**Chapter 1**

**INTRODUCTION**

In today's fast-paced educational environment, the efficient management of administrative tasks is vital for the smooth operation and effective functioning of colleges and universities. To address this challenge, we present the College ERP System, a comprehensive software solution designed to streamline and automate key administrative processes within educational institutions. The College ERP System empowers administrators, students, and faculty members with a user-friendly interface and a range of features tailored to their specific needs. Admin have the ability to add and delete faculty, departments, and students, ensuring accurate and up-to-date records. Faculty members play a crucial role in the educational process, and the College ERP System equips them with powerful tools to enhance their teaching and assessment capabilities. Faculty members can effortlessly create tests, upload marks, and mark attendance through a streamlined process. Through secure login credentials, students can conveniently view their test results, attendance records, and subject lists. This transparency fosters an environment of accountability and empowers students to take charge of their learning experience. The College ERP System harnesses a combination of modern technologies to deliver a seamless user experience. ReactJS and Tailwind CSS power the responsive and intuitive front-end interface, while the back-end leverages MongoDB for efficient data storage, Express.js for robust web application development, Redux for state management, and JWT for secure user authentication.

**1.1 Project description**

The College ERP project is a comprehensive system designed to streamline administrative tasks and enhance communication within a college. It includes modules for administrators, faculty, and students. Administrators can manage faculty, departments, and student records. Faculty members can create tests, mark attendance, and upload marks. Students can view their test results, attendance, and subject list. The project aims to improve efficiency, facilitate data management, and provide a user-friendly platform for seamless interaction among stakeholders in the college ecosystem.

**1.1.1 Objective**

**1. Streamline Administrative Tasks:** The primary objective of the College ERP System is to streamline administrative tasks related to managing faculty, departments, and student records. By providing a centralized platform for data management, minimize errors, and improve overall efficiency.

**2. Enhance Faculty Efficiency:** The system aims to empower faculty members by providing them with tools to create tests, upload marks, and mark attendance. By automating these processes, faculty

members can save time, reduce paperwork, and focus more on teaching and providing valuable feedback to students.

**3. Improve Student Experience:** The College ERP System aims to enhance the educational experience for students. By enabling students to log in and access their test results, attendance records, and subject lists, the system promotes transparency, accountability, and empowers students to actively engage in their academic journey.

**4. Facilitate Collaboration and Communication:** The system aims to improve collaboration and communication among different stakeholders within the college, including administrators, faculty members, and students. By providing a centralized platform, the system enables efficient sharing of information, fostering better coordination and decision-making.

**5. Ensure Data Accuracy and Security:** One of the key objectives of the College ERP System is to ensure data accuracy and security. By implementing secure authentication mechanisms, data encryption, and access controls, the system protects sensitive information and ensures that data remains confidential and reliable.

**6. Promote Efficiency and Productivity:** Overall, the College ERP System aims to promote efficiency and productivity within the college environment. By automating routine administrative tasks, eliminating manual paperwork, and providing user-friendly interfaces, the system enables users to focus their efforts on more value-added activities, leading to increased productivity.

**1.1.2 purpose**

The purpose of the College ERP System is to provide a comprehensive software solution that effectively manages administrative tasks in colleges and universities. The system aims to streamline processes, improve communication and collaboration, and enhance the overall efficiency of educational institutions.

**1.1.3 Scope**

The scope of the College ERP System may vary depending on the specific requirements and goals of the educational institution implementing it. However, the system generally focuses on streamlining administrative tasks, enhancing communication and collaboration, and providing efficient management of faculty, departments, and student-related information.

**Chapter:2**

**SOFTWARE AND HARDWARE REQUIREMENTS**

To be used efficiently, all computer software needs certain hardware components and software resources to be present on a computer. These prerequisites are known as(computer) system requirements and are of- ten used as a guideline as opposed to an absolute rule. Most software defines two sets of system requirements: minimum and recommended. With an increasing demand for higher processing power and resources in newer versions of software, system requirements tend to increase over time. Industry analysts suggest that this trend plays a bigger part in driving upgrades to exist computer systems than technological advancements.

**2.1 Software Requirement Specification**

1. **Operating System:** The College ERP System should be compatible with common operating systems such as Windows, macOS, and Linux.
2. **Web Server:** A web server, such as Apache or Chrome, is required to host the web application.
3. **Backend Framework:** The system requires the installation of Node.js and Express.js, which serve as the backend framework for handling server-side operations.
4. **Database:** The system relies on a MongoDB database for storing and managing data. MongoDB should be installed and configured.
5. **Frontend Framework:** ReactJS is used as the frontend framework for building the user interface. It needs to be installed along with supporting libraries and dependencies.
6. **State Management:** Redux is utilized for state management within the application. It should be installed and configured accordingly.
7. **Styling and UI Components:** Tailwind CSS is used for styling and Material UI Icons are used for icons and UI components. Both should be included as dependencies

**2.2 Hardware Requirements**

1. **Processor:** A modern processor with multiple cores, such as Intel Core i5 or AMD Ryzen, is recommended for efficient performance.
2. **RAM:** The system should have a minimum of 8 GB RAM to ensure smooth operation, especially when dealing with large datasets and concurrent user access.
3. **Storage:** Sufficient storage space is required to store the application code, dependencies, and database. A minimum of 100 GB of storage is recommended.
4. **Network Connectivity:** The system should have a stable internet connection for accessing the web application and interacting with the database.
5. **Display and Input:** A standard monitor or display with a minimum resolution of 1280x800 is sufficient. A keyboard and mouse or other input devices are required for user interaction

**2.3 Functional and Non-functional Requirements**

Functional requirements refer to the specific functionalities and features that a system or software should possess to fulfill its intended purpose. These requirements describe what the system should do and how it should behave in various scenarios. Examples of functional requirements for a college ERP system may include user authentication, course registration, grade management, and report generation.

Non-functional requirements, on the other hand, define the qualities and constraints of the system rather than its specific functionalities. These requirements focus on aspects such as performance, security, usability, reliability, and scalability. Examples of non-functional requirements for a college ERP system may include fast response times, data encryption for security, intuitive user interface, high availability, and support for a large number of concurrent users.

**2.3.1 Functional Requirements**

**User Registration and Authentication:** Users (administrators, faculty members, and students) should be able to register and create their accounts. The system should authenticate users' credentials (username and password) to ensure secure access.

**Administrator Functionality:** Administrators should be able to add, edit, and delete faculty, departments, and student records. Administrators should have access to comprehensive reports and analytics for decision-making. Admin should be able to manage user roles and permissions within the system.

**Faculty Functionality:** Faculty members should be able to create tests, upload marks, and mark attendance for students. Faculty members should have access to relevant student information and progress reports. Faculty members should be able to communicate with students and other stakeholders through messaging or discussion forums.

**Student Functionality:** Students should be able to log in to the system using their credentials.

Students should be able to view their test results, attendance records, and subject lists. Students should have the ability to communicate with faculty members and administrators if needed.

**Communication and Collaboration:** The system should provide features for communication and collaboration among administrators, faculty members, and students, such as messaging, announcements, and discussion forums.

**2.3.2 Non-Functional Requirements**

**Usability:** The system should have a user-friendly interface with intuitive navigation and clear instructions. The system should provide helpful error messages and feedback to guide users.

**Performance:** The system should be able to handle concurrent user access and provide a responsive user experience. The system should have optimized database queries and efficient data retrieval to minimize response times.

**Security:** The system should implement secure authentication and authorization mechanisms

to protect user data and prevent unauthorized access. The system should adhere to data protection regulations and ensure the confidentiality and integrity of user information.

**Scalability:** The system should be designed to handle increasing user loads and accommodate future growth in terms of users, data, and functionality. The system should be able to scale horizontally or vertically to meet growing demands.

**Reliability:** The system should have minimal downtime and be highly available to ensure uninterrupted access. The system should have data backup and disaster recovery mechanisms in place to prevent data loss.

**Compatibility:** The system should be compatible with different web browsers and devices to ensure accessibility for users.

**Chapter 3**

**DESIGN IMPLEMENTATION**

The design activity is often divided into two separate phase system design and detailed design. System design is also called top-level design. At the first level focus is on deciding which modules are needed for the system, the specifications of these modules and how the modules should be interconnected. This is called system design or top-level design. In the second level the internal design of the modules or how the specifications of the module can be satisfied is decided. This design level is often called detailed design or logic design.

**3.1 System Design**

The design of the College ERP System involves several components and architectural decisions to ensure its functionality, scalability, and maintainability. Here is an overview of the system design:

1. **Architecture:**

* The system follows a client-server architecture, where the client-side is built using ReactJS and the server-side uses Node.js with Express.js framework.
* The system adopts a three-tier architecture, consisting of the presentation layer (frontend), application layer (backend), and data layer (database)

1. **Frontend Design:**

* The frontend is developed using ReactJS, which provides a modular and component-based approach to building the user interface.
* The UI is designed to be responsive and user-friendly, following modern design principles.
* Tailwind CSS is utilized for styling, providing a utility-first approach for rapid UI development.

1. **Backend Design:**

* The backend is built using Node.js and Express.js, providing a robust and scalable server-side framework.
* APIs are designed and implemented using RESTful principles to handle various functionalities and data transactions.
* Authentication and authorization mechanisms are implemented using JWT (JSON Web Tokens) to secure user access and protect sensitive information.
* Business logic and data processing are implemented in the backend, ensuring efficient handling of operations and interactions with the database.
* Material UI Icons are used for displaying icons and UI components.

1. **Database Design:**

* MongoDB is chosen as the database to store and manage the system's data.
* The database schema is designed to represent entities such as faculty, departments, students, tests, and attendance.
* Proper indexing and query optimization techniques are applied to ensure efficient data retrieval and manipulation.

1. **Communication and Integration:**

* The system may incorporate communication features such as messaging and discussion forums to facilitate collaboration between users.
* Integration with external systems or services, such as email notifications or third-party APIs, can be considered to enhance system functionality.

1. **Security and Data Protection:**

* The system employs secure authentication and authorization mechanisms to ensure user access control and protect user data.
* Data encryption techniques can be utilized to secure sensitive information stored in the database.
* Proper measures are taken to prevent common security vulnerabilities.

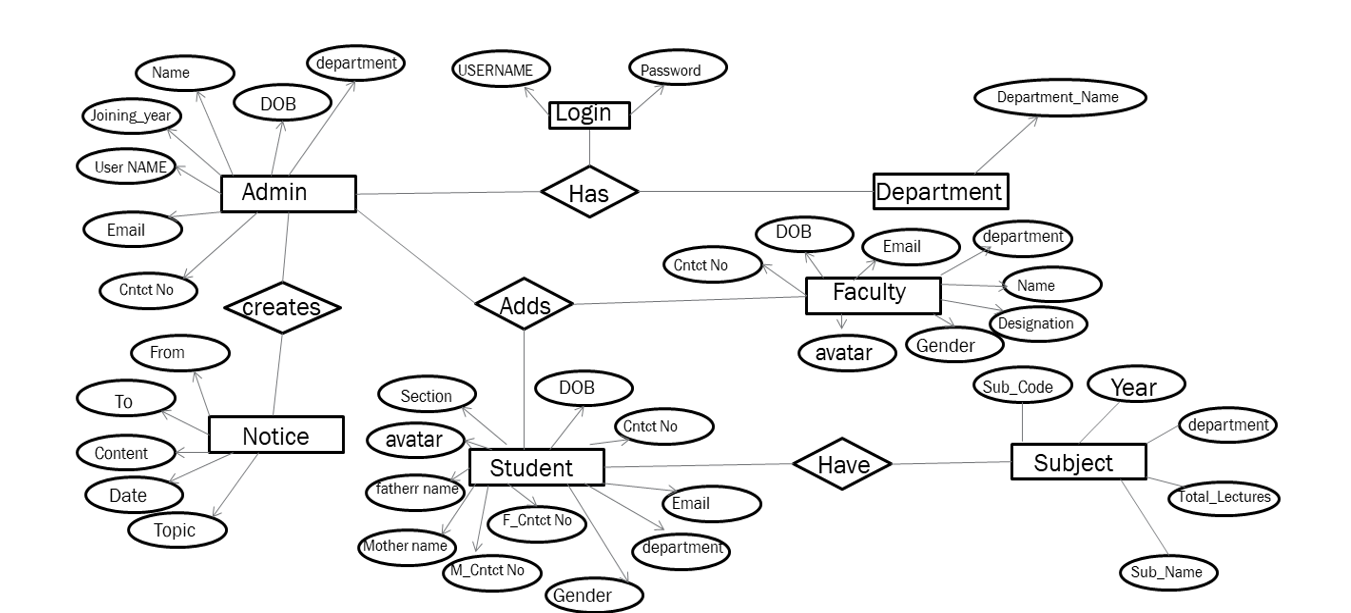
**3.2 ER diagram**

An ER (Entity-Relationship) diagram is a visual representation of the entities, attributes, and relationships within a system or database. It helps to model the logical structure of a system and depicts how different entities are related to each other. ER diagrams are widely used in database design and serve as a foundation for creating a relational database schema.

