Show Qr Code	Wed Apr 24 2024	18:08:21	18:08:44	0	
3	K Mohan	667	civil	Proffesor	Show Qr Code	Wed Apr 24 2024	23:02:26	23:07:50	5	
4	K Sireesha	669	mechanical	Asst Proffesor	Show Qr Code	Wed Apr 24 2024	23:05:48	23:09:34	4	

FACULTY TABLE DETAILS

DATABASE (NoSQL)

The screenshot displays the MongoDB Atlas web interface for a cluster named 'Cluster0' in the 'RAHUL'S ORG - 2024-04-02' project. The interface is in the 'Collections' tab, showing the 'test.students' collection. The collection has a storage size of 68KB, a logical data size of 28.57KB, 10 total documents, and an index total size of 36KB. The left sidebar shows the 'Database' section with a tree view containing 'sample_mflix', 'test', 'faculties', and 'students'. The main panel shows the 'test.students' collection details, including a 'Filter' input field and a 'Query Results' section displaying a single document. The document contains fields: '_id', 'rollNumber', 'name', 'entryTime', 'exitTime', 'year', 'branch', 'qrCode', and 'totalTime'. The bottom status bar shows the system time as 23:45 on 24-04-2024.

Cluster0

VERSION 7.0.8 REGION AWS Mumbai (ap-south-1)

DATABASES: 2 COLLECTIONS: 8

+ Create Database

Search Namespaces

sample_mflix

test

faculties

students

test.students

STORAGE SIZE: 68KB LOGICAL DATA SIZE: 28.57KB TOTAL DOCUMENTS: 10 INDEXES TOTAL SIZE: 36KB

Find Indexes Schema Anti-Patterns Aggregation Search Indexes

Generate queries from natural language in Compass

Filter Type a query: { field: 'value' }

Reset Apply Options

INSERT DOCUMENT

QUERY RESULTS: 1-10 OF 10

```
{
  "_id": ObjectId("662356555970baa9e9e37321"),
  "rollNumber": "575",
  "name": "N. Laya Dev",
  "entryTime": "2024-04-22T11:19:11.897+00:00",
  "exitTime": "2024-04-22T11:17:09.247+00:00",
  "year": "4",
  "branch": "cse",
  "qrCode": "data:image/png;base64,iVBORw0KGgoAAAANSUHEugAAALQAAAC8CAYAAAA9zQYyAAAA...",
  "_v": 0,
  "totalTime": "1"
}
```

STUDENTS DATA COLLECTION

The screenshot displays the MongoDB Atlas web interface for the same cluster, now showing the 'test.faculties' collection. The collection has a storage size of 60KB, a logical data size of 25.49KB, 6 total documents, and an index total size of 36KB. The left sidebar shows the 'Database' section with a tree view containing 'sample_mflix', 'test', 'faculties', and 'students'. The main panel shows the 'test.faculties' collection details, including a 'Filter' input field and a 'Query Results' section displaying a single document. The document contains fields: '_id', 'employeeId', 'name', 'entryTime', 'branch', 'user', 'designation', 'exitTime', 'qrCode', and 'totalTime'. The bottom status bar shows the system time as 23:45 on 24-04-2024.

Cluster0

VERSION 7.0.8 REGION AWS Mumbai (ap-south-1)

