STAFFSPHERE

Major project submitted in partial fulfillment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Under the esteemed guidance of

Dr G Krishna Lava Kumar
Associate Professor

By

NEHA MUNUGANTI (21R11A05D7)

Department of Computer Science and Engineering

Geethanjali College of Engineering and Technology (Autonomous)

Accredited by NAAC with A+ Grade: B.Tech. CSE, EEE, ECE accredited by NBA Sy. No: 33 & 34, Cheeryal (V), Keesara (M), Medchal District, Telangana – 501301

MAY – 2025

Geethanjali College of Engineering and Technology (Autonomous)

Accredited by NAAC with A+ Grade : B.Tech. CSE, EEE, ECE accredited by NBA Sy. No: 33 & 34, Cheeryal (V), Keesara (M), Medchal District, Telangana – 501301

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the B.Tech Major Project report entitled **"STAFF SPHERE"** is a bonafide work done by, **Neha Munuganti(21R11A05D7),** in partial fulfillment of the requirement of the award for the degree of Bachelor of Technology in "**Computer Science and Engineering**" from Jawaharlal Nehru Technological University, Hyderabad during the year 2024-2025.

Dr. G Krishna Lava Kumar
Associate Professor

HoD – CSE
Dr E Ravindra
Professor

External Examiner

Geethanjali College of Engineering and Technology (Autonomous)

Accredited by NAAC with A+ Grade : B.Tech. CSE, EEE, ECE accredited by NBA Sy. No: 33 & 34, Cheeryal (V), Keesara (M), Medchal District, Telangana – 501301

Department of Computer Science and Engineering

DECLARATION BY THE CANDIDATE

I, **Neha Munuganti bearing Roll No. 21R11A05D7,** hereby declare that the project report entitled **"STAFF SPHERE"** is done under the guidance of **Mr. G Krishna Lava Kumar**, **Associate Professor**, Department of Computer Science and Engineering, Geethanjali College of Engineering and Technology, is submitted in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Computer Science and Engineering**.

This is a record of bonafide work carried out by me and the results embodied in this project have not been reproduced or copied from any source. The results embodied in this project report have not been submitted to any other University or Institute for the award of any other degree or diploma.

Neha Munuganti (21R11A05D7)
Department of CSE, Geethanjali College of Engineering and Technology,
Cheeryal.

ACKNOWLEDGEMENT

It is with profound gratitude and sincere appreciation that I extend my heartfelt thanks to all those who have played a significant role in the successful completion of this undergraduate project.

First and foremost, I express my deep sense of respect and gratitude to our **Honourable Chairman, Mr. G. R. Ravinder Reddy**, for his constant encouragement and for nurturing a culture of academic excellence and innovation within the institution.

I would also like to express my sincere thanks to **Dr. S. Udaya Kumar**, **Director** , for his visionary leadership and continued support, which have provided the ideal environment and motivation for academic pursuits such as this project.

My heartfelt appreciation goes to **Dr. K. Sagar**, **Principal,** for his steadfast guidance, infrastructural support, and encouragement that have helped bring this project to successful fruition.

I am deeply grateful to **Dr. E. Ravindra**, **Head of the Department of Computer Science and Engineering**, for his academic leadership, valuable feedback, and continuous support throughout the duration of this project.

I extend my sincere thanks to the **Project Coordinators, Mr. S Durga Prasad, Mr P Krishna Rao**, for their organized planning, timely guidance, and constructive feedback which contributed immensely to the smooth and effective completion of the seminar.

A special note of appreciation is reserved for my guide, **Dr.G Krishna Lava Kumar, Associate Professor**, for his valuable insights, scholarly advice, and unwavering mentorship throughout the research and presentation phases of this project.

Lastly, I am ever grateful to my **parents and family** for their unconditional love, encouragement, and moral support. Their faith in me has been my greatest strength throughout this journey.

With genuine appreciation, I acknowledge every individual who has, in one way or another, contributed to the successful completion of this project.

With warm regards,
Neha Munuganti (21R11A05C6)
Department of Computer Science and Engineering
Geethanjali College of Engineering and Technology

ABSTRACT

In academic institutions, maintaining and managing faculty information is traditionally a manual and time-consuming process, often leading to data inconsistencies, inefficiencies, and difficulties during audits and reporting. To address these challenges, StaffSphere has been developed as a comprehensive, web-based platform for centralized faculty data management.StaffSphere enables faculty members to register themselves by entering their personal, professional, qualification, and additional details, along with uploading necessary certificates and documents. This information is securely stored in a MongoDB database for easy retrieval and management. Faculty can also interact with a chatbot to retrieve specific information and request profile edits, which are processed through an approval system managed by administrators.Administrators are provided with a dedicated dashboard that allows them to filter faculty data based on various criteria, generate visual insights through Chart.js graphs, and export selected information into Excel (.xlsv) reports. The system is designed with role-based access control, ensuring secure operations between faculty and admin users.Developed using React.js for the frontend, Node.js and Express.js for the backend, and MongoDB for database management, StaffSphere offers a scalable, responsive, and efficient solution for academic institutions to manage faculty profiles digitally. By automating manual processes, improving data accuracy, and supporting real-time analysis, StaffSphere significantly enhances operational efficiency and administrative reporting capabilities.

List of figures

List of Tables

Table of Contents

CHAPTER 1 INTRODUCTION

1.1 OVERVIEW OF THE PROJECT

StaffSphere is a comprehensive, web-based platform designed to streamline and centralize the management of faculty information in academic institutions. It addresses the growing need for an efficient and structured system to handle faculty registration, profile updates, document uploads, and administrative reporting. The platform is developed using a modern and scalable technology stack comprising React.js for the frontend, Node.js for the backend, and MongoDB as the database.

The primary objective of StaffSphere is to reduce manual workload and data duplication by providing faculty members with a guided registration process. Through a step-by-step interface, users can enter their personal details, professional qualifications, educational background, and work experience. The system also allows faculty to upload supporting documents such as certificates and ID proofs, which are securely stored and managed in the database.

From the administrative perspective, StaffSphere provides a powerful dashboard that offers features like custom data filtering, Excel report generation, and dynamic visualizations through pie charts. These tools help administrators gain quick insights into faculty distribution based on various attributes such as department, qualification, or experience. The system ensures that all collected data is easily retrievable, well-organized, and ready for institutional analysis or audit requirements.

Designed with user experience and scalability in mind, StaffSphere ensures secure data handling, intuitive navigation, and flexible modules that can be enhanced over time. It improves the overall efficiency of faculty management, providing both faculty members and administrators with a unified, reliable platform.

In summary, StaffSphere transforms traditional faculty management into a digital, organized, and data-driven process, supporting institutions in maintaining accurate records and making informed decisions with ease.

1.2 PROBLEM STATEMENT

In most academic institutions, the process of managing faculty information is still predominantly manual, fragmented, and inefficient. Faculty members are required to maintain their records individually, often using spreadsheets or paper-based documentation, which not only leads to inconsistencies but also makes the process of updating information cumbersome. Important academic and professional details such as qualifications, certifications, research publications, teaching experience, and career achievements are frequently scattered across different systems or formats, making it difficult to maintain a centralized and up-to-date profile.

This disorganized approach presents significant challenges for administrators as well. During critical processes like faculty appraisals, promotions, accreditation audits, institutional reviews, or research collaborations, retrieving accurate and complete faculty data becomes a time-consuming task. The absence of an integrated system often results in duplication of efforts, data mismatches, and a lack of reliable insights. Furthermore, the inability to analyze data efficiently hinders the institution's ability to make informed decisions or present verified information when required by regulatory bodies or external stakeholders.

In addition, institutions lack the tools to visually interpret faculty-related data, such as department-wise distribution, qualification levels, or teaching experience breakdowns. This limitation affects long-term strategic planning and resource allocation. As institutions strive to improve quality, transparency, and efficiency, the need for a digital solution that simplifies faculty data management and analysis becomes increasingly evident.

Therefore, it is essential to develop a unified, secure, and user-friendly platform that can automate the faculty registration process, organize academic and professional data, enable document uploads, and provide administrative access for reporting and analytics. Such a system would not only enhance institutional productivity but also empower faculty to maintain their academic journey in a structured and accessible manner.

1.3 OBJECTIVES OF THE PROJECT

- **Streamline Faculty Profile Management:** Allow easy registration and updates of personal and professional details.
- **Automate Profile Update Notifications:** Implement fortnightly notifications for faculty to update missing or incomplete profile information.
- **Facilitate Faculty Collaboration:** Incorporate a chatbot for faculty to connect with colleagues sharing similar research interests.
- **Provide Interest-Based Notifications:** Notify faculty about relevant seminars, webinars, and ongoing activities.
- **Simplify Administrative Tasks:** Enable administrators to filter faculty profiles and generate Excel reports for institutional use.
- **Visualize Faculty Data:** Integrate tools like pie charts for analyzing faculty data and aiding decision-making.
- **Enhance Communication Efficiency:** Allow administrators to send timely notifications to faculty members.
- **Leverage Modern Web Technologies:** Use React.js, Node.js, and MongoDB for a scalable and robust platform.
- **Ensure Data Security:** Protect sensitive faculty information through secure data storage and access protocols.
- **Support Long-Term Growth:** Provide an adaptable solution that scales with institutional needs.

1.4 SCOPE OF THE PROJECT

The StaffSphere project aims to provide a centralized, web-based solution for managing faculty data within academic institutions. Its scope includes the design, development, and deployment of a platform that digitizes the complete faculty registration and data handling process, addressing the inefficiencies and fragmentation of traditional systems. The system ensures that all faculty-related information—ranging from personal and professional details to academic qualifications and uploaded documents—is securely stored and easily accessible.

The platform is designed for two primary user roles: faculty members and administrators. Faculty members will have access to a guided, section-wise registration process where they can enter and update their information in a structured format. Administrators, on the other hand, will have access to powerful tools for data filtering, export, and analysis.

The project covers the following core functionalities:

- **Faculty Registration**: Step-by-step input of personal details, qualifications, certifications, experience, and achievements.
- **Document Uploads**: Secure uploading and storage of certificates, ID proofs, and other academic documents.
- **Admin Access**: Ability to filter faculty data using checkboxes and export selected information in Excel format.
- **Data Visualization**: Generation of interactive pie charts for visual analysis of faculty distribution by department, qualification, etc.

The platform is built using a robust tech stack:

- **Frontend**: React.js for building a user-friendly interface.
- **Backend**: Node.js with Express to handle requests and logic.
- **Database**: MongoDB for scalable and flexible data storage.

1.5 METHODOLOGY

The development of the StaffSphere project followed the Waterfall Software Development Life Cycle (SDLC) Model. This model was chosen due to its straightforward, phase-by-phase approach that is well-suited for academic projects with clearly defined requirements and deliverables.

In the Waterfall model, the software development process flows sequentially through a series of predefined stages. Each phase must be completed before moving on to the next, and there is typically no overlap. This linear methodology ensured a disciplined development process where all team members had a clear understanding of goals and timelines at each stage.

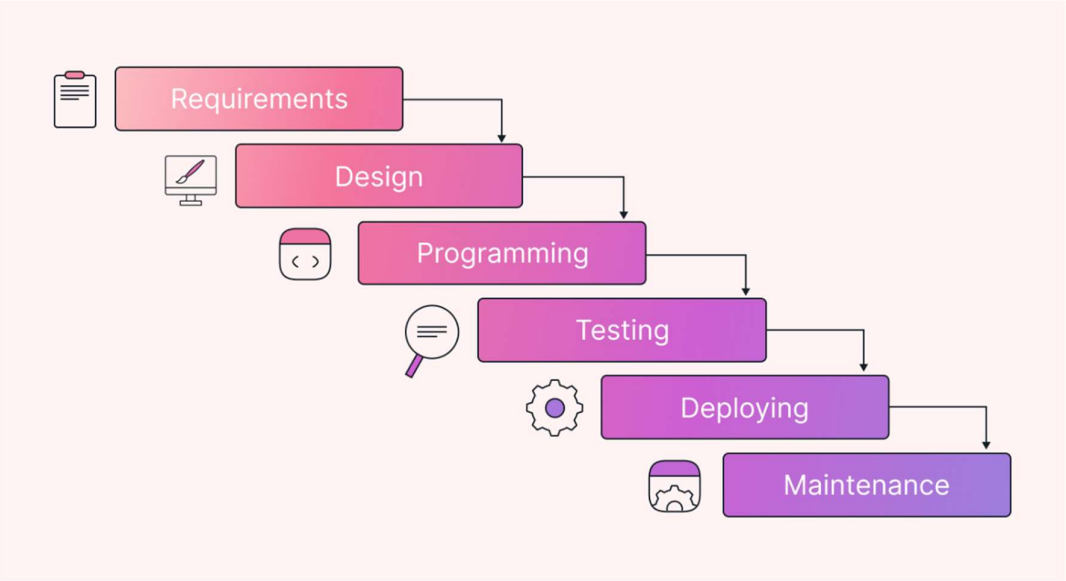


Fig 1.5.1 Waterfall Model

Key Reasons for Choosing the Waterfall Model:

- Offers a structured and well-documented process.
- Ideal for academic environments where project timelines and outputs are predefined.
- Makes it easier to manage tasks in a step-by-step manner.
- Helps ensure clarity and completeness before progressing to the next stage.