In an ER diagram, entities are represented as rectangles, attributes as ovals, and relationships as lines connecting the entities. The cardinality and participation constraints between entities are also indicated

in the diagram. It helps to visualize the data model and understand the relationships and dependencies between different entities and attributes.

For example, in the context of a college ERP system, an ER diagram may include entities such as Faculty, Department, Student, Test, and Attendance. The relationships between these entities, such as "Faculty teaches Department," "Student belongs to Department," and "Test is conducted for Student," can be represented with appropriate cardinalities and constraints.

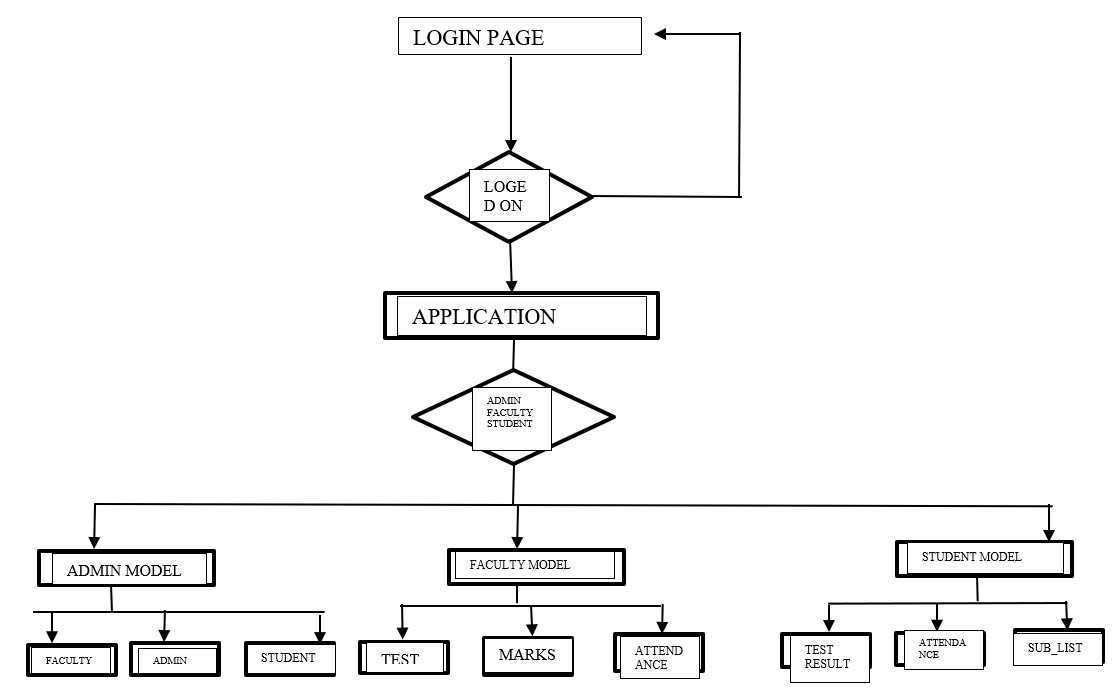


**Figure: 3.2 ER diagram**

**3.3 Workflow Diagram**

A workflow diagram, also known as a process flowchart or workflow chart, illustrates the sequence of steps or activities involved in a process or workflow. It provides a visual representation of how tasks are performed, their dependencies, and the flow of information or materials throughout the process. Workflow diagrams typically use various symbols and shapes to represent different elements, such as rectangles for tasks, diamonds for decisions, arrows for the flow of activities, and connectors for joining or splitting paths. They help to identify bottlenecks, inefficiencies, or areas of improvement in a process and enable stakeholders to understand the workflow more easily.

In the case of a college ERP system, a workflow diagram can be created to illustrate the process of adding and deleting faculty, departments, and students. It would show the sequence of steps involved, such as capturing information, validating data, making changes in the database, and updating relevant records. Similarly, workflow diagrams can be created for other processes within the system, such as creating tests, uploading marks, and marking attendance.

**Figure 3.3 workflow diagram**

**3.4 Source Code**

**3.4.1 App.js:**

The provided App.js code is the central component of a React application. It defines the routes and corresponding components for different pages within the application. The routes are configured using the `react-router-dom` library. The component hierarchy includes various components for login, profile management, data manipulation, and information display for administrators, faculty members, and students. Each route is associated with a specific component that is rendered when the corresponding URL is accessed.

import React from "react";

import { Routes, Route } from "react-router-dom";

import AddAdmin from "./components/admin/addAdmin/AddAdmin";

import AddDepartment from "./components/admin/addDepartment/AddDepartment";

import AddFaculty from "./components/admin/addFaculty/AddFaculty";

import AddStudent from "./components/admin/addStudent/AddStudent";

import AddSubject from "./components/admin/addSubject/AddSubject";

import AdminHome from "./components/admin/AdminHome";

import GetFaculty from "./components/admin/getFaculty/GetFaculty";

import GetStudent from "./components/admin/getStudent/GetStudent";

import GetSubject from "./components/admin/getSubject/GetSubject";

import AdminProfile from "./components/admin/profile/Profile";

import AdminPassword from "./components/admin/profile/update/password/Password";

import AdminUpdate from "./components/admin/profile/update/Update";

import CreateTest from "./components/faculty/createTest/CreateTest";

import FacultyHome from "./components/faculty/FacultyHome";

import MarkAttendance from "./components/faculty/markAttendance/MarkAttendance";

import FacultyProfile from "./components/faculty/profile/Profile";

import FacultyPassword from "./components/faculty/profile/update/password/Password";

import FacultyUpdate from "./components/faculty/profile/update/Update";

import UploadMarks from "./components/faculty/uploadMarks/UploadMarks";

import AdminLogin from "./components/login/adminLogin/AdminLogin";

import FacultyLogin from "./components/login/facultyLogin/FacultyLogin";

import Login from "./components/login/Login";

import StudentLogin from "./components/login/studentLogin/StudentLogin";

import StudentHome from "./components/student/StudentHome";

import StudentProfile from "./components/student/profile/Profile";

import StudentUpdate from "./components/student/profile/update/Update";

import StudentPassword from "./components/student/profile/update/password/Password";

import SubjectList from "./components/student/subjectList/SubjectList";

import TestResult from "./components/student/testResult/TestResult";

import Attendance from "./components/student/attendance/Attendance";

import DeleteAdmin from "./components/admin/deleteAdmin/DeleteAdmin";

import DeleteDepartment from "./components/admin/deleteDepartment/DeleteDepartment";

import DeleteFaculty from "./components/admin/deleteFaculty/DeleteFaculty";

import DeleteStudent from "./components/admin/deleteStudent/DeleteStudent";

import DeleteSubject from "./components/admin/deleteSubject/DeleteSubject";

import CreateNotice from "./components/admin/createNotice/CreateNotice";

const App = () => {

return (

<Routes>

<Route exact path="/" element={<Login />} />

{/\* Admin \*/}

<Route path="/login/adminlogin" element={<AdminLogin />} />

<Route path="/admin/home" element={<AdminHome />} />

<Route path="/admin/profile" element={<AdminProfile />} />

<Route path="/admin/update" element={<AdminUpdate />} />

<Route path="/admin/update/password" element={<AdminPassword />} />

<Route

path="/admin/updatepassword"

element={<AdminFirstTimePassword />} />

<Route path="/admin/createnotice" element={<CreateNotice />} />

<Route path="/admin/addadmin" element={<AddAdmin />} />

<Route path="/admin/deleteadmin" element={<DeleteAdmin />} />

<Route path="/admin/adddepartment" element={<AddDepartment />} />

<Route path="/admin/deletedepartment" element={<DeleteDepartment />} />

<Route path="/admin/addfaculty" element={<AddFaculty />} />

<Route path="/admin/deletefaculty" element={<DeleteFaculty />} />

<Route path="/admin/deletestudent" element={<DeleteStudent />} />

<Route path="/admin/deletesubject" element={<DeleteSubject />} />

<Route path="/admin/allfaculty" element={<GetFaculty />} />

<Route path="/admin/addstudent" element={<AddStudent />} />

<Route path="/admin/addsubject" element={<AddSubject />} />

<Route path="/admin/allsubject" element={<GetSubject />} />

<Route path="/admin/allstudent" element={<GetStudent />} />

{/\* Faculty \*/}

<Route path="/login/facultylogin" element={<FacultyLogin />} />

<Route path="/faculty/home" element={<FacultyHome />} />

<Route path="/faculty/password" element={<FacultyFirstTimePassword />} />

<Route path="/faculty/profile" element={<FacultyProfile />} />

<Route path="/faculty/update" element={<FacultyUpdate />} />

<Route path="/faculty/update/password" element={<FacultyPassword />} />

<Route path="/faculty/createtest" element={<CreateTest />} />

<Route path="/faculty/uploadmarks" element={<UploadMarks />} />

<Route path="/faculty/markattendance" element={<MarkAttendance />} />

{/\* Student \*/}

<Route path="/login/studentlogin" element={<StudentLogin />} />

<Route path="/student/home" element={<StudentHome />} />

<Route path="/student/password" element={<StudentFirstTimePassword />} />

<Route path="/student/profile" element={<StudentProfile />} />

<Route path="/student/update" element={<StudentUpdate />} />

<Route path="/student/update/password" element={<StudentPassword />} />

<Route path="/student/subjectlist" element={<SubjectList />} />

<Route path="/student/testresult" element={<TestResult />} />

<Route path="/student/attendance" element={<Attendance />} />

</Routes> );

};

export default App;

**3.4.2 AdminLogin.js:**

The provided AdminLogin.js code is a React component representing a login form for administrators. It utilizes local state variables to manage the username, password, and visibility of the password input. Redux is used for state management and dispatching the adminSignIn action. The form includes error handling, a loading spinner during login, and a UI design implemented with Tailwind CSS. The component transitions the form elements with animations when rendered.

import React, { useEffect, useState } from "react";

import { useNavigate } from "react-router";

import { useDispatch, useSelector } from "react-redux";

import { adminSignIn } from "../../../redux/actions/adminActions";

import VisibilityIcon from "@mui/icons-material/Visibility";

import VisibilityOffIcon from "@mui/icons-material/VisibilityOff";

import Spinner from "../../../utils/Spinner";

const AdminLogin = () => {

const [translate, setTranslate] = useState(false);

const [username, setUsername] = useState("");

const [password, setPassword] = useState("");

const [showPassword, setShowPassword] = useState(false);

const [loading, setLoading] = useState(false);

const dispatch = useDispatch();

const navigate = useNavigate();

const store = useSelector((state) => state);

const [error, setError] = useState({});

useEffect(() => {

setTimeout(() => {

setTranslate(true);

}, 1000);

}, []);

useEffect(() => {

if (store.errors) {

setError(store.errors);

}

}, [store.errors]);

const login = (e) => {

e.preventDefault();

setLoading(true);

dispatch(adminSignIn({ username: username, password: password }, navigate));

};

useEffect(() => {

if (store.errors) {

setLoading(false);

setUsername("");

setPassword("");

}

}, [store.errors]);

return (

<div className="bg-[#04bd7d] h-screen w-screen flex items-center justify-center">