DATABASES: 2 COLLECTIONS: 8

+ Create Database

Search Namespaces

sample_mflix

test

faculties

students

test.faculties

STORAGE SIZE: 60KB LOGICAL DATA SIZE: 25.49KB TOTAL DOCUMENTS: 6 INDEXES TOTAL SIZE: 36KB

Find Indexes Schema Anti-Patterns Aggregation Search Indexes

Generate queries from natural language in Compass

Filter Type a query: { field: 'value' }

Reset Apply Options

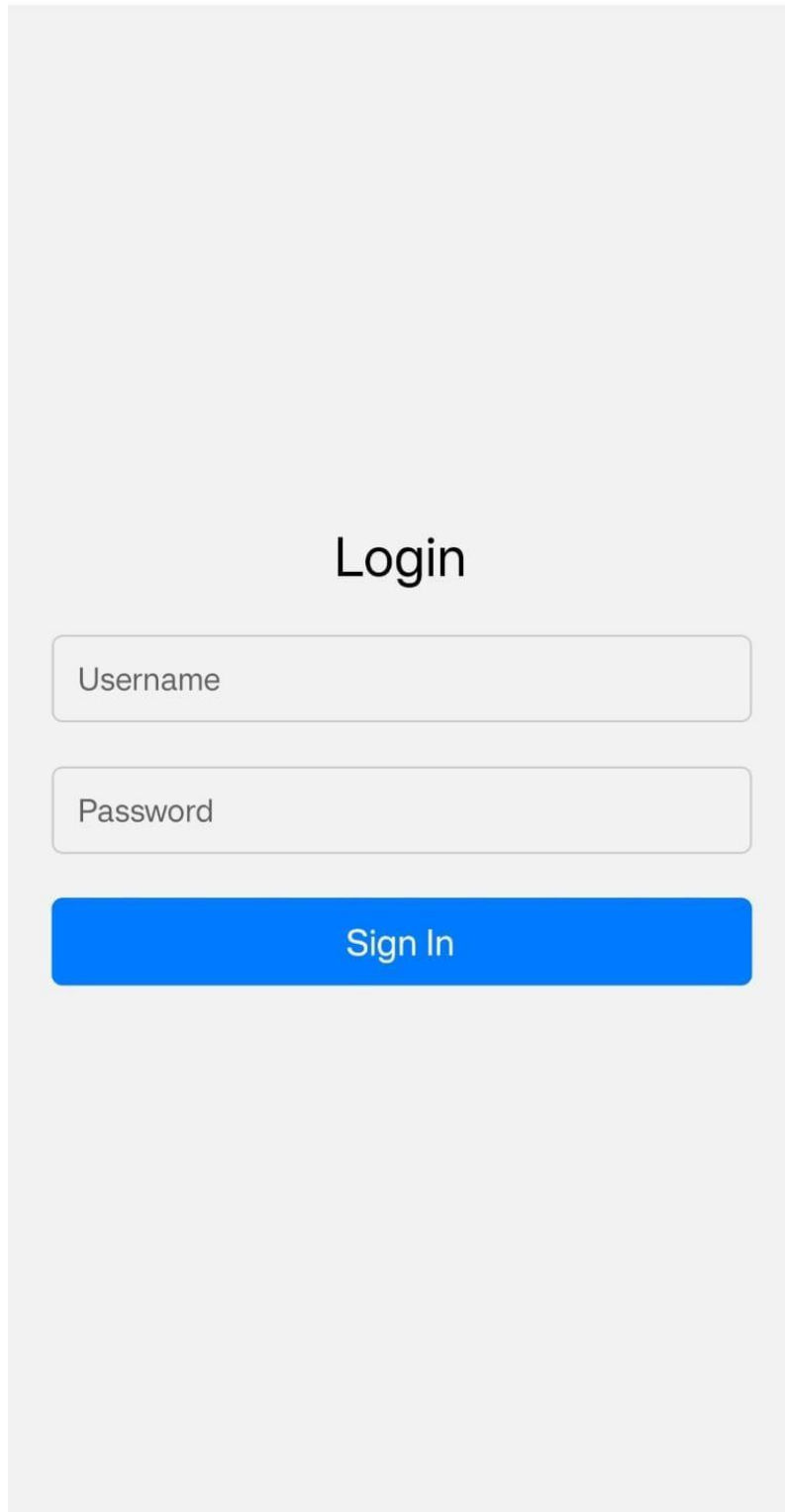
INSERT DOCUMENT

```
{
  "_id": ObjectId("66260e5d9c4de09c59e8cb97"),
  "employeeId": "247",
  "name": "B. Madhava Rao",
  "entryTime": "Mon Apr 22 2024 12:49:04 GMT+0000 (Coordinated Universal Time)",
  "branch": "cse",
  "user": "faculty",
  "designation": "Sr Asst Prof",
  "exitTime": "Wed Apr 24 2024 22:51:02 GMT+0000 (Coordinated Universal Time)",
  "qrCode": "data:image/png;base64,iVBORw0KGgoAAAANSUHEugAAAOQAAAD8CAYAAACIV41NAAAA...",
  "_v": 0,
  "totalTime": "2"
}
```

FACULTY DATA COLLECTION

MOBILE APP

Login

A mobile app login page mockup. It features a light gray background with a white rounded rectangle in the center. Inside the rectangle, the word "Login" is centered at the top. Below it are two input fields: "Username" and "Password", each with a light gray border and rounded corners. At the bottom of the rectangle is a blue button with rounded corners and the text "Sign In" in white.