StaffSphere Development Phases:

1. **Requirement Analysis** – Collected all functional and non-functional requirements from institutional needs, identifying user roles and system goals.
2. **System Design** – Created UI wireframes, designed the system architecture, and planned the MongoDB database schema and frontend-backend integration.
3. **Implementation** – Developed individual modules including faculty registration, admin login, document uploads, Excel export, and pie chart generation using React, Node.js, and MongoDB.
4. **Testing** – Conducted module-level and system-level testing to validate functionality, fix bugs, and ensure smooth performance across user roles.
5. **Deployment** – Deployed the fully functional application for demonstration and internal use within the institution.
6. **Maintenance** – Prepared for future additions such as chatbot integration and feedback analytics.

By using the Waterfall model, the development process was executed with clarity and order, ensuring each component of the system was thoroughly planned, built, and tested before moving forward.

1.6 Organization of the Report

This report is structured into ten well-defined chapters, each focusing on a specific phase of the StaffSphere project to ensure a comprehensive understanding of the system's development, analysis, and implementation.

- **Chapter 1** provides an introduction to the project, including its overview, problem statement, objectives, scope, the adopted SDLC model, and the organization of the report itself.
- **Chapter 2** presents a literature survey, analyzing existing systems, identifying their limitations, and establishing the need for the proposed system.

- **Chapter 3** focuses on system analysis, including feasibility studies, software requirements, and both functional and non-functional requirements.
- **Chapter 4** discusses system design, showcasing architecture diagrams, ER models, UML diagrams, and user interface design standards.
- **Chapter 5** elaborates on the implementation details, technology stack, module-wise development, integration approach, and coding practices.
- **Chapter 6** details the testing process, including different levels of testing, test cases, results, and quality assurance measures.
- **Chapter 7** highlights the results and performance of the system with supporting screenshots and comparative evaluations.
- **Chapter 8** concludes the project by summarizing the work done, discussing challenges faced, and suggesting future enhancements.
- **Chapter 9** lists the references, including technical papers, websites, and forums consulted during development.
- **Chapter 10** includes appendices such as additional diagrams, screenshots, and supporting documents relevant to the project.

CHAPTER 2 LITERATURE SURVEY

2.1 REVIEW OF EXISTING SYSTEM

The current faculty and staff management systems used in many academic institutions rely on outdated and inefficient methods, including manual data entry, paper-based documentation, and decentralized storage of information. Faculty details are often stored in disparate systems or spreadsheets, leading to inconsistencies, errors, and time-consuming administrative work. Faculty members usually have to update their information on their own, without any automated reminders or structured format. The lack of integration between systems and the absence of real-time data processing complicates faculty management for administrators. Furthermore, there is a limited capacity for generating customized reports or analytics based on faculty data. Additionally, faculty members often lack easy access to detailed profiles of other faculty with similar interests or expertise, reducing potential collaboration opportunities.

2.2 LIMITATIONS OF EXISTING APPROACHES

1. **Manual Data Entry**: Data entry for faculty profiles is often manual and time-consuming, leading to frequent human errors.
2. **Inconsistent Data**: Faculty records are often stored in different locations and formats, making it difficult to maintain consistency.
3. **Lack of Notifications**: Faculty are not regularly reminded to update their profiles, resulting in outdated information.
4. **No Collaboration Platform**: Faculty members do not have an integrated system to easily search for colleagues with similar research interests.
5. **Limited Reporting**: Administrators struggle to generate customized reports based on faculty information, making it difficult to analyze data efficiently.

2.3 NEED FOR PROPOSED SYSTEM

Staff Sphere is a web-based platform designed to streamline the management of faculty profiles in academic institutions. By centralizing all faculty-related information, the system eliminates the inefficiencies and inaccuracies of manual processes. Faculty members can easily create and update their profiles with personal, professional, and educational details. The system sends automated notifications to remind faculty to update missing or outdated information on a bi-weekly basis. Additionally, a chatbot feature allows faculty to find and contact other faculty members with similar research interests. Administrators can generate customized reports based on specific criteria and export them in Excel format. The system also provides real-time analytics in the form of pie charts, showing faculty distribution based on different areas of expertise.

Benefits of the Proposed System:

1. **Automated Profile Management**: Faculty profiles are created and updated through an easy-to-use interface, with automatic reminders for incomplete sections.
2. **Collaboration Opportunities**: Faculty can search for colleagues with similar interests and collaborate on research projects or publications.
3. **Customized Reporting**: Administrators can generate and download detailed reports based on faculty data, improving decision-making.
4. **Real-Time Data Analytics**: Pie charts and other data visualizations allow administrators to view faculty distribution and other useful metrics at a glance.
5. **Scalability**: The system can accommodate a growing number of faculty members and be customized to meet the needs of any academic institution.

2.4 COMPARATIVE STUDY

S No	Author (Publisher)	Paper Title	Year	Parameters and Objectives of Paper
1	Naomi A. Bajao, Gennylou P. Nuñez, Shaina Mae M. Bontia, Kevin Vincent C. Montecillo	Web-Based Faculty Profile Management System	2023	The paper introduces a web-based system for efficient faculty profile management, with objectives of simplifying the data entry and update process, allowing administrators to monitor and manage faculty profiles effectively.
2	Ms. Nishu Sethi, Anshu Malhotra	Efficiency Engine: Designing and Implementing an Academic Management System	2023	The paper focused on automating academic workflows to reduce human error, improve time efficiency, and ensure seamless data processing across departments.
3	Neha Gupta, Ritu Agarwal	Improving Faculty Data Management: A Cloud-Based Approach	2020	Focused on implementing a cloud solution for faculty data to improve reliability, accessibility, and record transparency.
4	Maria Fe P. Villanueva, John Michael R. Trinidad, and Adrian C. Santos	Faculty Portfolio Management System	2020	Focuses on creating a web-based platform for faculty to upload and update portfolios. The system includes features for report generation and admin monitoring to track faculty achievements and contributions.
5	Jonathan C. Rivera, Arnel G. Solis, and Mark Louie A. Valencia	Implementation of a Faculty Evaluation System Using AHP Algorithm	2019	Proposes a faculty evaluation system using the Analytic Hierarchy Process (AHP) algorithm to structure decision-making, focusing on improving the transparency and objectivity of faculty assessments.

S No	Paper Title	Algorithm Used	Your Observation and Impact on Your Project
1	Web-Based Faculty Profile Management System	None	The paper emphasizes ease of updating and managing faculty profiles, which directly impacts our project by providing a user-friendly interface for faculty members to manage their data.
2	Efficiency Engine: Designing and Implementing an Academic Management System	None	It emphasized the relevance of form validation and clean UI logic that StaffSphere adopted in its registration module.
3	Improving Faculty Data Management: A Cloud-Based Approach	None	Led to incorporating strong cloud security measures and authenticated API access in StaffSphere to facilitate future integration with external academic systems.
4	Faculty Portfolio Management System	None	The focus on portfolio management is crucial for enabling faculty to present their achievements. This feature will be integrated into our platform to allow faculty to update and maintain portfolios.
5	Implementation of a Faculty Evaluation System Using AHP Algorithm	AHP (Analytic Hierarchy Process)	The AHP algorithm's application to faculty evaluation can be incorporated into our project for future features, enhancing data-driven decision-making in faculty assessments.

S No	Author (Publisher)	Paper Title	Year	Parameters and Objectives of Paper
6	Maria Teresa S. Garcia and Allan Raymund S. Evangelista	A Faculty Record Management System Using PHP and MySQL	2018	Presents a system based on PHP and MySQL for automating faculty record management, aiming to reduce human errors and improve data retrieval speed and accuracy.
7	Sayali P. Dalke, Shruti A. Deshmukh, Janabai G. Dalave, Vaishnavi N. Sasane, Pooja K. Dhule	Web-Based Staff Management System	2017	The objective was to digitize manual staff records and create a digital interface with secure storage and quick retrieval capabilities.
8	Noopur Gupta, Rakesh K. Lenka, Rabindra K. Barik, Harishchandra Dubey	FAIR: A Hadoop-based Hybrid Model for Faculty Information Retrieval System	2017	This study proposed a hybrid data system for searching faculty profiles using a big data backend to handle vast academic datasets.
9	Kashish Ara Shakil, Shuchi Sethi, Mansaf Alam	An Effective Framework for Managing University Data using a Cloud-Based Environment	2015	It proposed a cloud framework to manage university-wide data, improving data access, reducing infrastructure cost, and enhancing backup.
10	Elson E. Paciano, Gloria B. Muhi, and Allan J. Alcazaren	Design and Implementation of a Web-Based Faculty Information System	2015	Describes a web-based faculty information management system that centralizes faculty profiles, including qualifications and certifications, for streamlined administrative processes and improved report generation.

S No	Paper Title	Algorithm Used	Your Observation and Impact on Your Project
6	A Faculty Record Management System Using PHP and MySQL	PHP, MySQL	The PHP and MySQL-based approach to faculty record management aligns with our system's backend structure, offering a solid foundation for storing and retrieving faculty data.
7	Web-Based Staff Management System	JSP, MySQL	Inspired secure login and role-based access in StaffSphere, allowing faculty and admins to manage data according to permission levels and roles.
8	FAIR: A Hadoop-based Hybrid Model for Faculty Information Retrieval System	Apache Spark and Hive	Demonstrated scalability techniques useful for future-proofing StaffSphere. It motivated integration options for external research databases and scalable data processing.
9	An Effective Framework for Managing University Data using a Cloud-Based Environment	Cloud Computing	Supported decision to use MongoDB Atlas and cloud architecture for StaffSphere, ensuring real-time data sync and multi-location access.
10	Design and Implementation of a Web-Based Faculty Information System	None	Centralizing faculty information as discussed in this paper is directly relevant to our project's goal of creating a unified platform for faculty data management.

Table 2.4.1 Comparative Study

CHAPTER 3 SYSTEM ANALYSIS

3.1 FEASIBILITY STUDY

3.1.1 TECHNICAL FEASIBILITY

The Staff Sphere platform is technically feasible as it will be developed using popular web development technologies such as Node.js, React, and MongoDB. These technologies are well-suited for developing scalable, real-time web applications. The platform will also be cloud-hosted, providing the flexibility to scale as the institution grows. The system will integrate easily with existing institutional databases and will be capable of handling large volumes of data.

3.1.2 ECONOMIC FEASIBILITY

One of the key advantages of StaffSphere is its cost-effectiveness. Since all core technologies used are open-source, there are no licensing costs involved. The system is designed to reduce manual workload and improve institutional efficiency, thereby saving administrative effort and operational costs in the long run. Hardware requirements are minimal, and cloud deployment can be done using cost-efficient services. As a result, the total investment required is reasonable for an academic institution, making the project economically viable.

3.1.3 OPERATIONAL FEASIBILITY

The platform is designed to be intuitive and user-friendly, ensuring smooth adoption by both faculty members and administrative staff. Faculty can easily register and update their profiles, upload certificates, and manage their data in a structured manner. On the other hand, administrators have access to powerful features like Excel report generation, visual analytics, and data segmentation. These capabilities align well with the operational needs of academic institutions, ensuring that the system can be effectively integrated into existing workflows. Therefore, StaffSphere is considered operationally feasible.

3.1.4 TIME AND COST ESTIMATION

The project was planned and executed over a period of approximately 4 to 5 months, divided into clearly defined phases such as requirement analysis, system design, module-wise development, testing, and deployment. The cost primarily includes developer time, minor cloud/server charges, and internet access, all of which are manageable within the project's scope. Given the structured planning and efficient use of resources, the project was completed within the estimated timeline and budget.