<div className="grid grid-cols-2">

<div

className={`h-96 w-96 bg-white flex items-center justify-center ${

translate ? "translate-x-[12rem]" : ""

} duration-1000 transition-all rounded-3xl shadow-2xl`}>

<h1 className="text-[3rem] font-bold text-center">

Admin

<br />

Login

</h1>

</div>

<form

onSubmit={login}

className={`${

loading ? "h-[27rem]" : "h-96"

} w-96 bg-[#2c2f35] flex flex-col items-center justify-center ${

translate ? "-translate-x-[12rem]" : ""

} duration-1000 transition-all space-y-6 rounded-3xl shadow-2xl`}>

<h1 className="text-white text-3xl font-semibold">Admin</h1>

<div className="space-y-1">

<p className="text-[#515966] font-bold text-sm">Username</p>

<div className="bg-[#515966] rounded-lg w-[14rem] flex items-center">

<input

onChange={(e) => setUsername(e.target.value)}

value={username}

type="text"

required

className="bg-[#515966] text-white px-2 outline-none py-2 rounded-lg placeholder:text-sm"

placeholder="Username" />

</div>

</div>

<div className="space-y-1">

<p className="text-[#515966] font-bold text-sm">Password</p>

<div className="bg-[#515966] rounded-lg px-2 flex items-center">

<input

onChange={(e) => setPassword(e.target.value)}

value={password}

required

type={showPassword ? "text" : "password"}

className=" bg-[#515966] text-white rounded-lg outline-none py-2 placeholder:text-sm"

placeholder="Password" />

{showPassword ? (

<VisibilityIcon

onClick={() => setShowPassword(!showPassword)}

className="cursor-pointer" />

) : (

<VisibilityOffIcon

onClick={() => setShowPassword(!showPassword)}

className="cursor-pointer" />

)}

</div>

</div>

<button

type="submit"

className="w-32 hover:scale-105 transition-all duration-150 rounded-lg flex items-center justify-center text-white text-base py-1 bg-[#04bd7d]">

Login

</button>

{loading && (

<Spinner

message="Logging In"

height={30}

width={150}

color="#ffffff"

messageColor="#fff" />

)}

{(error.usernameError || error.passwordError) && (

<p className="text-red-500">

{error.usernameError || error.passwordError}

</p>

)}

</form>

</div>

</div>

);

};

export default AdminLogin;

**3.4.3 FacultyLogin.js:**

The provided FacultyLogin.js code is a React component that represents a login form for faculty members. It includes form fields for username and password, with options to show or hide the password. It uses Redux for state management and dispatches an action to sign in the faculty upon form submission. The component also handles error handling and displays a loading spinner during the login process. The UI is designed using Tailwind CSS classes.

import React, { useEffect, useState } from "react";

import { useNavigate } from "react-router";

import { useDispatch, useSelector } from "react-redux";

import { signin } from "../../../redux/actions/adminActions";

import VisibilityIcon from "@mui/icons-material/Visibility";

import VisibilityOffIcon from "@mui/icons-material/VisibilityOff";

import Spinner from "../../../utils/Spinner";

import { facultySignIn } from "../../../redux/actions/facultyActions";

const FacultyLogin = () => {

const [translate, setTranslate] = useState(false);

const [username, setUsername] = useState("");

const [password, setPassword] = useState("");

const [showPassword, setShowPassword] = useState(false);

const [loading, setLoading] = useState(false);

const dispatch = useDispatch();

const navigate = useNavigate();

const store = useSelector((state) => state);

const [error, setError] = useState({});

useEffect(() => {

setTimeout(() => {

setTranslate(true);

}, 1000);

}, []);

useEffect(() => {

if (store.errors) {

setError(store.errors);

}

}, [store.errors]);

const login = (e) => {

e.preventDefault();

setLoading(true);

dispatch(

facultySignIn({ username: username, password: password }, navigate)

); };

useEffect(() => {

if (store.errors) {

setLoading(false);

setUsername("");

setPassword("");

}

}, [store.errors]);

return (

<div className="bg-[#5a51d6] h-screen w-screen flex items-center justify-center">

<div className="grid grid-cols-2">

<div

className={`h-96 w-96 bg-white flex items-center justify-center ${

translate ? "translate-x-[12rem]" : ""

} duration-1000 transition-all rounded-3xl shadow-2xl`}>

<h1 className="text-[3rem] font-bold text-center">

Faculty

<br />

Login

</h1>

</div>

<form

onSubmit={login}

className={`${

loading ? "h-[27rem]" : "h-96"

} w-96 bg-[#2c2f35] flex flex-col items-center justify-center ${

translate ? "-translate-x-[12rem]" : ""

} duration-1000 transition-all space-y-6 rounded-3xl shadow-2xl`}>

<h1 className="text-white text-3xl font-semibold">Faculty</h1>

<div className="space-y-1">

<p className="text-[#515966] font-bold text-sm">Username</p>

<div className="bg-[#515966] rounded-lg w-[14rem] flex items-center">

<input

onChange={(e) => setUsername(e.target.value)}

value={username}

type="text"

required

className="bg-[#515966] text-white px-2 outline-none py-2 rounded-lg placeholder:text-sm"

placeholder="Username"

/>

</div>

</div>

<div className="space-y-1">

<p className="text-[#515966] font-bold text-sm">Password</p>

<div className="bg-[#515966] rounded-lg px-2 flex items-center">

<input

onChange={(e) => setPassword(e.target.value)}

value={password}

required

type={showPassword ? "text" : "password"}

className=" bg-[#515966] text-white rounded-lg outline-none py-2 placeholder:text-sm"

placeholder="Password"

/>

{showPassword ? (

<VisibilityIcon

onClick={() => setShowPassword(!showPassword)}

className="cursor-pointer"

/>

) : (

<VisibilityOffIcon

onClick={() => setShowPassword(!showPassword)}

className="cursor-pointer"

/>

)}

</div>

</div>

<button

type="submit"

className="w-32 hover:scale-105 transition-all duration-150 rounded-lg flex items-center justify-center text-white text-base py-1 bg-[#04bd7d]">

Login

</button>

{loading && (

<Spinner

message="Logging In"

height={30}

width={150}

color="#ffffff"

messageColor="#fff"

/>

)}

{(error.usernameError || error.passwordError) && (

<p className="text-red-500">

{error.usernameError || error.passwordError}

</p>

)}

</form>

</div>

</div>

);

};

export default FacultyLogin;

**3.4.4 SudentLogin.js:**

The provided StudentLogin.js code is a React component that represents a login form for students. It

includes form fields for username and password, with options to show or hide the password. It uses Redux for state management and dispatches an action to sign in the student upon form submission. The component also handles error handling and displays a loading spinner during the login process. The UI is designed using Tailwind CSS classes.

import React, { useEffect, useState } from "react";

import { useNavigate } from "react-router";

import { useDispatch, useSelector } from "react-redux";

import VisibilityIcon from "@mui/icons-material/Visibility";

import VisibilityOffIcon from "@mui/icons-material/VisibilityOff";

import Spinner from "../../../utils/Spinner";

import { studentSignIn } from "../../../redux/actions/studentActions";

const StudentLogin = () => {

const [translate, setTranslate] = useState(false);

const [username, setUsername] = useState("");

const [password, setPassword] = useState("");

const [showPassword, setShowPassword] = useState(false);

const [loading, setLoading] = useState(false);

const dispatch = useDispatch();

const navigate = useNavigate();

const store = useSelector((state) => state);

const [error, setError] = useState({});

useEffect(() => {

setTimeout(() => {

setTranslate(true);

}, 1000);

}, []);

useEffect(() => {

if (store.errors) {

setError(store.errors);

}

}, [store.errors]);

const login = (e) => {

e.preventDefault();

setLoading(true);

dispatch(

studentSignIn({ username: username, password: password }, navigate)

);

};

useEffect(() => {

if (store.errors) {

setLoading(false);

setUsername("");

setPassword(""); }

}, [store.errors]);

return (

<div className="bg-[#d65158] h-screen w-screen flex items-center justify-center">

<div className="grid grid-cols-2">

<div

className={`h-96 w-96 bg-white flex items-center justify-center ${

translate ? "translate-x-[12rem]" : ""

} duration-1000 transition-all rounded-3xl shadow-2xl`}>

<h1 className="text-[3rem] font-bold text-center">

Student

<br />

Login

</h1>

</div>

<form

onSubmit={login}

className={`${

loading ? "h-[27rem]" : "h-96"

} w-96 bg-[#2c2f35] flex flex-col items-center justify-center ${

translate ? "-translate-x-[12rem]" : ""

} duration-1000 transition-all space-y-6 rounded-3xl shadow-2xl`}>

<h1 className="text-white text-3xl font-semibold">Student</h1>

<div className="space-y-1">

<p className="text-[#515966] font-bold text-sm">Username</p>

<div className="bg-[#515966] rounded-lg w-[14rem] flex items-center">

<input

onChange={(e) => setUsername(e.target.value)}

value={username}

type="text"

required

className="bg-[#515966] text-white px-2 outline-none py-2 rounded-lg placeholder:text-sm"

placeholder="Username" />

</div>

</div>

<div className="space-y-1">

<p className="text-[#515966] font-bold text-sm">Password</p>

<div className="bg-[#515966] rounded-lg px-2 flex items-center">

<input

onChange={(e) => setPassword(e.target.value)}

value={password}

required

type={showPassword ? "text" : "password"}

className=" bg-[#515966] text-white rounded-lg outline-none py-2 placeholder:text-sm"

placeholder="Password"

/>

{showPassword ? (

<VisibilityIcon

onClick={() => setShowPassword(!showPassword)}

className="cursor-pointer"

/>

) : (

<VisibilityOffIcon

onClick={() => setShowPassword(!showPassword)}

className="cursor-pointer" />

)}

</div>

</div>

<button

type="submit"

className="w-32 hover:scale-105 transition-all duration-150 rounded-lg flex items-center justify-center text-white text-base py-1 bg-[#04bd7d]">

Login

</button>

{loading && (

<Spinner

message="Logging In"

height={30}

width={150}

color="#ffffff"

messageColor="#fff"

/>

)}

{(error.usernameError || error.passwordError) && (

<p className="text-red-500">

{error.usernameError || error.passwordError}

</p>

)}

</form>

</div>

</div>

);

};

export default StudentLogin;

**3.4.5 Index.js:**

The provided index.js code sets up a server using Express.js, establishes a connection to a MongoDB database, and defines routes for different entities (admin, faculty, and student). It utilizes middleware like bodyParser and cors for handling requests and enables API endpoints for each entity. The server

listens on a specified port and displays a message when it is running successfully.

import express from "express";

import bodyParser from "body-parser";

import mongoose from "mongoose";

import cors from "cors";

import dotenv from "dotenv";

import adminRoutes from "./routes/adminRoutes.js";

import studentRoutes from "./routes/studentRoutes.js";

import facultyRoutes from "./routes/facultyRoutes.js";

const app = express();

dotenv.config();

app.use(bodyParser.json({ limit: "30mb", extended: true }));

app.use(bodyParser.urlencoded({ limit: "30mb", extended: true }));

app.use(cors());

app.use("/api/admin", adminRoutes);

app.use("/api/faculty", facultyRoutes);

app.use("/api/student", studentRoutes);

const PORT = process.env.PORT || 5000;

app.get("/", (req, res) => {

res.send("Hello to college erp API");});

mongoose

.connect(process.env.CONNECTION\_URL, {

useNewUrlParser: true,

useUnifiedTopology: true, })