Login

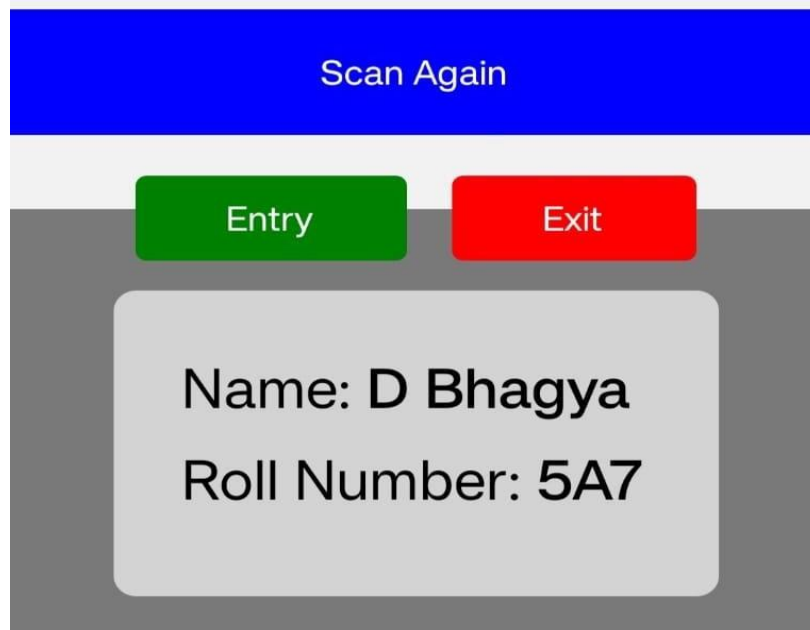
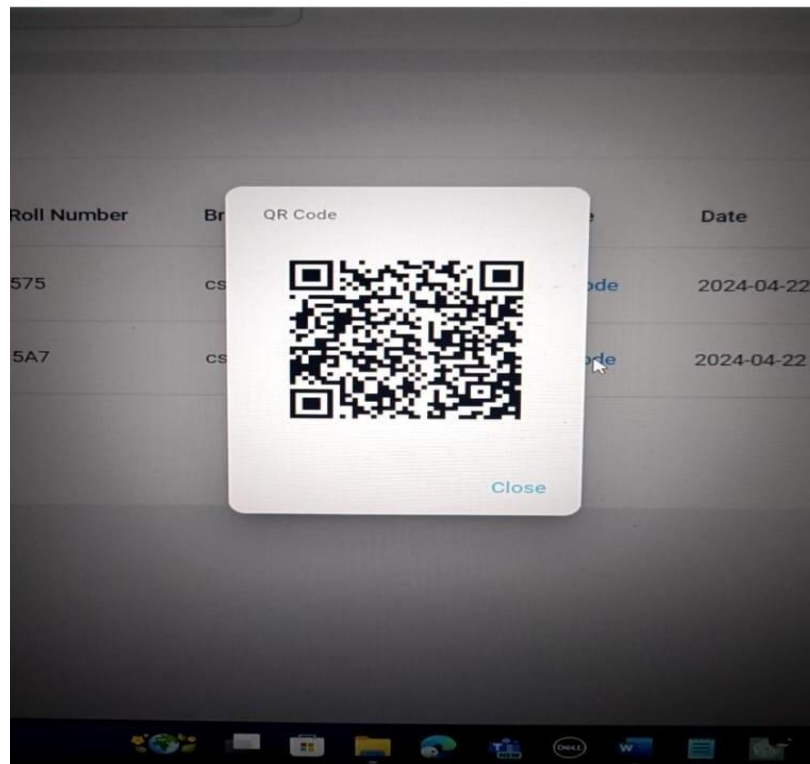
Username

Password

Sign In

MOBILE APP LOG IN PAGE

← Home



MOBILE APP INTERFACE

CHAPTER-06

TESTING

6.1 Introduction to Testing/Testing Concepts:

Testing is a critical phase in the development process of the QR-Based Campus Exit Management System. It ensures that the system functions as expected, meets the requirements, and is reliable and secure.