3.2 SOFTWARE REQUIREMENTS SPECIFICATION (SRS)

This section outlines the software requirements essential for the successful development and deployment of the **StaffSphere** system. It includes the required software tools, their purpose in the system, and the specific modules where each is applied.

Software Requirements Used in the Project

The following software tools and technologies were required to design and build the StaffSphere system:

- React.js
- CSS
- Node.js
- Express.js
- MongoDB
- Chart.js
- Excel (.xlsv) Integration

These tools collectively supported frontend rendering, backend logic, data storage, visualization, and reporting.

Purpose of the Software Requirements

Each software was selected based on the functionality and performance it provided for the system:

- **React.js** was chosen for building a dynamic, component-based frontend that supports smooth user interactions and real-time updates.
- **CSS** was used to maintain consistent styling across the system, ensuring a responsive and clean UI for both faculty and admin users.
- **Node.js** provided the backend runtime to handle API requests, business logic, and data processing.
- **Express.js** enabled quick and modular route creation for user authentication, data handling, and admin operations.
- **MongoDB** served as the NoSQL database, allowing structured storage of faculty information using flexible document schemas.
- **Chart.js** was integrated to generate visual statistics like pie and bar charts for admin analysis.
- **Excel (.xlsv) integration** was added to support data export for offline access, audits, and reporting needs.

Application of Software to Project Modules

Each software component contributed to specific modules within the system:

- **React.js**
  Applied to the Faculty Registration, Faculty Home Page, Profile Editing, and Admin Dashboard modules.
- **CSS**
  Used across all frontend modules for layout styling, form design, and responsive adjustments.
- **Node.js**
  Implemented in Login, OTP Verification, Profile Edit Handling, and Data Retrieval APIs.
- **Express.js**
  Used in Route Management, Data Filtering, Admin Requests Handling, and Password Reset logic.

- **MongoDB**
  Connected to Faculty Data Storage, including the registrations and relievedFaculty collections, across all modules involving user data.
- **Chart.js**
  Specifically used in the Admin Dashboard to display visual insights like faculty distribution and experience graphs.
- **Excel (.xlsv)**
  Integrated in the Admin Reporting Module for exporting filtered faculty data into spreadsheets.

3.3 FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS

In order to ensure the successful development and deployment of the StaffSphere application, it is essential to clearly define both the functional and non-functional requirements. Functional requirements specify the core operations and features that the system must perform. On the other hand, non-functional requirements define the system's quality attributes like performance, usability, reliability, and security, which ensure that the system operates efficiently under defined conditions. Together, these requirements serve as a foundation for design, development, testing, and validation of the application.

FUNCTIONAL REQUIREMENTS

1. **User Management**:
   - Faculty members will be able to create and update their profiles, which include personal, professional, and academic information.
   - Administrators will have the ability to access and edit any profile, and manage faculty data centrally.
2. **Report Generation**:
   - Admins will have the capability to generate reports based on selected criteria, with the option to export the data in Excel format.
3. **Analytics**:
   - The system will provide visual data analytics in the form of pie charts and graphs to help admins analyze faculty data for decision-making.

- Provides a dashboard about the faculty data in the database

4. **Chatbot for Collaboration**:
    - Faculty members will have access to a chatbot for identifying colleagues with similar research interests. The chatbot will suggest faculty for collaboration based on their profiles.

NON-FUNCTIONAL REQUIREMENTS

1. **Performance**:
    - The system should respond to faculty profile updates and admin queries in less than 2 seconds.
    - The report generation should take no longer than 5 seconds for datasets with up to 1,000 records.
2. **Usability**:
    - The user interface must be intuitive, with minimal training required for both faculty and administrators.
    - The system should be mobile-friendly and responsive across all screen sizes.
3. **Availability**:
    - The system should be operational 99% of the time, with planned downtime communicated in advance.

CHAPTER 4 SYSTEM DESIGN

4.1 SYSTEM ARCHITECTURE

The system architecture of StaffSphere provides a clear representation of how various users and components interact with one another to manage faculty data efficiently. The architecture primarily revolves around two user roles — Faculty and Admin, each with distinct responsibilities and operations.

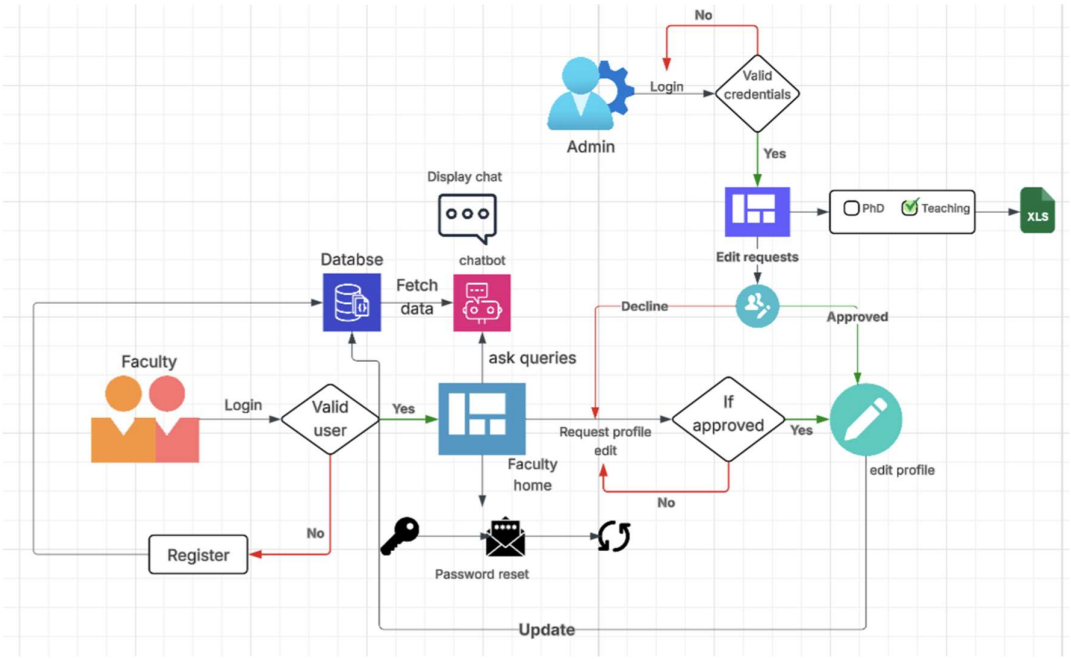


Fig 4.1.1 System Architecture

Faculty Role

The faculty members are the primary users of the system. When accessing the system for the first time, they begin with the registration process, where they enter their personal, professional, qualification, additional, and upload details. Upon successful registration, this information is stored securely in the database.

Once registered, faculty can log in using their faculty ID and password. After a successful login, they are redirected to their Faculty Home Page.

From this page, faculty can perform several important functions:

- **Chatbot Interaction**: Faculty can interact with an AI-powered chatbot to ask queries related to other faculty members, such as interests, expertise, or availability. The chatbot fetches this information from the database and displays relevant responses.
- **Profile Edit Request**: If a faculty member wishes to update their information, they must first request profile editing. This request is sent to the admin for approval. Once approved, the "Request Edit" button becomes "Edit Profile", and the faculty can proceed to update their details and commit the changes.
- **Password Reset**: In case of forgotten credentials, the faculty can reset their password by entering their faculty ID and registered email. If the details are validated, an OTP is sent to the email. After verifying the OTP, the faculty can set a new password and are redirected back to the login page.

Admin Role

The Admin manages the overall operations of the system. Admins log in using their name and password. After successful authentication, they access the Admin Dashboard, which provides control over several key functions:

- **Edit Request Management**: Admins can view the list of profile edit requests from faculty. They have the authority to either approve or decline each request. If approved, the faculty will see the option to edit their profile.
- **Filtering and Data Visualization**: Admins can apply filters using checkboxes and sub-checkboxes to segment the faculty data based on various categories such as PhD status, teaching experience, etc. The filtered data can be exported to Excel and is also represented through visual charts and graphs.
- **View Statistics**: A dedicated Stats section provides a graphical view of the complete faculty data, giving the admin insights into various aspects of the database through a dashboard interface.
- **Relieving Faculty**: Admins can relieve a faculty member from the system by entering their faculty ID, effectively removing their data from the platform.

4.2 DATABASE DESIGN

The database for the StaffSphere project is built using MongoDB, a flexible and scalable NoSQL database that supports hierarchical data structures through collections and documents. The database used in this system is named StaffDb, which houses two key collections: registrations and relievedFaculty. Each collection is designed with a detailed schema to accommodate the specific requirements of faculty data management in an academic institution.

StaffDB – Mongodb Database

The database is structured to store and manage comprehensive faculty profiles. Its schema is designed to handle a wide range of fields — from personal information to academic achievements and uploaded documents — making it suitable for dynamic querying, filtering, and reporting.

Collections in StaffDB

a. registrations Collection

This collection stores data of all **active faculty members**. Each document in this collection includes the following structured sections:

- **personalInfo**: Contains personal and contact details such as name, phone numbers, email addresses, gender, caste, faculty ID, Aadhaar, PAN, and address.
- **qualificationInfo**: Captures educational details including UG, PG, and PhD-related information. The phdList array allows storage of multiple PhD records with supporting metadata and certificate references.
- **experienceInfo**: Holds teaching, research, and industry experience, including start dates, subjects taught, and associated projects or companies.
- **additionalDetails**: Covers areas of interest, certifications, exams, projects, and professional memberships.
- **uploads**: Stores file paths or links for uploaded documents like Aadhaar card, PAN card, and passport photo.

- **password**: A secure field to store the user's password (defaulted to faculty ID on initial registration).
- **editRequestStatus**: Tracks whether a faculty member has requested a profile edit.

Fig 4.2.1 Registrations Collection

b. relievedFaculty Collection

This collection stores data of faculty who have been relieved from the institution. It maintains the same structure as the registrations collection, preserving all historical data including personal, educational, and experience-related information. However, it does not include fields like editRequestStatus since profile editing is no longer applicable after relieving.

Fig 4.2.2 Relieved Date

4.3 UML DIAGRAMS

USECASE DIAGRAM

The Use Case Diagram of the StaffSphere project illustrates the interactions between the system and its two primary users: Faculty and Admin. Faculty can perform actions such as registration, login, interacting with the chatbot, requesting profile edits, resetting passwords, and updating their profile after approval. Admin users are responsible for logging in, approving edit requests, applying filters, exporting data, viewing statistics, and relieving faculty. The diagram clearly maps each user to their respective functionalities. It provides a visual overview of the system's functional requirements and the different ways users engage with the application.

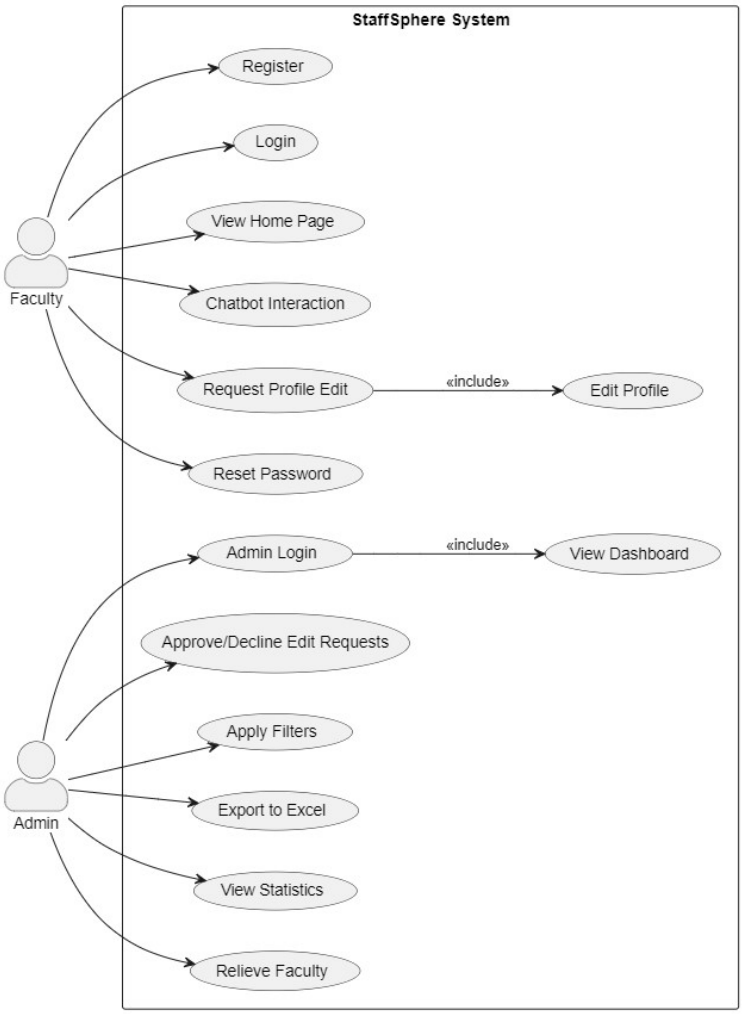


Fig 4.3.1 Usecase diagram

CLASS DIAGRAM

The Class Diagram for the StaffSphere system represents the static structure of the application by detailing the main classes, their attributes, and relationships. The key classes include **Faculty**, which holds personal, educational, professional, and additional details, and **Admin**, which manages system-level operations like approvals and data filtering. The **Database class** connects with two collections: registrations for active faculty and relievedFaculty for removed records. The diagram helps visualize how data is organized and how different parts of the system interact structurally. It supports the object-oriented design approach used in the development of the project.