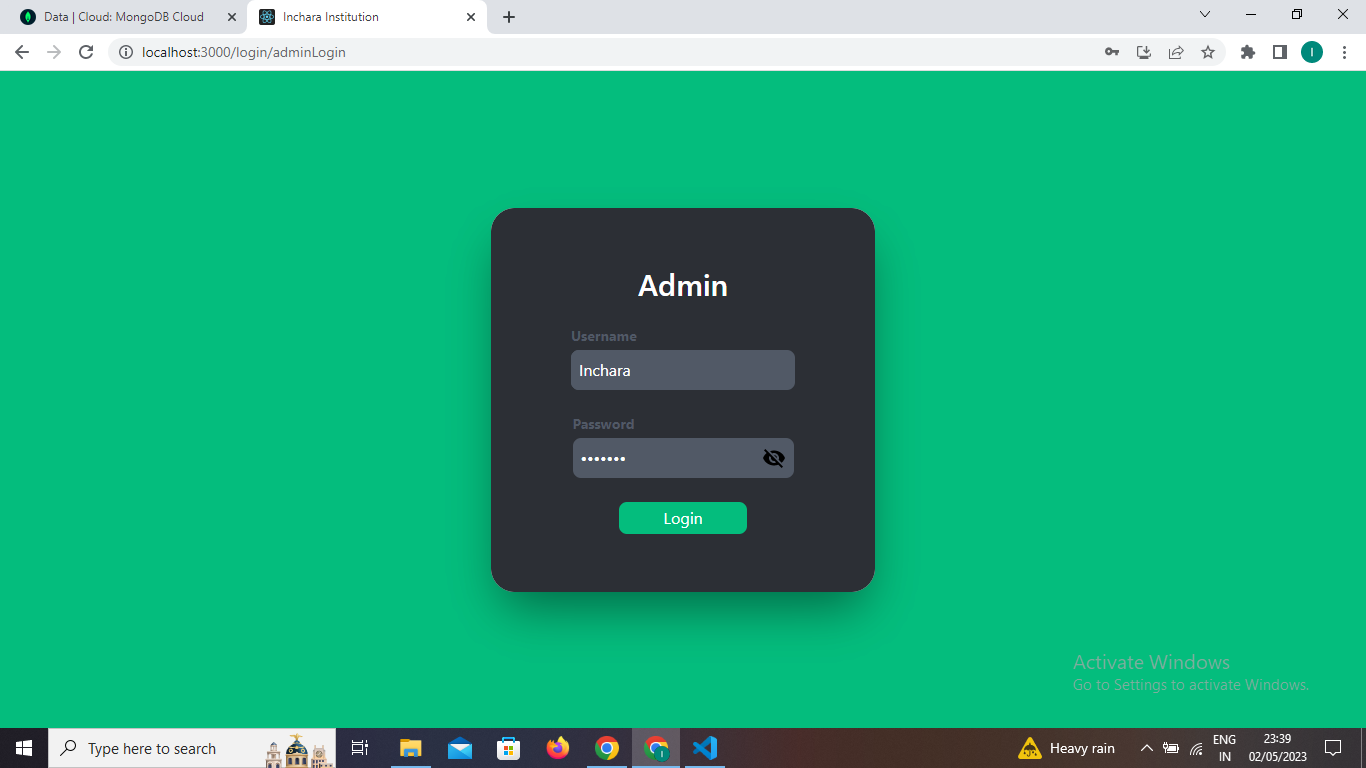
.then(() =>

app.listen(PORT, () => console.log(`Server running on port ${PORT}`)) )

.catch((error) => console.log("Mongo Error", error.message));

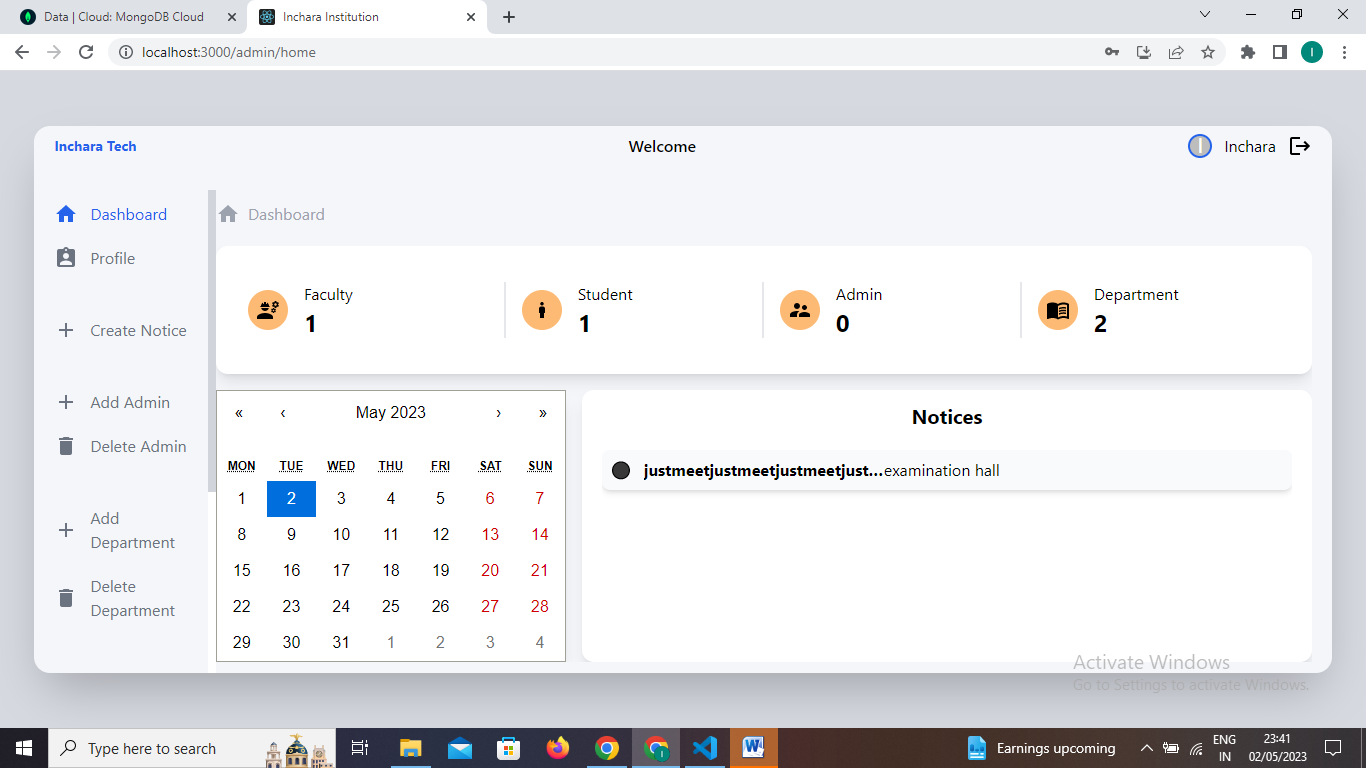
**3.4 Snapshots**

**Admin login Page:** The admin login page allows admins to enter their username and password, which are then submitted to the server for authentication. If the credentials are valid, the admin is granted access to the admin home page; otherwise, admin doesn’t exist error msg is displayed.



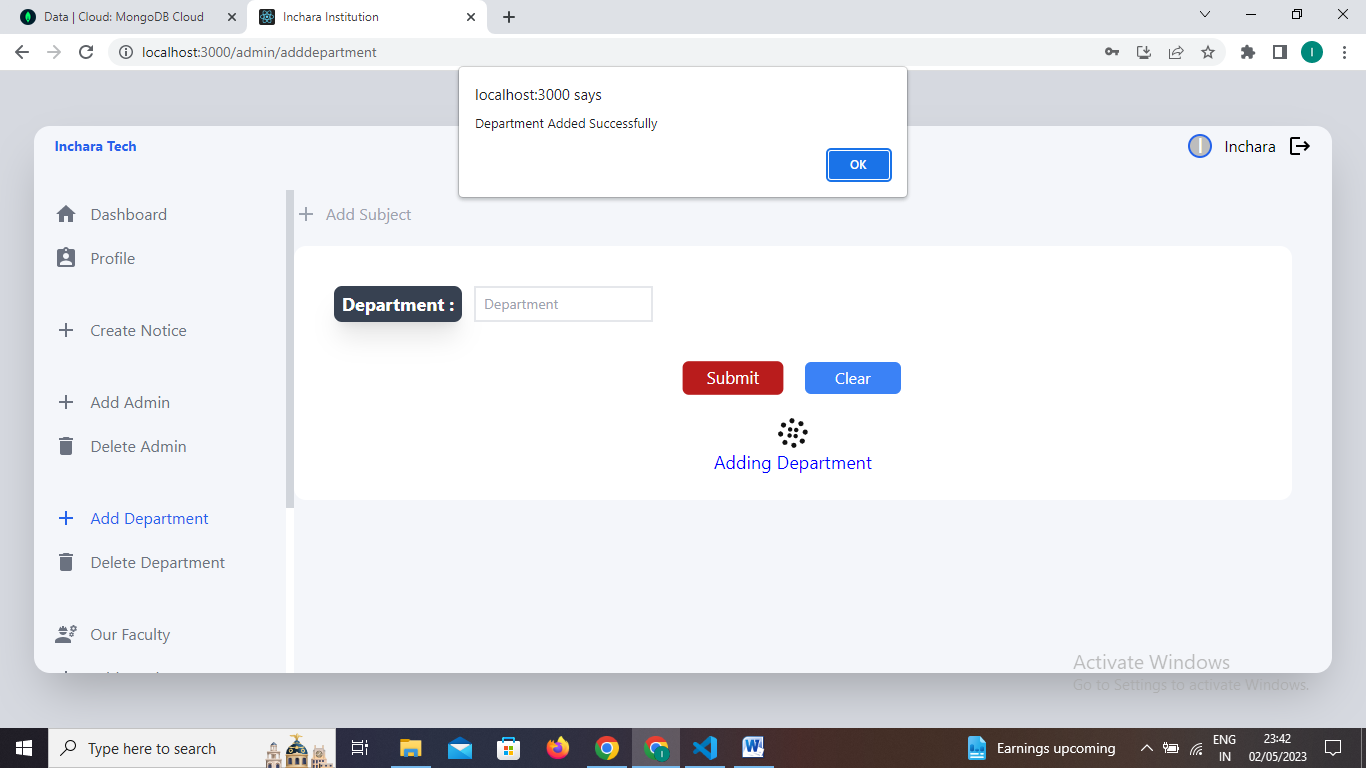
**Figure 3.4.1: Admin Login Page**

**Admin Home page:** The admin home page is a dashboard that provides access to various administrative functionalities and information. It typically includes features such as managing users, departments, faculties, students, subjects, creating notices, and viewing data related to the institution**.**



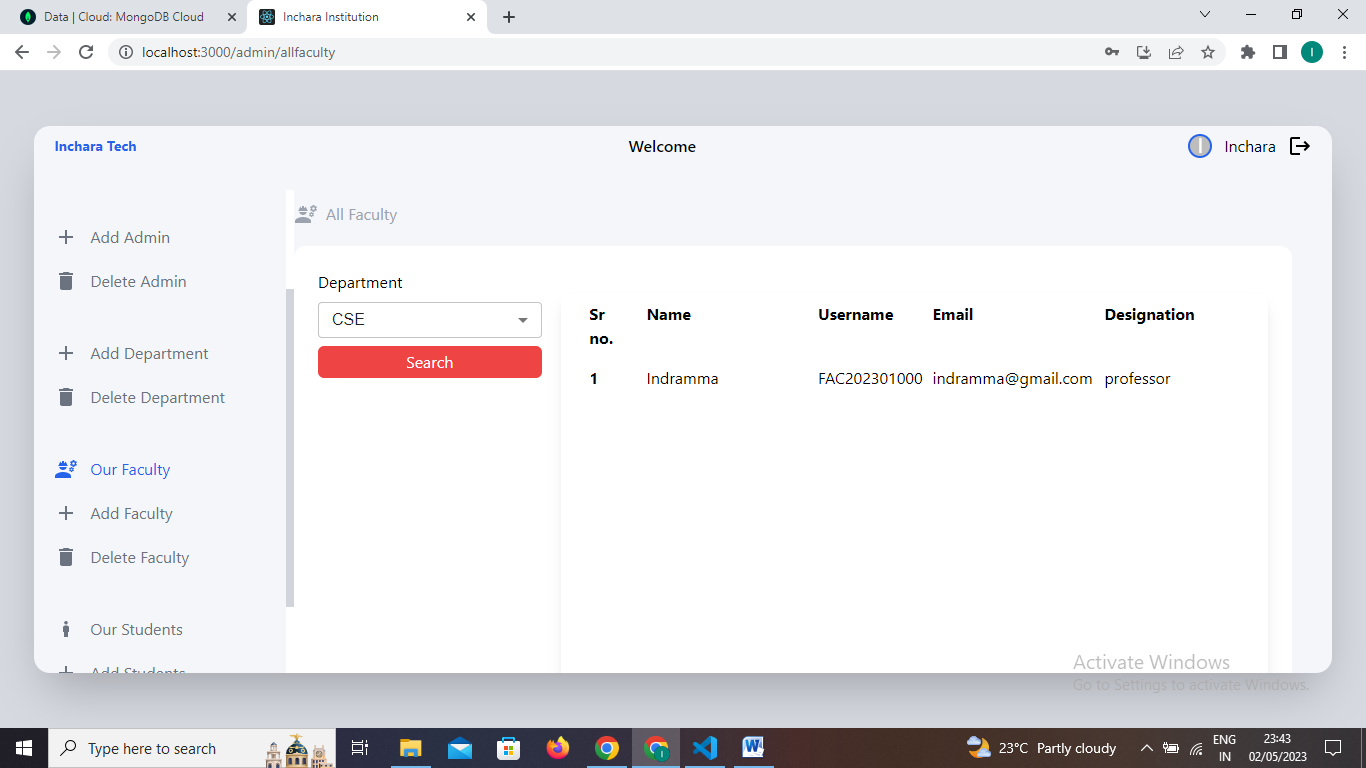
**Figure 3.4.2: Admin Home Page**

**Add Department:** The "admin add department" functionality allows the admin to create and add a new department to the system or database.