Here is an in-depth overview of the testing concepts and methodologies employed:

1. Unit Testing:

Unit testing involves testing individual components or units of code in isolation to validate their functionality. In our project:

- We wrote unit tests for frontend React components using testing libraries like Jest and Enzyme.
- Backend Node.js APIs were tested using frameworks like Mocha and Chai.
- Unit tests were focused on verifying the behaviour of each component and function, covering edge cases and error scenarios.

2. Integration Testing:

Integration testing evaluates the interaction between various components of the system. Key points include:

- We performed integration tests to ensure smooth communication between the frontend and backend.
- Testing APIs with tools like Postman ensured that the endpoints function correctly and return the expected responses.
- Integration tests also verified the integration of the MongoDB database with the backend, ensuring data persistence and retrieval.

3. End-to-End Testing:

End-to-end testing validates the entire flow of the system, simulating real-world scenarios.

Here's what we did:

- We used Cypress for end-to-end testing of the web application, covering user journeys from login to exit monitoring.
- For the mobile app, Appium was employed to automate testing across various devices and platforms, ensuring consistent behaviour.
- End-to-end tests checked the complete flow of scanning QR codes, updating exit records, and monitoring exits.

4. Performance Testing:

Performance testing evaluates the system's responsiveness and stability under different load conditions:

- Conducted load tests, simulating multiple users scanning QR codes concurrently.
- Performance tests measured response times of APIs and database queries to ensure they meet acceptable thresholds.
- Scalability tests were performed to assess how the system handles increased loads and concurrent users.

5. Security Testing:

Security testing identifies and mitigates vulnerabilities in the system:

- We conducted security scans to detect and fix security issues.
- We have done penetration testing to identify potential entry points for malicious attacks.
- Security tests verified that user data is encrypted, APIs are protected against injection attacks, and access controls are in place.

6.2 Sample Test Cases:

Below are the test cases that cover various aspects of the QR-Based Campus Exit Management System:

1. User Registration:

Test Case 1: Verify that a new user can register successfully with valid information.

Test Case 2: Verify that registration fails if required fields are left empty.

Test Case 3: Verify that registration fails if the email address format is incorrect.

2. QR Code Generation:

Test Case 4: Verify that a unique QR code is generated for each registered user.

Test Case 5: Verify that the QR code contains the correct user information.

Test Case 6: Verify that the QR code is scannable and leads to the correct user profile.

3. Exit Monitoring:

Test Case 7: Verify that the security personnel can scan a QR code using the mobile app.

Test Case 8: Verify that the exit record is updated in the database with the correct timestamp.

Test Case 9: Verify that the exit status is displayed accurately on the web application in real-time.

4. Error Handling:

Test Case 10: Verify that appropriate error messages are displayed if the QR code is invalid or expired.

Test Case 11: Verify that the system handles network errors gracefully and provides informative error messages.

5. Performance:

Test Case 12: Verify that the system response time remains below 2 seconds under normal load conditions.

Test Case 13: Verify that the system can handle 100 concurrent users scanning QR codes simultaneously without significant degradation in performance.

6. Security:

Test Case 14: Verify that user passwords are securely hashed and stored in the database.

Test Case 15: Verify that API endpoints are protected with authentication and authorization mechanisms.

These test cases ensure thorough testing of the system, covering functional requirements, usability, performance, and security aspects.

This comprehensive testing approach ensures the reliability and effectiveness of the QR-Based Campus Exit Management System, meeting both functional and non-functional requirements.