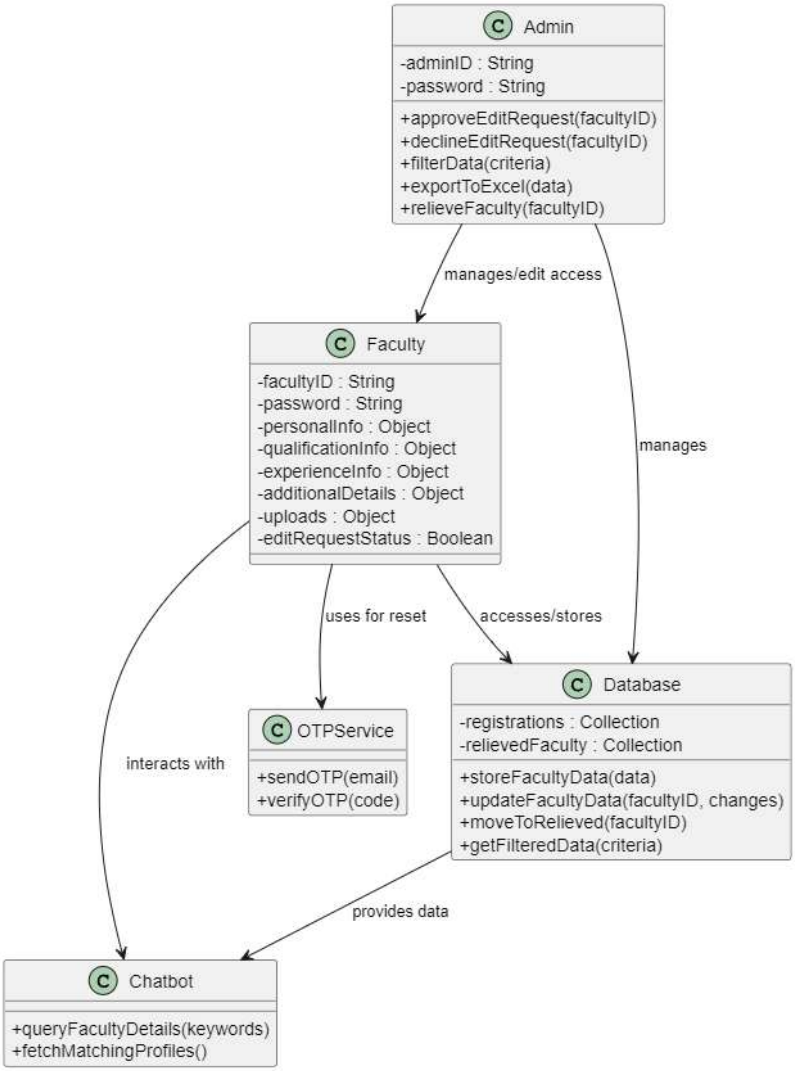


Fig 4.3.2 Class Diagram

ACTIVITY DIAGRAM

The Activity Diagram illustrates the dynamic workflow of the StaffSphere system by representing the step-by-step execution of key processes. It highlights how faculty members move through stages like registration, login, profile editing, and password reset, depending on their interaction. Similarly, it maps the admin's workflow, including login, handling edit requests, applying filters, and relieving faculty. The diagram captures conditional flows, decisions, and parallel processes, making it easier to understand the system's logic. It serves as a visual guide to how activities are coordinated and executed within the application.

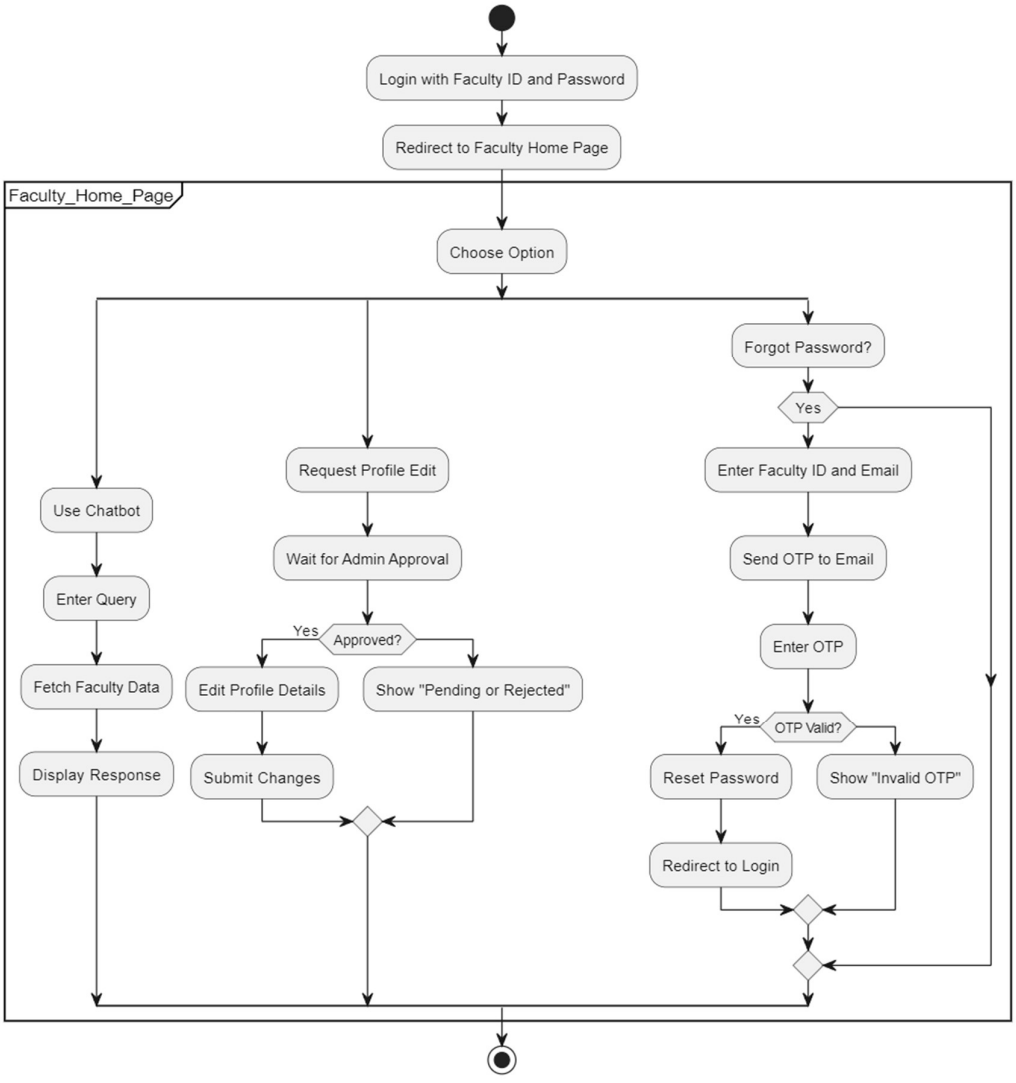


Fig 4.3.3 Activity Diagram

SEQUENCE DIAGRAM

The Sequence Diagram illustrates the order of interactions between various components of the StaffSphere system during specific operations. It shows the communication flow between Faculty, Frontend, Backend, and the Database during processes like login, profile edit, or password reset. The diagram visually represents how a faculty member sends a request, how the system validates credentials, and how responses are sent back in sequence. It also captures the admin's flow during request approval and data filtering. This diagram helps in understanding the real-time flow of control and the timing of operations within the system.

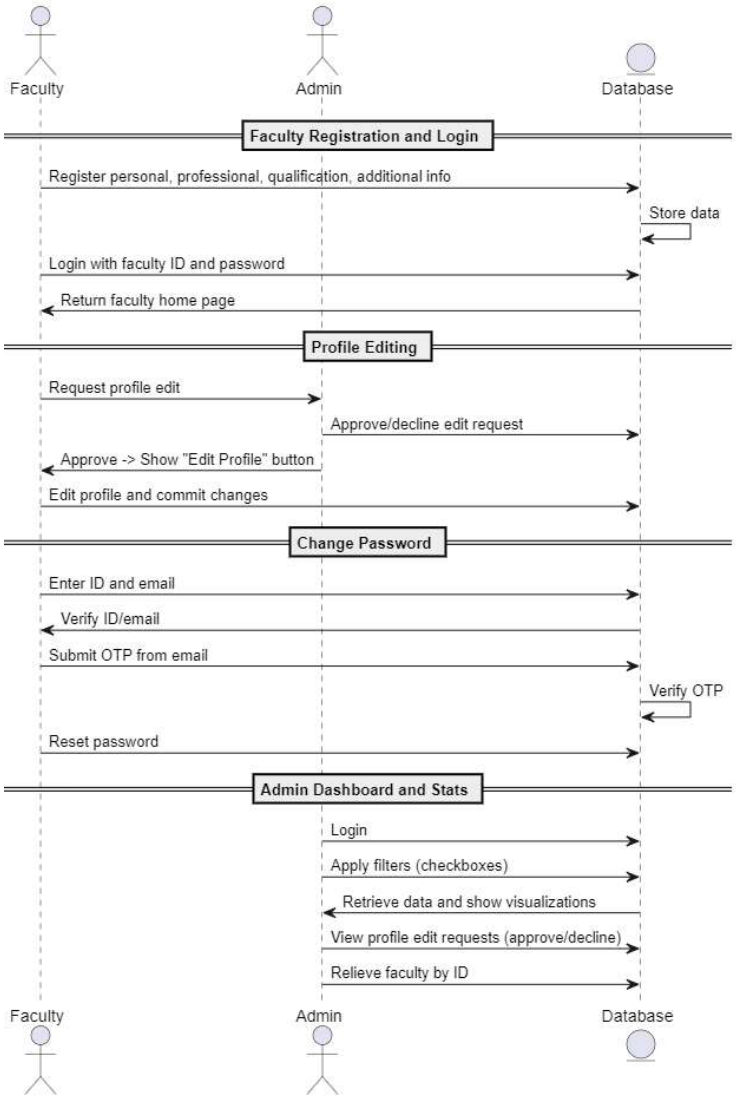


Fig 4.3.4 Sequence Diagram

OBJECT DIAGRAM

The Object Diagram represents a snapshot of the system at a specific moment in time, showing the actual instances (objects) of classes and their relationships. In the StaffSphere project, it illustrates real-world examples such as a Faculty object with values filled for personal, qualification, and experience details, and an Admin object handling approval status or filter settings. The diagram reflects how these objects interact with the Database and other components during system execution. It helps in understanding the system's state and data structure during runtime, based on the defined class blueprint.