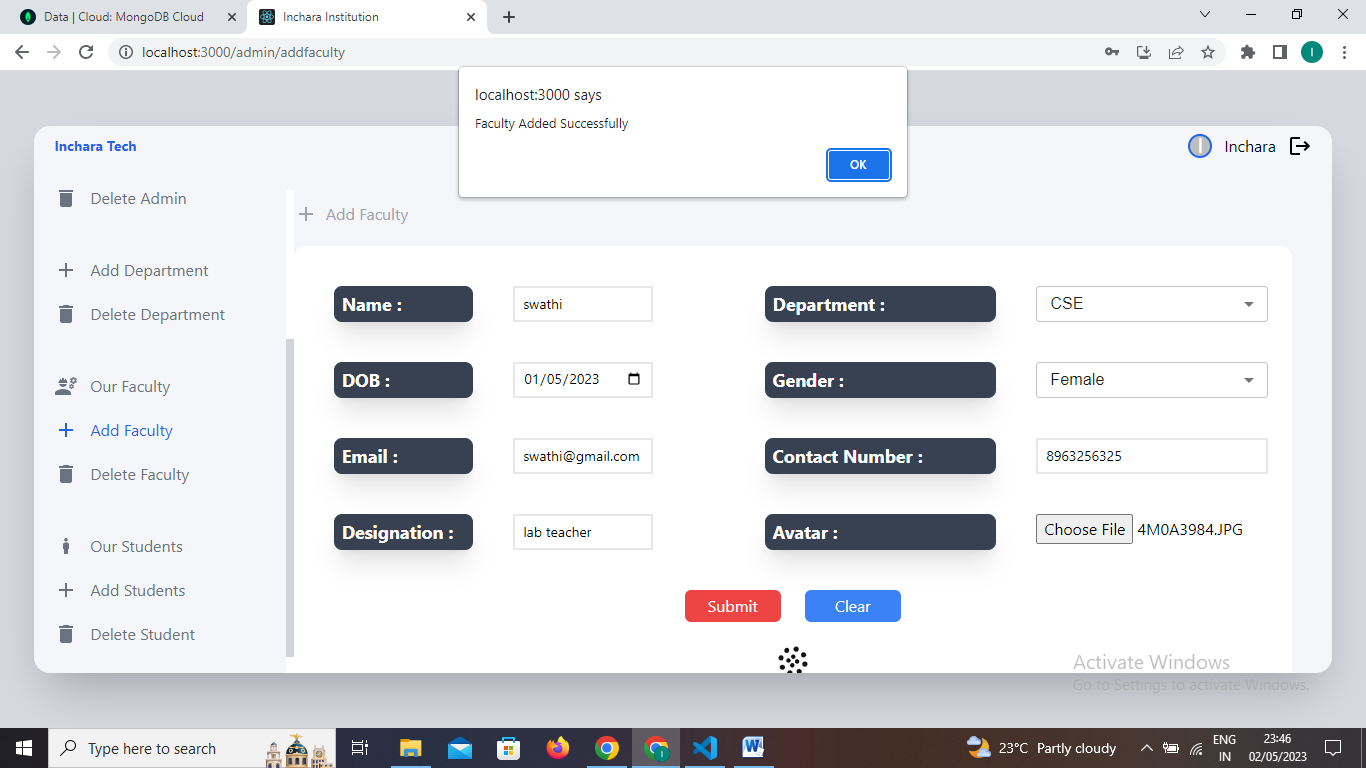
**Figure 3.4.3: Add Department**

**Our Faculty:** In the "Our Faculty" section of the admin page, when selecting the Computer Science and Engineering (CSE) department, it displays the list of faculties who are associated with the CSE department. This feature provides a convenient way to view and access the specific faculty members belonging to the selected department, facilitating effective communication and management within the CSE department.



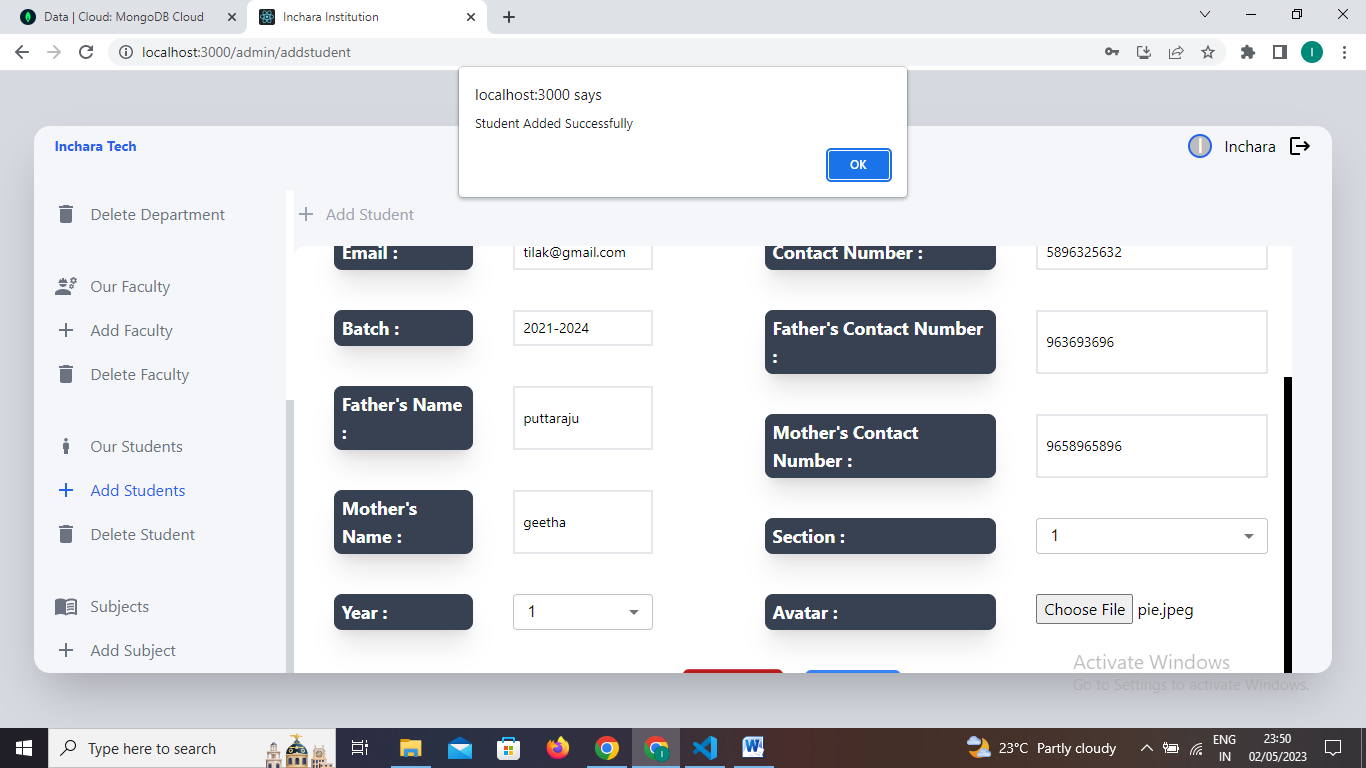
**Figure 3.4.4: Our Faculty**

**Add Faculty:** In the admin page, the "Add Faculty" feature allows admin to add new faculty members to the college or institution. Admin can input relevant information such as name, department, contact details, and other necessary details for the new faculty member, enabling seamless onboarding and management of faculty resources.



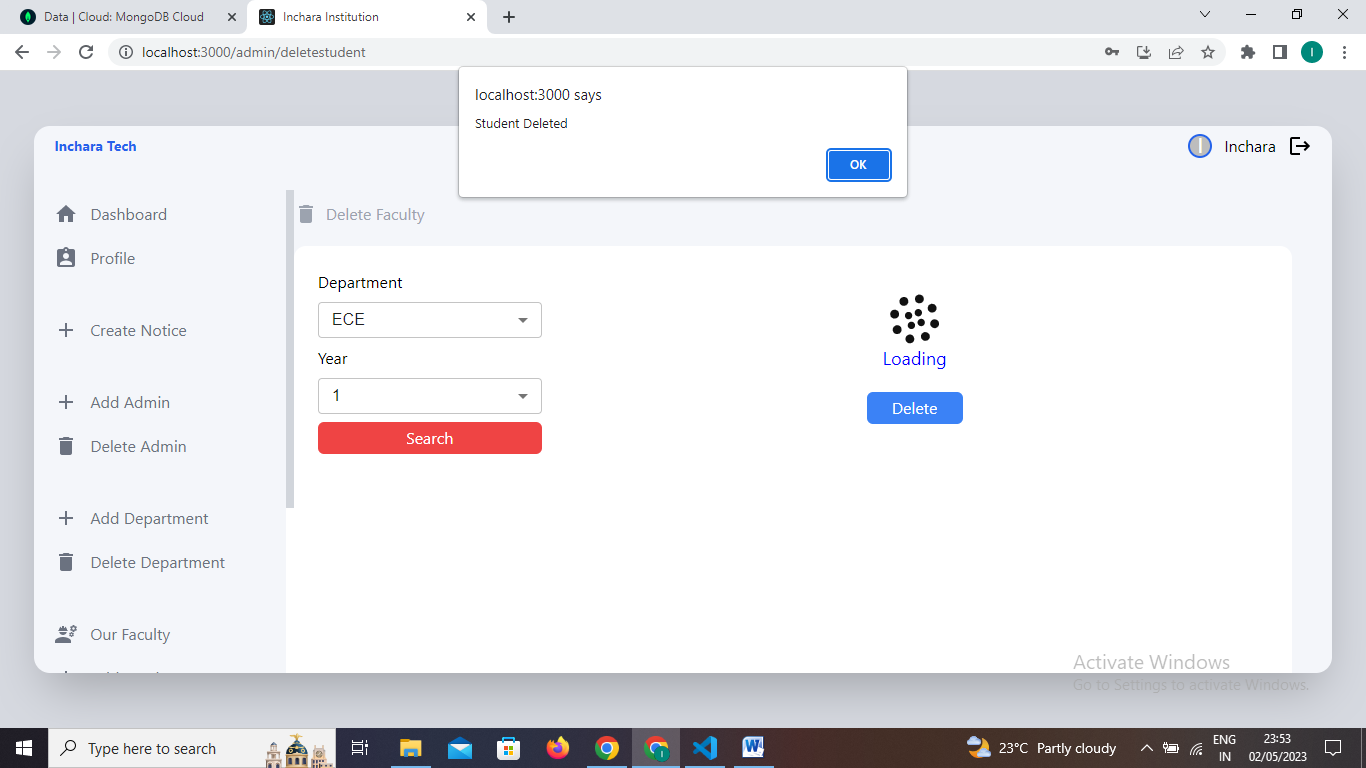
**Figure 3.4.5: Add Faculty**

**Add Student:** In the admin page, the "Add Students" feature allows admin to add new students to the system by providing their relevant information such as name, roll number, department, and other details. This helps in maintaining a comprehensive database of students and facilitates their management within the institution.