Here are some sample test cases for the QR-Based Campus Exit Management System:

Test Case ID	Test Case Description	Input	Expected Output	Actual Output	Result
Test Case_01	Verify security personnel can scan QR code	Scanned QR code	QR code scanned successfully	QR code scanned successfully	Pass
Test Case_02	Scan valid student QR code	Valid student QR code	Successful exit recorded	Successful exit recorded	Pass
Test Case_03	Scan valid faculty QR code	Valid faculty QR code	Successful exit recorded	Successful exit recorded	Pass
Test Case_04	Scan invalid QR code (incorrect format)	Invalid QR code format	Error: Invalid QR code	Error: Invalid QR code	Pass
Test Case_05	View exit history for student	Student ID	List of previous exits of the student	List of previous exits of the student	Pass
Test Case_06	View exit history for faculty	Faculty ID	List of previous exits of the faculty	List of previous exits of the faculty	Pass
Test Case_07	Database connectivity test	Connected to database	Successful database operations	Successful database operations	Pass
Test Case_08	Network connectivity test	Connected to network	Successful communication with server	Successful communication with server	Pass
Test Case_09	Simultaneous exit by multiple users	Multiple users	Multiple exits recorded simultaneously	Multiple exits recorded simultaneously	Pass
Test Case_10	View exit history for invalid user ID	Invalid user ID	Error: Invalid user ID	Error: Invalid user ID	Pass
Test Case_11	View exit history for empty records	No previous exits	Empty exit history	Empty exit history	Pass
Test Case_12	Verify system performance under the heavy load	More than 100 concurrent users	No significant degradation in system performance	No significant degradation in system performance	Pass
Test Case_13	Verify system response time under normal load	Normal load	Calculate the System response time	Calculate the System response time	Pass
Test Case_14	Verify exit status displayed accurately	Web - application	Exit status displayed in real-time	Exit status displayed in real-time	Pass
Test Case_15	Security: Verify secure storage of admin passwords	Passwords	Password securely stored in database	Password securely stored in database	Pass

These test cases cover various aspects of the QR-Based Campus Exit Management System, ensuring that it meets functional requirements, usability, performance, and security standards.

CONCLUSION

The QR-Based Campus Exit Management System provides a robust solution to monitor the movement of students and faculty in and out of the campus premises efficiently. By leveraging technologies like React.js, Material UI, Node.js, MongoDB, and React Native, we have developed a system that enhances campus security, automates exit monitoring, and provides user-friendly interfaces for administrators and security personnel.

Through this project, we've concluded that implementing a QR-based exit management system significantly improves campus security and operational efficiency. The system ensures accurate monitoring of exits, reduces manual effort, and provides real-time data updates. Additionally, the use of scalable and cost-effective technologies makes the system accessible to educational institutions with varying budgets.

Overall, the QR-Based Campus Exit Management System represents a step forward in campus security and management, providing a foundation for further innovation and improvement in the future.

FUTURE ENHANCEMENT

1. Push Notifications:

- Introduce push notifications to notify administrators and users about important events, such as late exits, unauthorized entries, or system updates.
- Push notifications will enhance communication and ensure timely responses to critical events.

2. Offline Mode:

- Develop an offline mode for the mobile app to allow security personnel to scan QR codes and update exit records even when internet connectivity is unavailable.
- Offline mode will ensure continuous operation of the system, even in areas with poor network coverage.

3. Integration with Student/Faculty Databases:

- Integrate the system with existing student and faculty databases to streamline user registration and improve data accuracy.
- It may also integrate using normalization techniques, which needs less space to store & retrieve the data accurately.
- Integration with databases will eliminate duplicate data entry and ensure consistency across systems.

4. Enhanced Reporting and Analytics:

- Implement advanced reporting features to provide administrators with deeper insights into exit activities, peak hours, and user behaviour.
- Analytics tools will enable administrators to make data-driven decisions and optimize campus security measures.

These future enhancements will further improve the functionality, security, and usability of the QR-Based Campus Exit Management System, ensuring it remains effective in meeting the evolving needs of Educational Institutions.

REFERENCES

- React.js Documentation:
<https://reactjs.org/docs/getting-started.html>
- Node.js Documentation:
<https://nodejs.org/en/docs/>
- MongoDB Documentation:
<https://docs.mongodb.com/>
- RESTful API Design Best Practices:
<https://restfulapi.net/>
- React Native Documentation:
<https://reactnative.dev/docs/getting-started>
- Project Expo Documentation:
<https://docs.expo.io/>
- <https://www.youtube.com/>
- <https://www.google.com/>
- Software engineering book (Author :R.Pressman)

**-THE END-
THANK YOU**