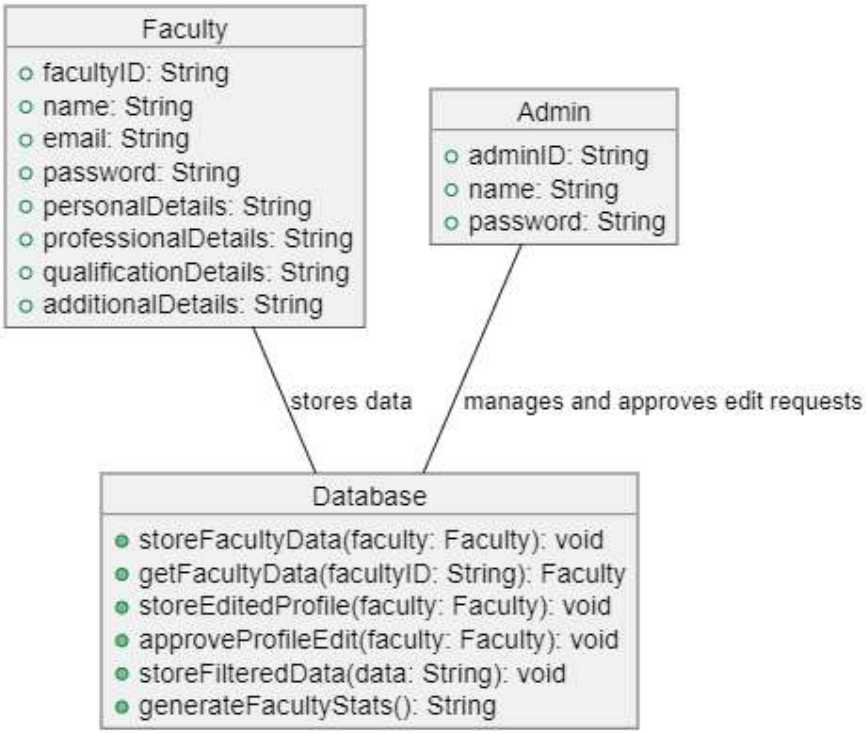


Fig 4.3.5 Object Diagram

DATAFLOW DIAGRAM

The Data Flow Diagram (DFD) of the "StaffSphere" project visually represents the flow of data within the system. It highlights how data moves between users (faculty and admin) and the database. The diagram shows the faculty's registration process, where their personal, professional, qualification, and other details are stored. It also depicts how faculty can edit their profile after admin approval and reset their password. The admin has the ability to view and filter data, approve or decline profile edit requests, and relieve faculty members. This diagram ensures that the data handling process is clearly defined, from data entry to storage and retrieval, making it easy to understand the system's operations.

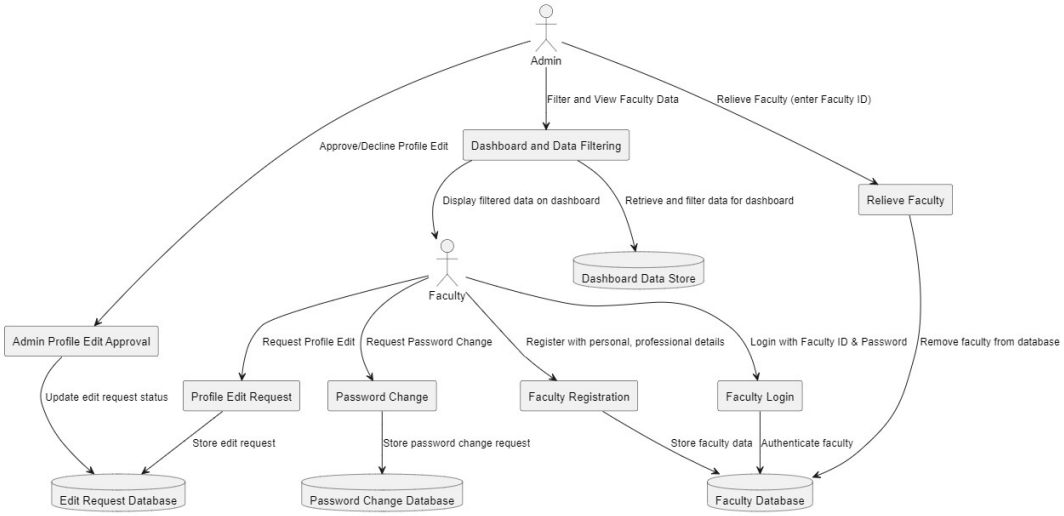


Fig 4.3.6 Dataflow Diagram

STATE DIAGRAM

The State Diagram illustrates the different states that an object, particularly the Faculty user, can occupy throughout its lifecycle within the StaffSphere system. It begins with the initial registration state, transitions to logged-in state, and may move to states like awaiting edit approval, editing profile, or password reset. For admins, states include logged in, reviewing requests, and relieving faculty. Each state change is triggered by specific actions or events, such as approvals, submissions, or errors. This diagram helps in understanding how the system reacts to user actions and transitions based on internal conditions.

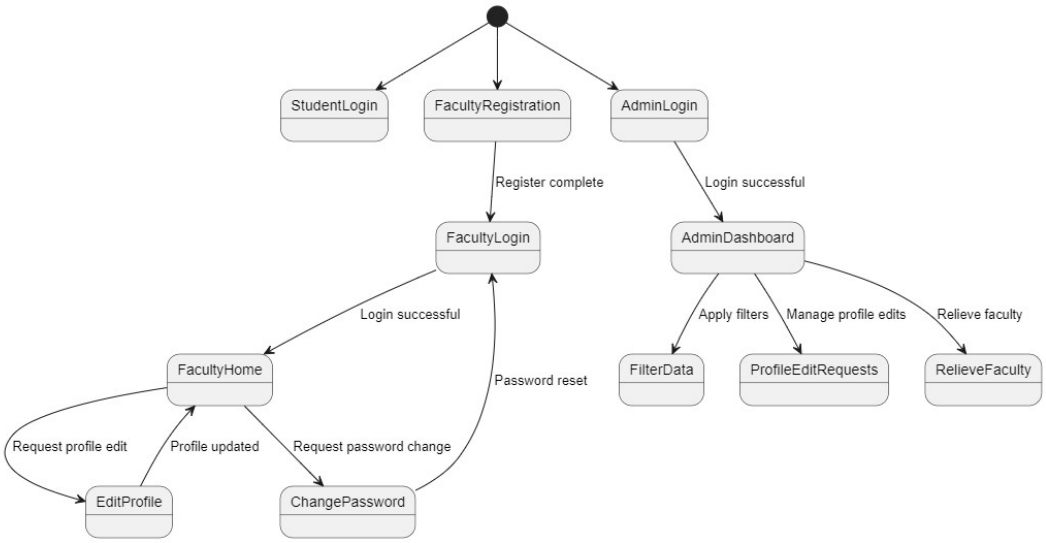


Fig 4.3.7 State Diagram

4.4 USER INTERFACE DESIGN

The user interface (UI) of the StaffSphere system is designed to offer a simple, intuitive, and responsive experience for both faculty and admin users. The primary objective of the UI design is to ensure that users can perform their tasks with ease, without needing extensive technical knowledge. The interface is structured to support clean navigation, minimal input errors, and smooth transitions between various modules.

The system is developed using React.js for the frontend, with styling handled by Tailwind CSS, ensuring a modern and responsive layout. The UI adapts to various screen sizes, making the application accessible on desktops, laptops, and tablets. All inputs, buttons, and forms follow a consistent design language, maintaining uniformity across different pages.

Design Approach and Layout

The application layout is role-based, meaning that faculty and admin users see different sets of pages and controls based on their login credentials. Faculty users primarily interact with forms for registration, chatbot queries, profile editing, and password management. Admin users, on the other hand, access a more comprehensive dashboard with controls for filtering data, managing edit requests, and exporting reports.

Each form or page is broken down into logical sections:

- The **Faculty Registration** page is split into personal, qualification, professional, additional, and uploads sections to reduce visual clutter and improve focus.
- The **Admin Dashboard** is organized into panels that group related actions together — for example, checkboxes for filtering, request review table, export options, and statistics charts.
- The **Faculty Home Page** contains clearly labeled buttons to interact with the chatbot, request profile edits, and access the password reset module.

User Interaction Flow

Upon successful login, users are directed to their respective dashboards. Faculty users can navigate through key features such as:

- **Chatbot Interaction**: A conversational interface for retrieving faculty data based on specific interests.
- **Profile Edit**: A button that changes from “Request Edit” to “Edit Profile” after admin approval.
- **Password Reset**: A step-by-step flow that verifies the user’s email and sends an OTP for password recovery.

Admins interact with checkboxes and sub-checkboxes to filter faculty based on department, qualification, experience, and more. These selections dynamically generate data views that can be exported to Excel and visualized as charts.

Design Considerations

The UI follows the principles of:

- **Clarity**: Use of proper labels, icons, and color codes for different sections.
- **Consistency**: Uniform styles across all input fields, buttons, and navigation elements.
- **Feedback**: Real-time validation messages and success/error alerts for all form actions.
- **Accessibility**: Contrast-friendly colors and clear typography for readability.

Navigation is kept simple, with top-level menus or dashboard cards guiding users to important features. The use of collapsible sections and step-wise progress ensures that users are not overwhelmed by too much information at once.

Responsiveness and Accessibility

The interface is responsive and performs well on various devices. Layout elements adjust based on screen size, ensuring all users have a consistent experience regardless of their device. Components like modals, buttons, and input fields are touch-friendly and keyboard-accessible, improving usability for all types of users.

4.5 DESIGN STANDARDS FOLLOWED (IEEE, ISO, ETC.)

The design and development of the **StaffSphere** system followed several industry-recognized standards and best practices to ensure quality, consistency, usability, and maintainability. These standards were applied across various stages, including system design, user interface layout, database structuring, and documentation.

1. Software Design Standards

The system design adheres to the IEEE 1016 standard for Software Design Descriptions. This ensured that the architecture and module-level designs were well-structured, modular, and aligned with the project's functional and non-functional requirements. UML diagrams such as Use Case, Class, Sequence, and Activity were created using standard UML 2.x notation for clear and standardized modeling.

2. User Interface (UI/UX) Standards

The UI was designed following principles from ISO 9241-210: Human-Centered Design for Interactive Systems, which focuses on usability, clarity, and accessibility. Consistent use of components, color schemes, and intuitive navigation ensured a seamless experience for both faculty and admin users. Responsive web design practices were adopted to support access from various screen sizes and devices.

3. Coding Standards

Frontend development using React.js followed structured component-based design and followed Google JavaScript Style Guide conventions. Proper code indentation, naming conventions, and reusable components were maintained throughout. For the backend (Node.js), RESTful API design conventions and separation of concerns were implemented to keep the logic modular and scalable.

4. Database Design Guidelines

The database schema in MongoDB was structured following MongoDB Schema Design Best Practices, such as using embedded documents for related data (e.g., personalInfo, qualificationInfo) and separating collections (registrations and relievedFaculty) based on faculty status. This ensured optimized data retrieval and easy maintainability.

4.6 SAFETY & RISK MITIGATION MEASURES

While developing and deploying the **StaffSphere** system, several potential risks were identified and addressed to ensure the reliability, security, and smooth functioning of the application. Appropriate safety and mitigation strategies were incorporated during the design and development phases to reduce the impact of these risks.

1. Data Security and Unauthorized Access

To prevent unauthorized access to sensitive faculty data, secure authentication mechanisms were implemented. Faculty and admin users must log in with valid credentials. Passwords are stored securely, and OTP-based email verification is used for password reset operations.

Mitigation:

- Secure login system with encrypted credentials
- OTP verification for password recovery
- Session-based access control to restrict unauthorized actions

2. Data Loss or Corruption

Unintended data loss or corruption can occur due to crashes or database issues.

Mitigation:

- Separation of active and relieved faculty data into different collections
- Well-structured MongoDB schemas to avoid redundancy and inconsistency

3. Input Validation and Form Errors

Invalid data entries during registration or profile editing could affect data accuracy.

Mitigation:

- Form validation implemented at both frontend and backend levels
- Mandatory fields and format checks
- Real-time error messages for missing or incorrect inputs

CHAPTER 5 IMPLEMENTATION

5.1 TECHNOLOGY STACK

The **StaffSphere** project is built using a modern web development stack that combines efficient frontend rendering, secure backend handling, interactive data visualization, and flexible database storage. The technologies were chosen based on their reliability, scalability, and ease of integration for a modular full-stack application.