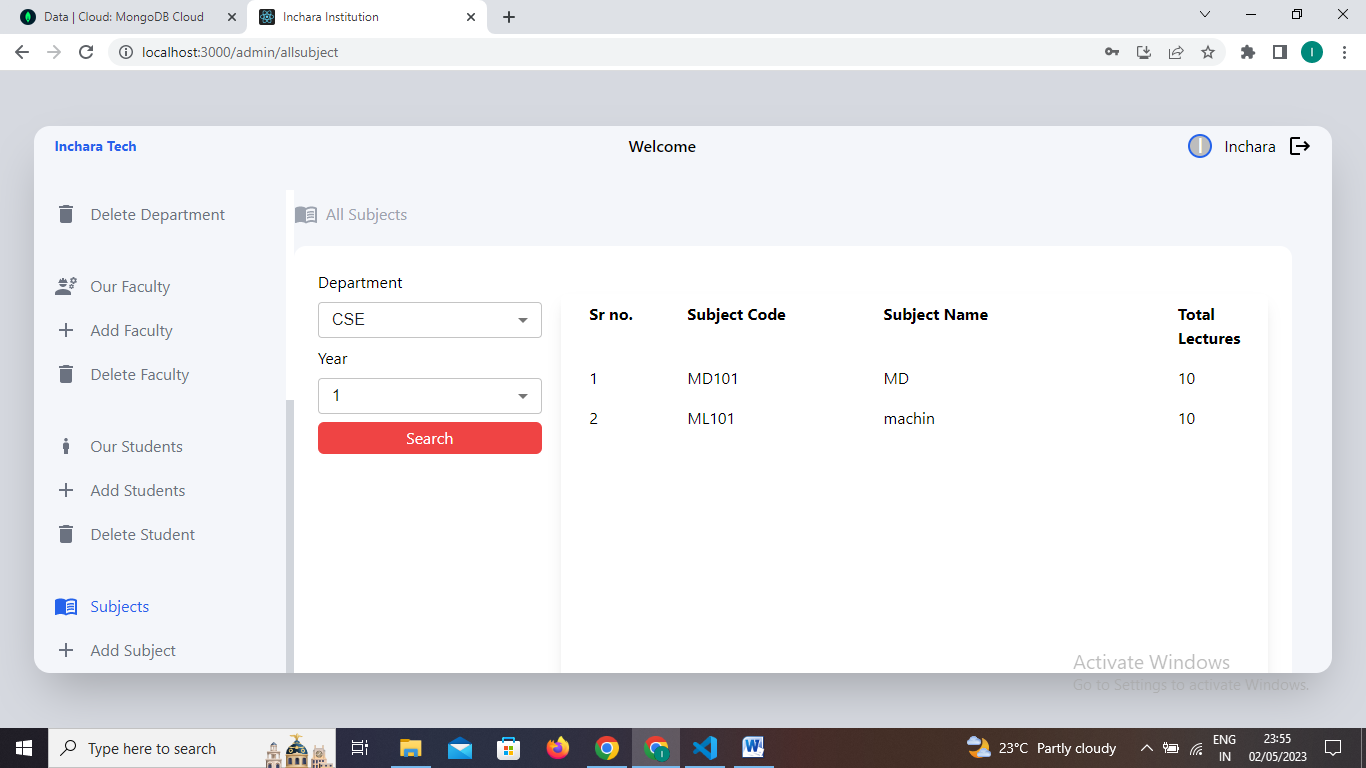
**Figure 3.4.6: Add Student**

**Delete Student:** In the admin page, the "Delete Student" feature allows admin to delete students from the system based on their department or branch. By selecting a specific department, admin can view and delete students belonging to that particular branch, helping to manage and update the student records effectively.



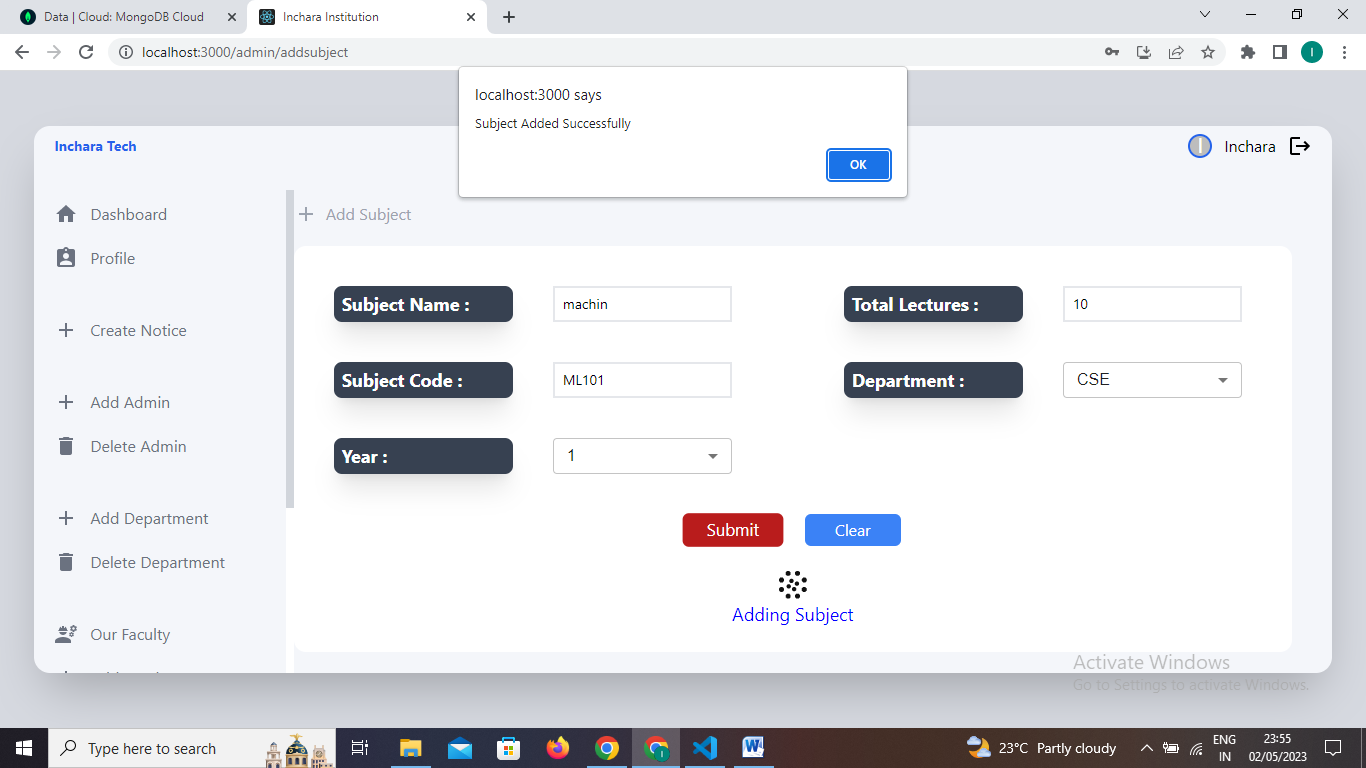
**Figure 3.4.7: Delete Student**

**Subject:** In the admin page, the "View Subjects" feature provides admin with a list of subjects or courses offered in the institution. This feature allows administrators to view and manage the subjects available, including adding new subjects, modifying existing ones, or removing subjects from the curriculum as needed.



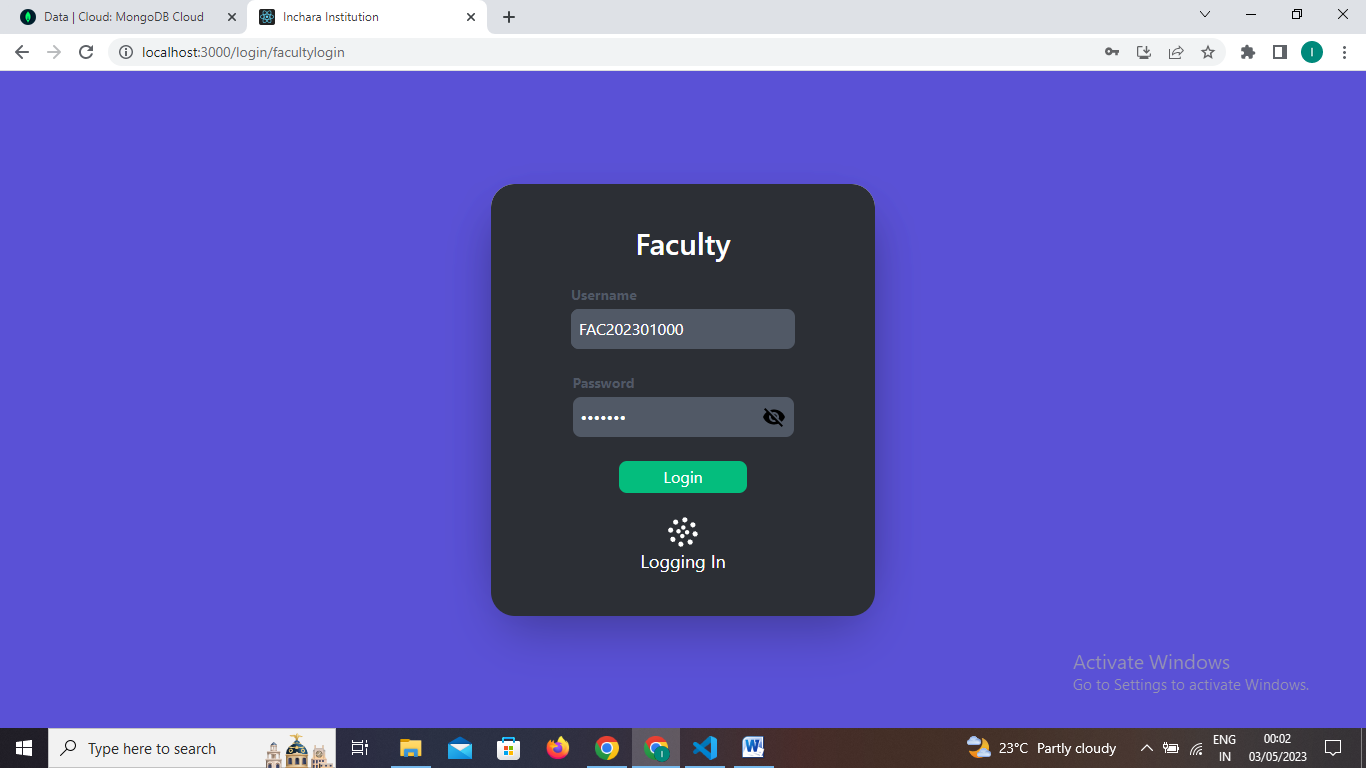
**Figure 3 4.8: Subjects**

**Add Subject**: In the admin page, the "Add Subject" feature allows admin to input various fields related to a subject, such as the subject name, year, subject code, number of lectures, and department. This feature enables admin to create and add new subjects to the curriculum, providing essential information for effective subject management within the institution.