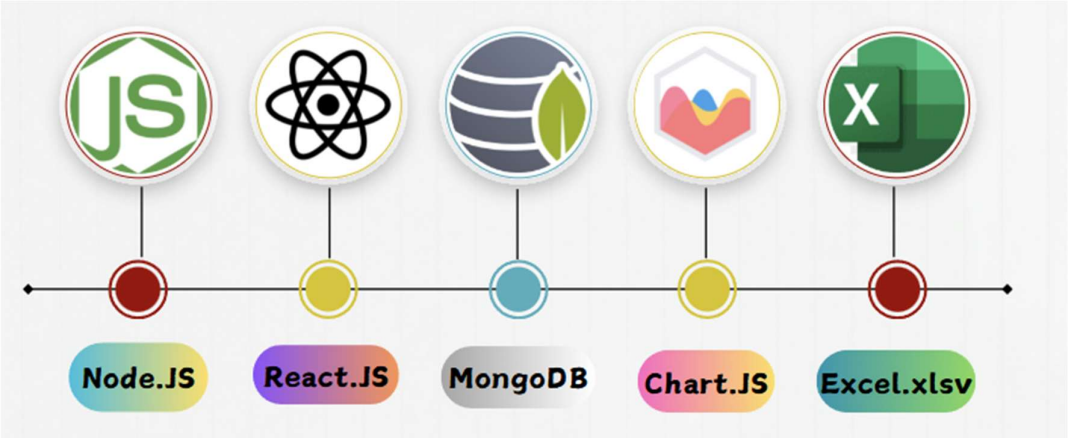


Fig 5.1.1 Technology Stack

Frontend

- **React.js**: A component-based JavaScript library used to build dynamic and interactive user interfaces. It helps manage the state of complex forms and views, especially for faculty and admin dashboards.
- **CSS**: Used for designing and styling the UI elements. It ensures a clean, consistent, and responsive layout across all modules of the application.
- **Chart.js**: A JavaScript charting library integrated into the frontend to generate interactive pie charts, bar charts, and other visual statistics for admin users.

Backend

- **Node.js**: A JavaScript runtime environment used for executing server-side logic and handling API requests efficiently.

- **Express.js**: A lightweight and flexible Node.js web framework used to define routes, handle requests/responses, and connect the frontend to the database.

Database

- **MongoDB**: A NoSQL database used to store faculty information in structured collections. Two collections are maintained: registrations for active faculty and relievedFaculty for those who have been relieved. Its document-based schema supports complex, nested data formats.

Data Export

- **Excel (.xlsv) Integration**: The admin panel includes functionality to export filtered faculty data into Excel spreadsheets in .xlsv format. This allows for easy offline access, reporting, and institutional record-keeping.

This technology stack ensures seamless integration between components, scalability for future enhancements, and a user-friendly experience across both faculty and admin roles.

5.2 MODULE-WISE IMPLEMENTATION

The StaffSphere system is designed in a modular fashion to ensure that each functional component operates independently while contributing to the system as a whole. Each module is implemented using relevant frontend and backend technologies to handle specific tasks efficiently. The following are the major modules developed:

Faculty Registration Module

This module allows new faculty members to register by entering their personal, professional, qualification, additional details, and uploading documents. The data is validated on the client side using React and stored in the MongoDB database via backend APIs built using Node.js and Express.js.

Faculty Home Page Module

This module serves as the central dashboard for faculty. It includes options like chatbot access, profile edit request, and password reset. The interface is designed in React, with conditional rendering based on user actions and approval status.

Chatbot Module

This module allows faculty to interact with a chatbot to fetch information about other faculty members based on interests or keywords. The chatbot queries the MongoDB database and returns matching profiles using simple text-based responses.

Profile Editing Module

Faculty members can request to edit their profile. Once the request is approved by the admin, they are allowed to update any section of their previously submitted data. The updated details are saved back to the database after committing the changes.

Password Reset Module

This module enables users to reset their password using OTP-based email verification. The user provides their faculty ID and registered email. Upon verification, an OTP is sent, which when entered correctly, allows the user to reset their password securely.

Admin Login and Dashboard Module

The admin login module provides secure access to admin-only features. The dashboard displays all faculty records and includes options for handling edit requests, applying filters, viewing statistics, exporting to Excel, and relieving faculty.

Data Filtering and Excel Export Module

Admins can apply filters through checkboxes and sub-checkboxes to extract specific data from the database. The filtered data can be downloaded as an .xlsv Excel file for documentation and analysis purposes.

Faculty Relieving Module

This module allows the admin to relieve a faculty member by entering their faculty ID. The corresponding record is removed from the registrations collection and moved to the relievedFaculty collection for historical tracking.

5.3 CODE INTEGRATION STRATEGY

The code integration strategy for the **StaffSphere** project focused on ensuring seamless communication between the frontend, backend, and database layers. The system was developed using a modular approach, where individual features and components were implemented and tested separately before being integrated into the main application.

The **frontend**, developed using **React.js**, was organized into reusable components representing various forms, dashboards, and action buttons. Each component communicated with the backend through **RESTful API endpoints**. These endpoints were developed using **Express.js** and served as the interface between the frontend and the **MongoDB** database.

To maintain clear separation of concerns, all backend logic was structured into separate route files and controller functions. This allowed easy debugging and testing of each module independently before integrating it into the application flow. Environment variables were used to manage configurations such as database URLs and email service credentials securely.

Integration testing was carried out by connecting each module incrementally. Initially, the registration and login modules were connected and tested for data flow. Once verified, other modules like profile editing, chatbot querying, password reset, and admin dashboard functionalities were progressively added and tested in the full-stack environment.

Special care was taken to handle **asynchronous operations**, such as OTP generation and email sending, using appropriate callbacks and async/await structures. Frontend validation was also synchronized with backend validation to prevent data conflicts or errors during submission.

This strategy ensured that the codebase remained modular, maintainable, and scalable while supporting smooth integration between all system components.

5.4 SAMPLE CODE SNIPPETS

Server.js

```
import express from "express";
import cors from "cors";
import mongoose from "mongoose";
import cookieParser from "cookie-parser";
import dotenv from "dotenv";
dotenv.config();
const app = express();
app.use(cors({
   origin: "http://localhost:3000",
   credentials: true, //
}));
.then(() => console.log("MongoDB Connected Successfully"))
.catch((error) => console.error("MongoDB Connection Error:", error));
app.use("/api/registration", registrationRoutes);
app.get("/", (req, res) => {
   res.send("Server is running!");
});
const PORT = process.env.PORT || 5000;
app.listen(PORT, () => console.log(`Server running on http://localhost:${PORT}`));
```

registrationRoutes.js

```
import express from "express";
import Registration from "../models/registrationModel.js";
const router = express.Router();
router.post("/submit", async (req, res) => {
  try {
    console.log("Received Request Body:", JSON.stringify(req.body, null, 2));
    if (!req.body.personalInfo || !req.body.qualificationInfo || !req.body.experienceInfo) {
      return res.status(400).json({ success: false, error: "Missing required fields" });
    }
    if (!req.body.personalInfo.facultyID) {
      return res.status(400).json({ success: false, error: "Faculty ID is required." });
    }
    req.body.password = req.body.personalInfo.facultyID;
    const newRegistration = new Registration(req.body);
    await newRegistration.save();
    console.log("Data Saved Successfully");
    res.status(201).json({ success: true, message: "Registration successful" });
  } catch (error) {
    console.error("Error Saving Data:", error);
    res.status(400).json({ success: false, error: error.message });
```

```
  }
});
router.get("/all", async (req, res) => {
  try {
    const registrations = await Registration.find().lean();
    res.status(200).json(registrations);
  } catch (error) {
    console.error("Error Fetching Data:", error);
    res.status(500).json({ success: false, error: "Server error while fetching data" });
  }
});
export default router;
```

facultyHomeRoutes.js

```
import express from "express";
import Registration from "../models/registrationModel.js";
import { verifyFaculty } from "./authRoutes.js";
import { spawn } from "child_process";
const router = express.Router();
const getMatchingFaculty = async (query) => {
    return new Promise((resolve, reject) => {
        const pythonProcess = spawn("python", ["./scripts/ai_similarity.py", JSON.stringify(query)]);
        let data = "";
        pythonProcess.stdout.on("data", (chunk) => { data += chunk.toString(); });
        pythonProcess.stderr.on("data", (err) => { console.error("Python Error:", err.toString()); });
        pythonProcess.on("close", () => {
            try {
                const expandedData = JSON.parse(data);
                resolve(expandedData);
            } catch (error) {
                reject(error);
}});});};
router.get("/profile", verifyFaculty, async (req, res) => {
    try {
      const faculty = await Registration.findOne(
        { "personalInfo.facultyID": req.faculty.facultyID },
        { password: 0 }
      );
      if (!faculty) {
        return res.status(404).json({ success: false, message: "Faculty profile not found" });
      }
      res.json(faculty);
    } catch (error) {
      console.error("Error fetching profile:", error);
      res.status(500).json({ success: false, message: "Error fetching profile" });
    }});
```

```
router.post("/end-edit", verifyFaculty, async (req, res) => {
  try {
    const { facultyID } = req.faculty;
    const faculty = await Registration.findOneAndUpdate(
      { "personalInfo.facultyID": facultyID },
      { $set: { editRequestStatus: "None" } },
      { new: true }
    );
    if (!faculty) {
      return res.status(404).json({ success: false, message: "Faculty not found" });
    }
    res.json({ success: true, message: "Editing completed successfully!", updatedProfile: faculty });
  } catch (error) {
    console.error("Error ending edit session:", error);
    res.status(500).json({ success: false, message: "Server error. Please try again later." });
  }
});
router.post("/chatbot", verifyFaculty, async (req, res) => {
  try {
    const { query } = req.body;
    const facultyID = req.faculty.facultyID;
    const expandedData = await getMatchingFaculty(query);
    const matchingFaculty = expandedData?.matching_faculty || [];
    if (matchingFaculty.length === 0) {
      return res.json({ message: `No faculty members found with interests related to "${query}".`
});
    }
    const filteredFaculty = matchingFaculty.filter(f => f.facultyID !== facultyID);
    if (filteredFaculty.length === 0) {
      return res.json({ message: `No other faculty members found with interests related to
"${query}".` });
    }
    const responseMessage = filteredFaculty.map(f =>
      `${f.name} - ${f.phone}`
    ).join("\n");
    res.json({ message: `Faculty interested in "${query}":\n${responseMessage}` });
  } catch (error) {
    console.error("Chatbot Error:", error);
    res.status(500).json({ message: "Something went wrong, please try again later." });
  }
});
export default router;
```

ai_similarity.py

```
import sys
import json
```

```
from pymongo import MongoClient
import ollama
try:
    client = MongoClient("mongodb://localhost:27017/")
    db = client["StaffDB"]
    collection = db["registrations"]
except Exception as e:
    print(json.dumps({"error": f"MongoDB connection failed: {str(e)}"}))
    sys.exit(1)
try:
    user_query = sys.argv[1]
except Exception as e:
    print(json.dumps({"error": f"Invalid input JSON: {str(e)}"}))
    sys.exit(1)
print(f"📩 User Query Received: {user_query}", file=sys.stderr)
faculty_list = []
try:
    all_faculty_cursor = collection.find({}, {"_id": 0, "personalInfo.facultyID": 1,
"personalInfo.name": 1, "personalInfo.personalPhone": 1, "additionalDetails.areasOfInterest": 1})
except Exception as e:
    print(json.dumps({"error": f"MongoDB query failed: {str(e)}"}))
    sys.exit(1)
if not faculty_list:
    print(json.dumps({"error": "No faculty data found in database."}))
    sys.exit(1)
matching_faculty = []
try:
    for faculty in faculty_list:
        for interest in faculty["areasOfInterest"]:
file=sys.stderr)
            if similarity_result == "yes":
                matching_faculty.append({
                    "facultyID": faculty["facultyID"],
                    "name": faculty["name"],
                    "phone": faculty["phone"],
                    "matchedInterest": interest
                })
                break
except Exception as e:
    print(json.dumps({"error": f"Ollama processing failed: {str(e)}"}))
    sys.exit(1)
print(json.dumps({"matching_faculty": matching_faculty}))
```

Index.js

```
import React, { useState } from "react";
import { FaHome, FaUserShield, FaSignInAlt, FaUserPlus, FaInfoCircle } from "react-icons/fa";
import styles from "../styles/Index.module.css";
import Home from "./Home";
import AdminLogin from "./AdminLogin";
import Login from "./Login";
const Index = () => {
  const [activePage, setActivePage] = useState("home");
  return (
    <div className={styles.indexContainer}>
      <header className={styles.header}>
        <marquee className={styles.marquee} scrollamount="10">
          <span>Geethanjali College Of Engineering and Technology</span>
        </marquee></header>
      <div className={styles.mainLayout}>
        <nav className={styles.sidebar}>
          <ul className={styles.navList}>
            <li onClick={() => setActivePage("home")}>
              <FaHome className={styles.navIcon} /> Home
            </li><li onClick={() => setActivePage("admin")}>
              <FaUserShield className={styles.navIcon} /> Admin</li>
            <li onClick={() => setActivePage("login")}>
              <FaSignInAlt className={styles.navIcon} /> Faculty</li><li>
              <a href="/register"><FaUserPlus className={styles.navIcon} /> Register</a></li><li>
              <a href="/about"><FaInfoCircle className={styles.navIcon} /> About</a>
            </li></ul></nav>
        <div className={styles.contentArea}>
          {activePage === "home" && <Home />}
          {activePage === "admin" && <AdminLogin />}
          {activePage === "login" && <Login />}</div></div></div> );};export default Index;
```

RegistrationForm.js

```
import React, { useState } from "react";
import PersonalInfo from "./PersonalInfo";
import { motion, AnimatePresence } from "framer-motion";
import axios from "axios";
const RegistrationForm = () => {
  const [step, setStep] = useState(1);
  const [completedStages, setCompletedStages] = useState([false, false, false, false, false]);
  const [formData, setFormData] = useState({
    personalInfo: {
      name: "", personalPhone: "", alternativePhone: "", personalEmail: "",
      collegeEmail: "", gender: "", maritalStatus: "", caste: "", facultyID: "",
      aadhaar: "", pan: "", address: ""
```

```
  },
  qualificationInfo: {
    educationLevel: "", ugUniversityName: "",  ugLocation: "",  ugRollNumber:
"",  ugSpecialization: "",  ugCompletionDate: "",
    pgUniversityName: "",  pgLocation: "",  pgRollNumber: "",  pgSpecialization:
"",  pgCompletionDate: "",
    phdStatus: "",
    phdList: []
  },
  experienceInfo: {
    teachingExp: "", teachingStartDate: "", teachingDuration: "",
    teachingSubjects: [{ subject: "", timesTaught: "" }],
    researchExp: "", researchList: [{ name: "", link: "" }],
    industryExp: "", industryList: [{ company: "", role: "", duration: "" }],
    dateOfJoining: "",dateOfReveal: ""
  },
  additionalDetails: {
    areasOfInterest: [], examList: [{ name: "", examCertificate: "" }],
    certifications: [{ name: "", provider: "", certificationLink: "" }],
    projects: [{ name: "", domain: "", status: "" }],
    memberships: [{ name: "" }]
  },
  uploads: { aadharCard: "", panCard: "", passportPhoto: "" }
});
const cleanData = (data) => {
  if (typeof data === "string") {
    return data.trim() === "" ? "NA" : data.trim();
  } else if (Array.isArray(data)) {
    return data.map(cleanData);
  } else if (typeof data === "object" && data !== null) {
    return Object.fromEntries(
      Object.entries(data).map(([key, value]) => [key, cleanData(value)])
    );
  }
  return data;
};
const validateStep = () => {
  const { personalInfo, qualificationInfo, experienceInfo, additionalDetails, uploads } = formData;
  if (step === 1) {
    const requiredFields = [
      personalInfo.name, personalInfo.personalPhone, personalInfo.personalEmail,
      personalInfo.gender, personalInfo.maritalStatus, personalInfo.caste,
      personalInfo.facultyID, personalInfo.aadhaar, personalInfo.pan, personalInfo.address
    ];
    return requiredFields.every((field) => field.trim() !== "");
  }
  if (step === 2) {
```

```
    if (!qualificationInfo) return false;
    if (!qualificationInfo.educationLevel || qualificationInfo.educationLevel.trim() === "") return
false;
    if (!requiredUGFields.every(field => field && field.trim() !== "")) return false;
    if (qualificationInfo.educationLevel === "PG") {
      const requiredPGFields = [
        qualificationInfo.pgUniversityName, qualificationInfo.pgLocation,
        qualificationInfo.pgRollNumber, qualificationInfo.pgSpecialization,
        qualificationInfo.pgCompletionDate
      ];
        return true;});}}
  if (step === 3) {
    const requiredFields = [experienceInfo.dateOfJoining];
    if (!requiredFields.every((field) => field.trim() !== "")) return false;
    if (experienceInfo.teachingExp === "Yes" && (!experienceInfo.teachingStartDate.trim() ||
!experienceInfo.teachingDuration.trim())) {
      return false;
    }
    if (experienceInfo.researchExp === "Yes" && experienceInfo.researchList.some((r) =>
!r.name.trim() || !r.link.trim())) {
      return false;
    }
    if (experienceInfo.industryExp === "Yes" && experienceInfo.industryList.some((i) =>
!i.company.trim() || !i.role.trim() || !i.duration.trim())) {
      return false;}}
  if (step === 4) {
    return additionalDetails.areasOfInterest.length > 0;}
  if (step === 5) {
    const requiredFields = [uploads.aadharCard, uploads.panCard, uploads.passportPhoto];
    return requiredFields.every((file) => file.trim() !== "");}return true;};
 const nextStep = () => {
  if (!validateStep()) {
    alert("❌ Please fill all required fields before proceeding.");
    return;}
  setCompletedStages((prevStages) => {
 const prevStep = () => {
  setStep((prev) => prev - 1);
 const submitForm = async () => {
  try {
    const updatedFormData = cleanData(formData);
    const response = await axios.post(
      "http://localhost:5000/api/registration/submit",
      updatedFormData,
      { headers: { "Content-Type": "application/json" }});
    if (response.data.success) {
      console.log("Form Submitted Successfully:", response.data);
      window.location.href = "/thank-you";
```

```
    } else {
      alert("❌ Registration failed, please try again.");}} catch (error) {
    console.error("❌ Submission Error:", error.response?.data || error.message);
    alert("Error submitting form.");}};
 return (<><div className="top-buttons">
    <button className="home-btn" onClick={() => (window.location.href = "/")}>Home</button>
    <button className="login-btn" onClick={() => (window.location.href = "/login")}>Login
</button></div>
   <div className="registration-form-container">
    <div className="progress-stages">
      {["Personal", "Qualification", "Experience", "Additional Details", "Uploads"].map((label,
index) => (
        <div key={index} className={`stage ${step > index ? 'active' : ''} ${completedStages[index]
? `completed-${index + 1}` : ''}`}>
          <span>{index + 1}</span>
          <p>{label}</p></div>))}</div>
    <div className="progress-container"><div
      className="progress-bar"
      style={{ width: `${(completedStages.filter(Boolean).length / 5) * 100}%` }}></div></div>
    <AnimatePresence mode="wait">
     <motion.div key={step} initial={{ x: "100%", opacity: 0 }} animate={{ x: 0, opacity: 1 }}
exit={{ x: "-100%", opacity: 0 }} transition={{ duration: 0.5 }}>
       {step === 1 && <PersonalInfo formData={formData} setFormData={setFormData} />}
       {step === 2 && <QualificationInfo formData={formData} setFormData={setFormData} />}
       {step === 3 && <ExperienceForm formData={formData} setFormData={setFormData} />}
       {step === 4 && <AdditionalDetailsForm formData={formData}
setFormData={setFormData} />}
       {step === 5 && <UploadsForm formData={formData} setFormData={setFormData} />}
      </motion.div>
    </AnimatePresence>
    <nav className="navi">
      {step > 1 && <Button className="back-btn" variant="outline"
onClick={prevStep}>Back</Button>}
      {step < 5 ? <Button className="next-btn" onClick={nextStep}>Next</Button> : <Button
className="submit-btn" onClick={submitForm}>Submit</Button>}
    </nav></div></>);};
export default RegistrationForm;
```