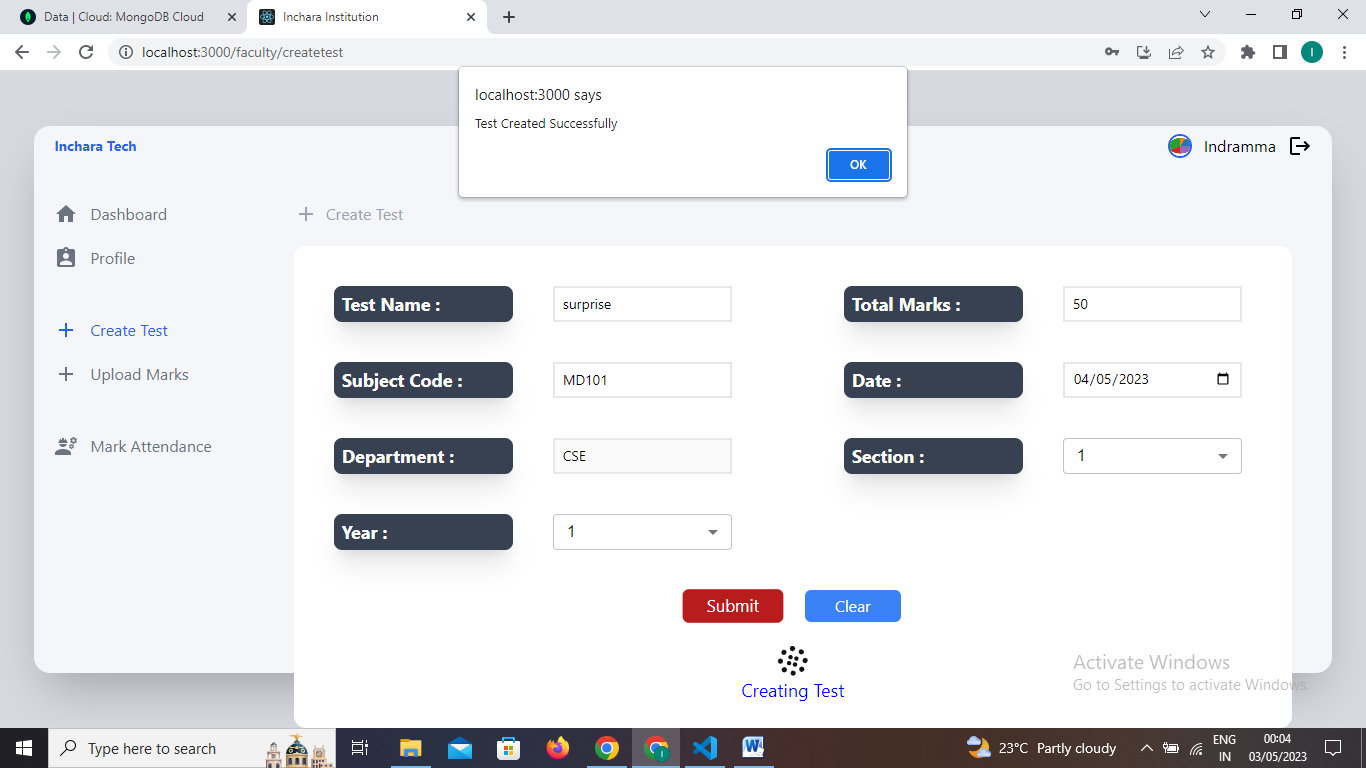
**Figure 3.4 9 Add Subject**

**Faculty login page:** The Faculty login page allows admins to enter their username and password, which are then submitted to the server for authentication. If the credentials are valid, the faculty is granted access to the faculty home page; otherwise, faculty doesn’t exist error msg is displayed.



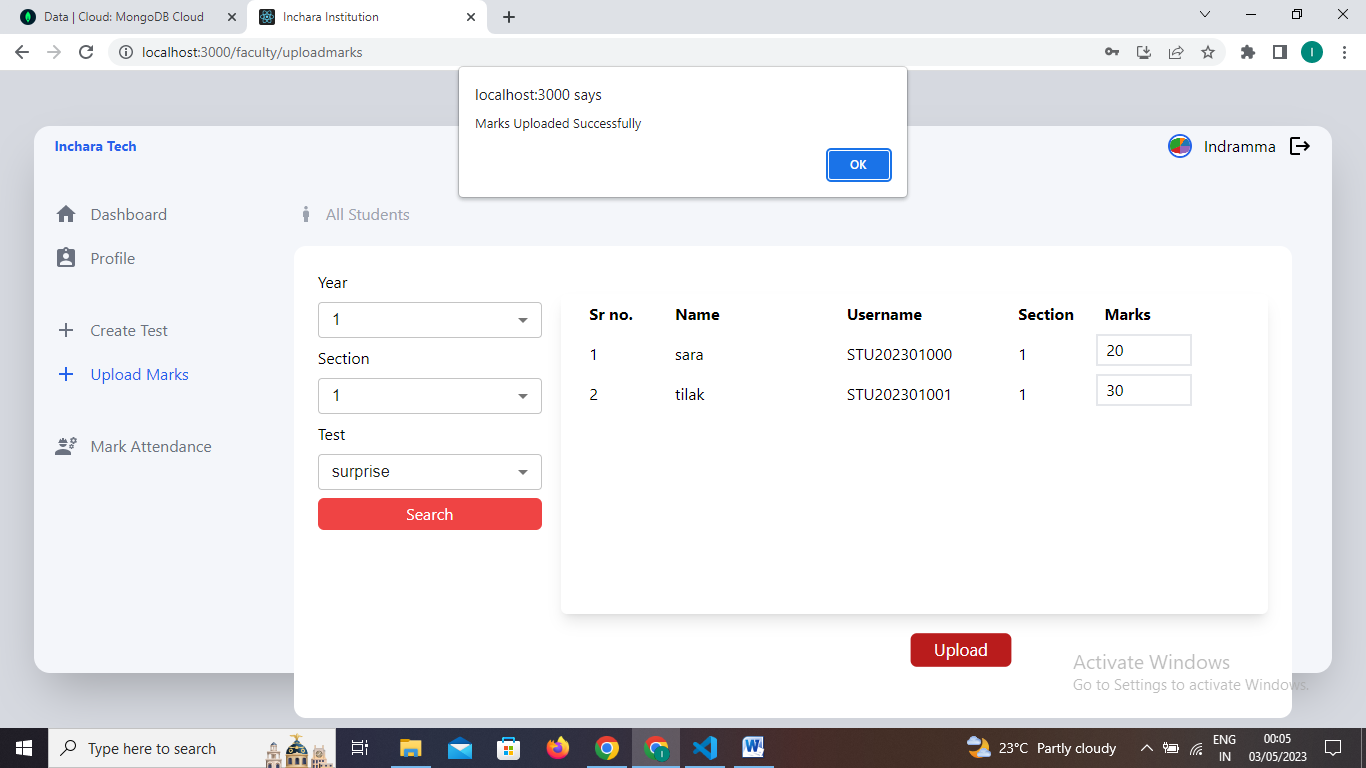
**Figure 3.4.10: Faculty Login Page**

**Faculty create Test**: Faculty can create tests for students by providing the following details in the test creation fields: test name (a name to identify the test), subject code (code specific to the subject being assessed), year (academic year or semester), department (the department to which the test belongs), total marks (the maximum score achievable for the test), and section date (the date on which the test will be conducted). These fields enable faculty to design and schedule tests for student evaluation.



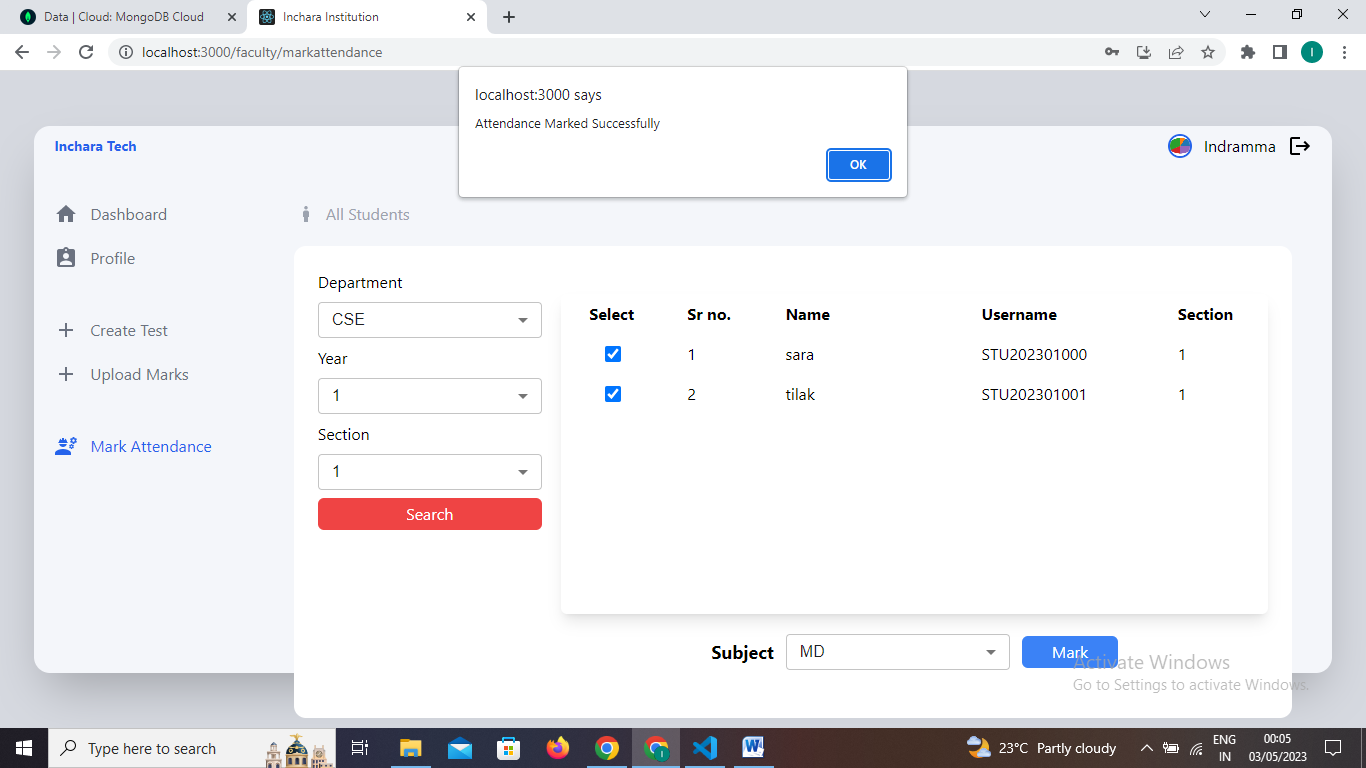
**Figure 3.4.11: Faculty Create Test**

**Upload Marks:** Faculty can upload marks for students by entering the student's details, such as student ID or name, along with their corresponding marks in the upload marks section. This allows faculty members to accurately record and update the performance of students in various assessments or examinations.



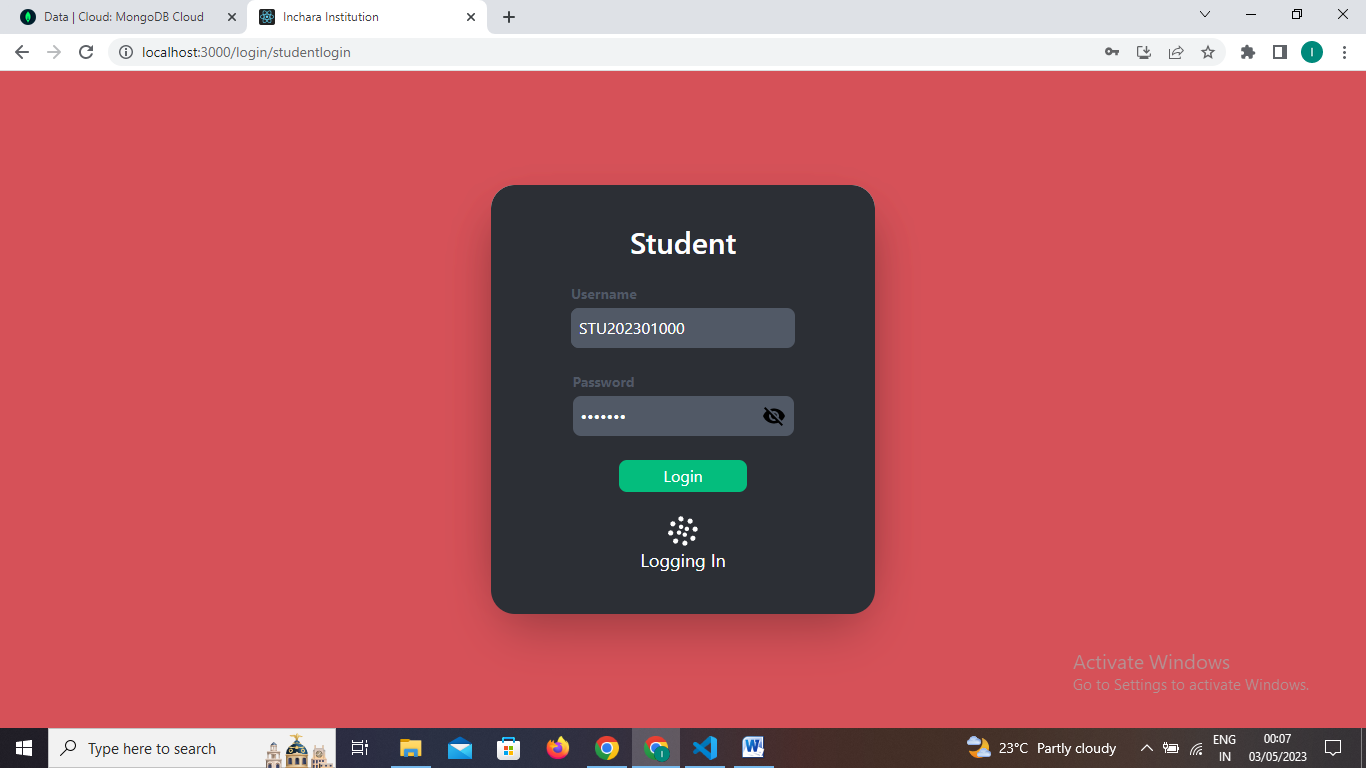
**Figure 3.4.12: upload Marks**

**Mark Attendance:** Faculty can mark attendance for students by selecting the respective class or session and indicating the presence or absence of each student. This helps in keeping track of student attendance records and monitoring their attendance patterns.



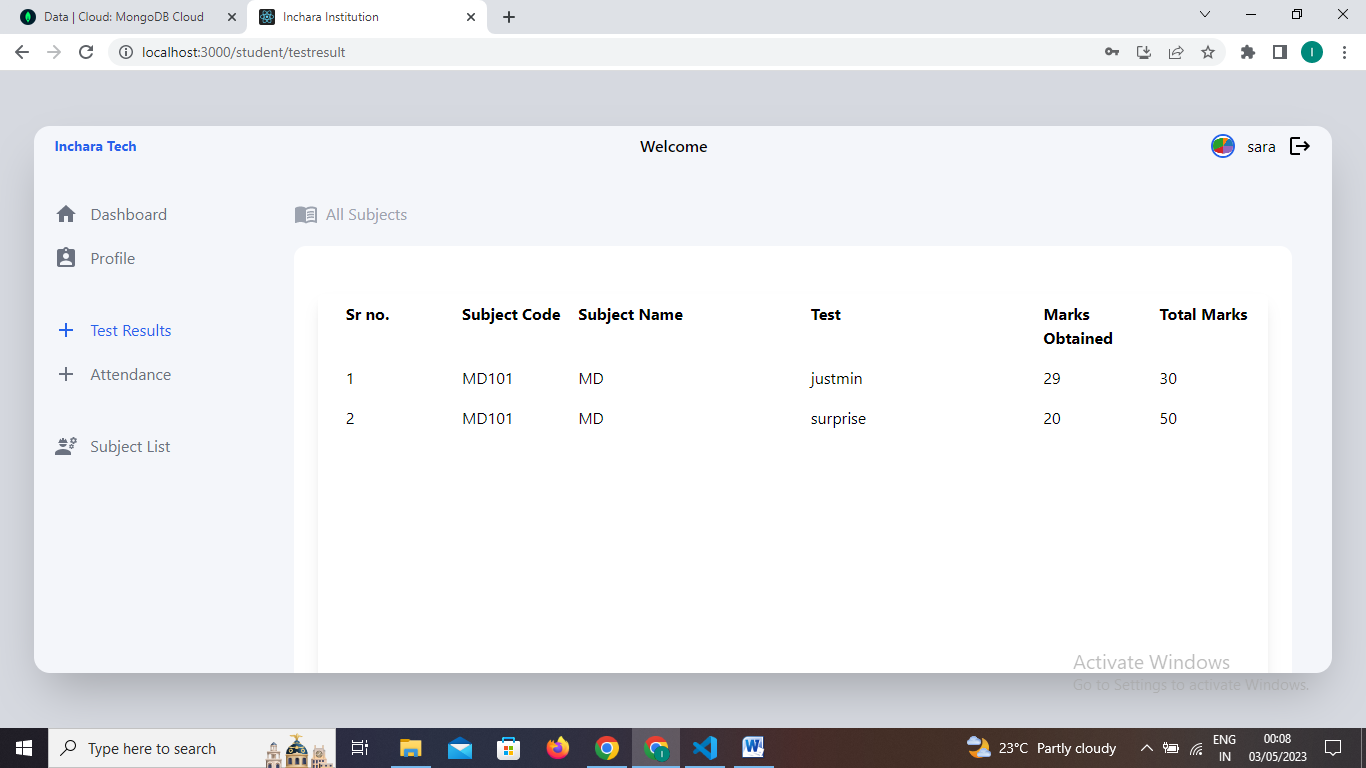
**Figure 3.4.13: Mark attendance**

**Student Login page:** The student login page allows students to authenticate themselves by entering their username and password. Once logged in, students can access their personalized account and avail various features and information specific to their student profile.



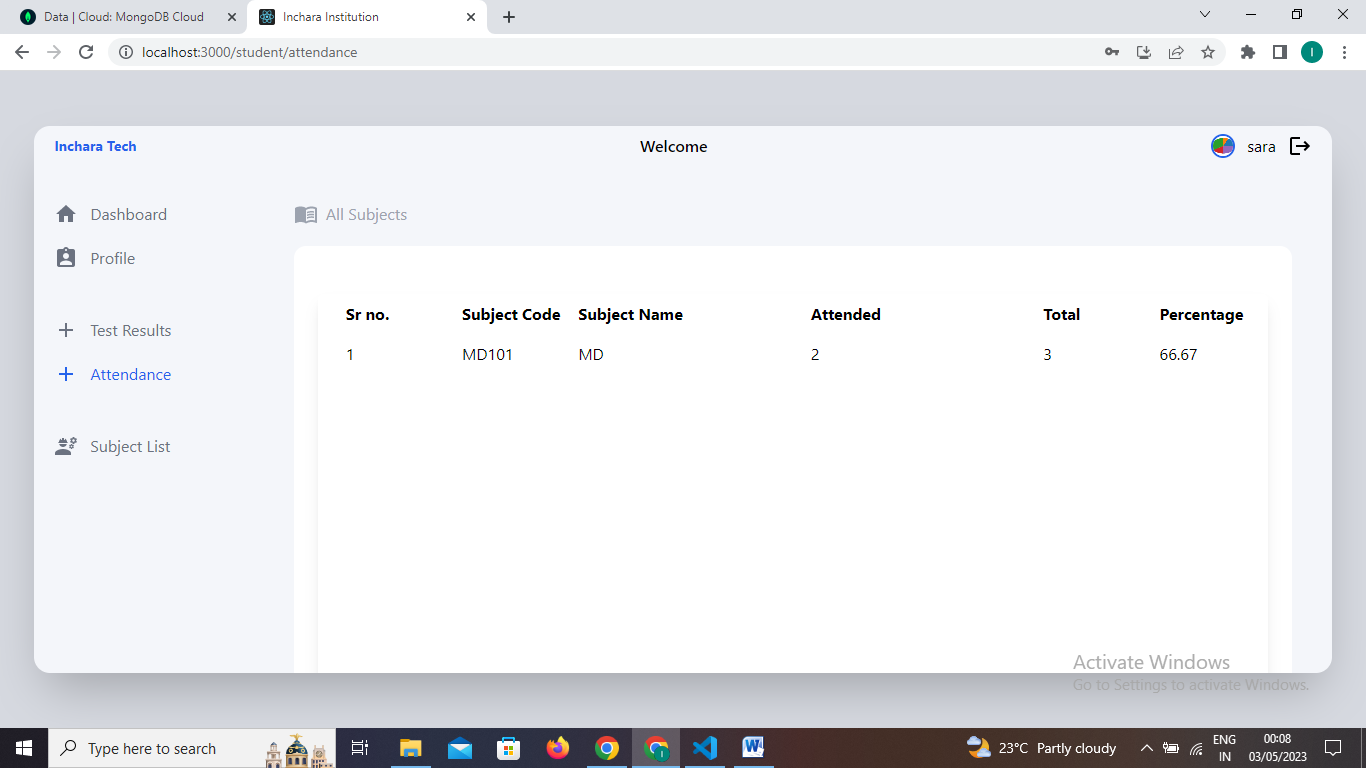
**Figure 3.4.14: Student Login Page**

**Test Results:** In the test results, students can view the marks or scores they have obtained in each subject or test they have taken. This allows them to assess their performance and gauge their understanding and progress in specific subjects. The marks serve as a measure of their academic achievement and can help identify areas for improvement or areas of strength.



**Figure 3.4.15: Test Result**

**Attendance:** In the attendance section, students can view their attendance records for each subject. The displayed information typically includes the subject name and code, the number of classes attended by the student, the total number of classes held for that subject, and the attendance percentage calculated based on these values. This allows students to track their attendance progress, assess their attendance performance in each subject, and maintain an overview of their overall attendance percentage.



**Figure 3.4.16: Attendance**

**Chapter 4**

**RESULT**

In the college ERP project, the admin have the capability to manage and maintain the system by adding or deleting faculty members, departments, and students. Faculty members have the authority to create tests for students, upload marks, and mark attendance. Students can log in using their credentials and access their test results, attendance records, and subject lists. This comprehensive system facilitates efficient management of academic activities, promotes transparency, and enables effective communication between faculty and students.

**Chapter 5**

**CONCLUSION AND FUTURE SCOPE**

**5.1 Conclusion**

In conclusion, the college ERP system provides a centralized platform for admin, faculty, and students to manage various aspects of academic operations. It streamlines administrative tasks such as faculty and student management, while empowering faculty members to create tests, manage marks, and track attendance. Students benefit from easy access to their test results, attendance records, and subject information. Overall, the college ERP system enhances efficiency, communication, and transparency within the educational institution.

The college ERP sysytem is an important and useful system for managing various aspects of a college's operations. Its implementation can help improve the overall functioning of a college, and enable stakeholders to better track and manage academic information.

**5.2 Future Scope**

The future scope of the college ERP project includes potential enhancements and expansions such as:

1. Integration with other systems: The ERP can be integrated with other systems like library management, financial management, or hostel management systems to create a comprehensive solution for the entire institution.

2. Mobile application development: Developing a mobile application version of the ERP can enhance accessibility and convenience for administrators, faculty, and students, allowing them to access information and perform tasks on the go.

3. Advanced analytics and reporting: Implementing advanced analytics and reporting capabilities can provide valuable insights into student performance, attendance patterns, and overall institutional performance, aiding in data-driven decision making.

4. Communication and collaboration features: Adding features like discussion forums, chat functionality, and document sharing capabilities can facilitate better communication and collaboration among administrators, faculty, and students.

5. Integration with external platforms: Integration with external platforms such as online learning management systems or job portals can enhance the functionality and value of the ERP system.

6. Automated workflows and notifications: Implementing automated workflows and notifications for tasks such as fee payment reminders, assignment submission deadlines, or exam schedules can improve

efficiency and reduce manual intervention.

7. Continuous improvement and updates: Regular updates, bug fixes, and incorporating user feedback are crucial for the long-term success and usability of the college ERP system.

Overall, the future scope of the college ERP project lies in continuous improvement, expansion of features, and adapting to emerging technologies and requirements in the education sector.

**REFERENCE**

* <http://www.cloudinary.com/>
* <https://reactjs.org/>
* <https://www.mongodb.com/>
* <https://nodejs.org/en/>
* <https://tailwindcss.com/>
* [Redux Essentials, Part 1: Redux Overview and Concepts | Redux](https://redux.js.org/tutorials/essentials/part-1-overview-concepts)