Admin.js

```
import React, {useEffect, useState } from "react";
import {useNavigate} from "react-router-dom";
import axios from "axios";
import styles from "../styles/Admin.module.css";
import Modal from "../pages/Modal.js";
import { Pie, Bar } from "react-chartjs-2";
import { Chart as ChartJS, ArcElement, Tooltip, Legend } from "chart.js";
```

```
ChartJS.register(ArcElement, Tooltip, Legend);
const AdminPage = () => {
  const [selectedFilters, setSelectedFilters] = useState([]);
  const [error] = useState("");
  const chartOptions = {
    responsive: true,
    maintainAspectRatio: false,
    plugins: {
      legend: {
        position: "top",
},},};
  const navigate = useNavigate();
  useEffect(() => {
    axios
      .get("http://localhost:5000/api/admin/auth/dashboard", { withCredentials: true })
      .then(() => {
        setLoading(false);})
      .catch(() => {
        navigate("/admin-login");
      });}, [navigate]);
    const handleLogout = async () => {
      try {
        await axios.post("http://localhost:5000/api/admin/auth/logout", {}, { withCredentials: true });
        navigate("/");
      } catch (error) {
        console.error("Logout failed:", error);}};
  if (loading) return <p>Loading...</p>;
  if (error) return <p>{error}</p>;
  const filterOptions = [{label: "Done PhD",
      value: "phd",
      subFilters: [
        "All Details", "PhD Specialization", "Status", "Completion Certificate", "Guide Details",
        "Phone Number", "Email"]},{
      label: "Teaching Experience",
      value: "teaching",
      label: "Area of Interest",
      value: "areaOfInterest",
    },{label: "Education Details",
      value: "educationDetails",
      subFilters: [
        "Education Level","UG","PG"
      ]},{label: "Date of Joining",
      value: "dateOfJoining",
    },{label: "Projects Worked",
      value: "projects",
      subFilters: ["Project Details","Phone Number", "Email"]},{
      label: "Competitive Exams Given",
```

```
    value: "competitiveExams",
    subFilters: ["Exam Details", "Phone Number", "Email"]},{
    label: "Certifications Have",
    value: "certifications",
    subFilters: ["Certification Details", "Phone Number", "Email"]},{
    label: "Memberships",
    value: "memberships",
    subFilters: ["Membership Name","Phone Number", "Email"]}];
 const handleRevealSubmit = async () => {
  if (!facultyID.trim()) {
    setRevealMessage("Please enter a Faculty ID.");
    return;
  }try{
    const response = await axios.post("http://localhost:5000/api/admin/reveal-faculty", { facultyID
});
    setRevealMessage(response.data.message);
    setFacultyID("");
  } catch (error) {
    console.error("Error revealing faculty:", error);
    setRevealMessage("Failed to reveal faculty. Please try again.");}};
 const handleEditRequestsClick = async () => {
  setShowEditRequests((prev) => !prev);
  if (!showEditRequests) {
    try {
      const response = await axios.get("http://localhost:5000/api/admin/edit-requests");
      setEditRequests(response.data.requests);
    } catch (error) {
      console.error("Error fetching edit requests:", error);}}};
 const handleApprove = async (facultyID) => {
  try {
    const response = await axios.post("http://localhost:5000/api/admin/approve-edit", { facultyID });
    if (response.data.success) {
      alert(`Edit request for ${facultyID} approved.`);
      setEditRequests((prevRequests) => prevRequests.filter((req) => req.facultyID !== facultyID));
    }} catch (error) {
    console.error("Error approving edit request:", error);
    alert("Failed to approve edit request.");}};
 const handleDecline = async (facultyID) => {
  try {
    const response = await axios.post("http://localhost:5000/api/admin/decline-edit", { facultyID });
    if (response.data.success) {
      alert(`Edit request for ${facultyID} declined.`);
      setEditRequests((prevRequests) => prevRequests.filter((req) => req.facultyID !== facultyID));
    }} catch (error) {
    console.error("Error declining edit request:", error);
    alert("Failed to decline edit request.");}};
 const handleFilterChange = (value) => {
```

```
  setSelectedFilters((prevFilters) =>
    return { ...prev, [mainFilter]: updatedSubFilters };});};
const handleCloseSubFilterModal = () => {
  setShowSubFilterModal(null);};
const handleClearSelection = () => {
  setSelectedFilters([]);
  setSelectedSubFilters({});};
const handleClearDates = () => {
  setStartDate("");
  setEndDate("");}
const handleExport = async () => {
  if (!startDate || !endDate) {
    alert("Please select both Start Date and End Date.");
    return;
  }if (selectedFilters.length === 0) {
    alert("Please select at least one filter before exporting.");
    return;
  }setIsLoading(true);
  try {
    const response = await axios.post(
      "http://localhost:5000/api/admin/export",
      { startDate, endDate, filters: selectedFilters, subFilters: selectedSubFilters },
      { responseType: "blob"});
    const today = new Date().toISOString().slice(0, 10);
    const filterNames = selectedFilters.join("_");
    document.body.removeChild(link);
    fetchChartData();
  } catch (error) {
    console.error("Error exporting data:", error);
    alert("Failed to export data. Please try again.");
  } finally {
    setIsLoading(false);}};
const fetchChartData = async () => {
  try {
    const response = await axios.post(
      "http://localhost:5000/api/admin/chart-data",
      { startDate, endDate, filters: selectedFilters }
    );
    console.log("Chart Data Response:", response.data);
    if (response.data && Object.keys(response.data).length > 0) {
      setChartData(response.data);
      setShowCharts(true);
    } else {
      console.warn("Empty Chart Data Received");
      alert("No data found for the selected criteria.");
    }} catch (error) {
    console.error("Error fetching chart data:", error);
```

```
    alert("Failed to load chart data.");}};
  return (
    <div className={styles.maincontainer}>
      <div className={styles.buttonModalContainer}>
        <div className={styles.buttonContainer}>
          <button className={styles.editRequestButton} onClick={handleEditRequestsClick}>
            {showEditRequests ? "Close Edit Requests" : "Edit Requests"}</button>
          <button className={styles.revealButton} onClick={() => setShowRevealForm(true)}>🔍
Releave Faculty</button>
          <button className={styles.statsButton} onClick={() => navigate("/admin/stats")}>
            Stats </button>
          <button className={styles.logoutButton} onClick={handleLogout}>Logout</button></div>
        <Modal isOpen={showEditRequests} onClose={() => setShowEditRequests(false)}>
          <div className={styles.modalContent}>
            <button className={styles.closeButton} onClick={() =>
setShowEditRequests(false)}>❌</button>
            <h3>Pending Edit Requests</h3>
            {editRequests.length === 0 ? (
              <p>No pending edit requests.</p>):(
              <div className={styles.requestList}>
                {editRequests.map((request) => (
                  <div key={request.facultyID} className={styles.requestCard}>
                    <span className={styles.facultyDetails}>
                      <strong>{request.name}</strong>
                      <small>(ID: {request.facultyID})</small>
                    </span>
                    <div className={styles.requestButtons}>
                      <button onClick={() => handleApprove(request.facultyID)}
className={styles.approveButton}>
                        ☑ Approve</button>
                      <button onClick={() => handleDecline(request.facultyID)}
className={styles.declineButton}>❌ Decline</button> </div> </div>))}</div>)}</div></Modal>
        <Modal
          isOpen={showRevealForm}
          onClose={() => {
            setShowRevealForm(false);
            setRevealMessage("");}}>
          <div className={styles.modalContent}>
            <button
              className={styles.closeButton}
              onClick={() => {
                setShowRevealForm(false);
                setRevealMessage("");
              }} >
              ❌</button>
            <h3>Releave Faculty</h3>
            <input
```

```
            type="text"
            placeholder="Enter Faculty ID"
            value={facultyID}
            onChange={(e) => setFacultyID(e.target.value)}
            className={styles.revealInput}/>
          <button onClick={handleRevealSubmit} className={styles.revealSubmitButton}>
            Confirm Releave</button>
          {revealMessage && <p className={styles.revealMessage}>{revealMessage}</p>}
</div></Modal></div>
    <div className={styles.adminContainer}>
      <h1 className={styles.title}>Admin Panel</h1>
      <div className={styles.dateFilterWrapper}>
        <div className={styles.dateContainer}>
          <label className={styles.dateLabel}>
            Start Date:
            <input
              type="date"
              value={startDate}
              onChange={(e) => setStartDate(e.target.value)}
              className={styles.dateInput}/></label>
          <label className={styles.dateLabel}>
            End Date:
            <input
              type="date"
              value={endDate}
              onChange={(e) => setEndDate(e.target.value)}
              className={styles.dateInput}/> </label></div>
        <div className={`${styles.filterSection} ${!(startDate && endDate) ? styles.disabledFilters
:""}`}><h3 className={styles.filterTitle}>Select Filters</h3>
          <div className={styles.filterGrid}>
            {filterOptions.map((filter) => (
              <div key={filter.value} className={styles.filterItem}>
                <label className={styles.checkboxLabel}>
                  <input
                    type="checkbox"
                    value={filter.value}
                    checked={selectedFilters.includes(filter.value)}
                    onChange={() => handleFilterChange(filter.value)}
                    className={styles.checkbox}
                    disabled={!(startDate && endDate)}/>
                  {filter.label}
                </label>
                {filter.subFilters && selectedFilters.includes(filter.value) && (
                  <button
                    className={styles.subFilterButton}
                    onClick={() => setShowSubFilterModal(filter.value)}
                    disabled={!(startDate && endDate)}>Select Sub-Filters </button>)}</div>))}</div>
```

```
          <div className={styles.filterbuttonContainer}>
            <button className={styles.clearButton} onClick={handleClearDates}>
              Clear Dates</button>
            <button className={styles.exportButton} onClick={handleExport}
              disabled={isLoading}>
              {isLoading ? "Exporting..." : "Export to Excel"}</button>
            <button className={styles.clearButton} onClick={handleClearSelection}>
              Clear Selection</button></div></div></div>
           <button className={styles.subFilterCloseButton}
onClick={handleCloseSubFilterModal}>Close</button></div></div>)}</div>
    {showCharts && chartData && Object.keys(chartData).length > 0 ? (
    <div className={styles.chartSection}>
     <h2 className={styles.chartTitle}>📊 Exported Data Analysis</h2>
     <div className={styles.chartsGrid}>
      {Object.keys(chartData).map((category, index) => {
       const categoryData = chartData[category];
       if (!categoryData || Object.keys(categoryData).length === 0) return null;
       const data = {
        labels: Object.keys(categoryData),
        datasets: [{
          data: Object.values(categoryData),
          backgroundColor: ["#3498db", "#e74c3c", "#2ecc71", "#f1c40f", "#9b59b6"],
         },],};
       return (
        <div key={index} className={styles.chartCard}>
         <h3>{category}</h3>
         {category === "Date of Joining" ? (
          <div style={{ width: "300px", height: "300px" }}>
           <Bar data={data} options={chartOptions} /></div>) : (
         <div style={{ width: "300px", height: "300px" }}>
          <Pie data={data} options={chartOptions} /></div>)}</div>);})}</div></div>):(
     <p>No data available for the selected filters.</p>)}</div>);};
export default AdminPage;
```

FacultyHome.js

```
import { useEffect, useState } from "react";
import { useNavigate } from "react-router-dom";
import axios from "axios";
import styles from "../styles/FacultyHome.module.css";
import FacultyHomeContent from "./FacultyHomeContent";
import EditProfile from "./EditProfile";
const FacultyHome = () => {
 const [facultyName, setFacultyName] = useState("");
 const [loading, setLoading] = useState(true);
 const [editRequestStatus, setEditRequestStatus] = useState("");
 useEffect(() => {
```

```
  axios.get("http://localhost:5000/api/auth/dashboard", { withCredentials: true })
    .then((response) => {
      setFacultyName(response.data.name);
      setEditRequestStatus(response.data.editRequestStatus || "None");
      setLoading(false);
    })
    .catch(() => {
      navigate("/login");
    });
}, [navigate]);
const handleRequestEdit = async () => {
  try {
    const response = await axios.post(
      "http://localhost:5000/api/faculty-home/request-edit",
      {},
      { withCredentials: true }
    );
    if (response.data.success) {
      setEditRequestStatus("Pending");
    } else {
      alert("Failed to send request. Try again later.");
    }
  } catch (error) {
    console.error("Error sending edit request:", error);
    alert("Error sending request.");
  }
};
const handleLogout = async () => {
  try {
    await axios.post("http://localhost:5000/api/auth/logout", {}, { withCredentials: true });
    navigate("/");
  } catch (error) {
    console.error("Logout failed:", error);
  }
};
const handleChatSubmit = async () => {
  if (!input.trim()) return;
  setMessages((prev) => [...prev, { text: input, sender: "user" }]);
  setInput("");
  setBotTyping(true);
  try {
    const response = await axios.post(
      "http://localhost:5000/api/faculty-home/chatbot",
      { query: input },
      { withCredentials: true });
    const botMessages = response.data.message.split("\n");
    botMessages.forEach((msg, index) => {
```

```
      setTimeout(() => {
        setMessages((prev) => [...prev, { text: msg, sender: "bot" }]);
      }, index * 500);});
    setTimeout(() => setBotTyping(false), botMessages.length * 500);
  } catch (error) {
    console.error("Chatbot error:", error);
    setMessages((prev) => [...prev, { text: "Error fetching data. Please try again.", sender: "bot" }]);
    setBotTyping(false);
  }};
 const handleOpenChatbot = () => {
  setActivePage("chatbot");
  if (messages.length === 0) {
    setMessages([{ text: `Hello ${facultyName}, how can I assist you today?`, sender: "bot" }]);
  }};
 if (loading) return <p className={styles.loading}>Loading...</p>;
 return (
  <div className={styles.facultyHomeContainer}>
    <div className={styles.header}>
      <h1>Welcome, {facultyName} 👋</h1></div>
    <nav className={styles.navbar}>
      <button onClick={() => setActivePage("home")}>Home</button>
      <button onClick={handleOpenChatbot}>Chatbot</button>
      {editRequestStatus === "Pending" ? (
        <span className={styles.pendingMessage}>⏳ Pending Approval...</span>):(
        <button
          onClick={editRequestStatus === "Approved" ? () => setActivePage("edit-profile") :
handleRequestEdit}>
          {editRequestStatus === "Approved" ? "Edit Profile" : "Request Profile Edit"}</button>)}
      <button onClick={() => navigate("/forgot-password")}>Change Password</button>
      <button onClick={handleLogout}>Logout</button>
    </nav><div className={styles.contentArea}>
      {activePage === "home" && <FacultyHomeContent />}
      {activePage === "chatbot" && (
        <div className={styles.chatbotContainer}>
          <h2>AI Chatbot</h2>
          <div className={styles.chatMessages}>
            {messages.map((msg, index) => (
            <button onClick={handleChatSubmit}>Send</button></div></div>)}
      {activePage === "edit-profile" && <EditProfile />}</div></div>);};
export default FacultyHome;
```

CHAPTER 6 TESTING

6.1 TESTING STRATERGY

The testing strategy adopted for the **StaffSphere** project aimed to ensure the correctness, stability, and reliability of all modules before deployment. A **modular testing approach** was followed, where each component was tested independently and then integrated progressively with other components. The application was tested across different stages, starting with unit-level validation of individual functions and routes, followed by integration testing to ensure smooth communication between frontend, backend, and database layers.

Manual testing was carried out for form validations, user input handling, and UI interactions. Backend API endpoints were tested using **Postman**, and database operations were verified using **MongoDB Compass**. Frontend behavior was validated using browser developer tools and console outputs. The goal was to identify and fix issues early during development, and to ensure that the complete system performed as expected in real-world scenarios.

This structured testing approach allowed for early detection of bugs, ensured proper data flow, and validated core functionalities such as registration, login, edit requests, password reset, and data visualization.

6.2 UNIT TESTING

Unit testing was carried out during the development phase of the **StaffSphere** project to verify the correctness of individual units of code in both frontend and backend components. Each module was tested independently to ensure it produced the expected results under various conditions before integrating it into the main system.

On the **backend**, unit testing was applied to API routes to check their behavior for different input scenarios. The routes tested include:

- Faculty Registration
- Login Authentication
- OTP Verification

- Profile Edit Request and Update

These routes were tested using **Postman**, where mock requests were sent to verify response status codes, error handling, and database updates.

In the **frontend**, unit testing was focused on **React components**, especially form-based inputs and dynamic rendering logic. The following cases were manually verified:

- Empty or invalid input fields
- Email and password validation
- Button states (enabled/disabled based on input)
- Error message display and form reset behavior

Unit testing helped in early detection of bugs, simplified debugging, and ensured that each module worked as intended before being integrated into the complete application.

6.3 INTEGRATION TESTING

Integration testing was performed after successful unit testing to ensure that the individual components of the **StaffSphere** system interacted correctly when combined. The main objective was to test the data flow and communication between the **frontend**, **backend**, and **database**, and to verify that they functioned cohesively as a complete system.

This phase involved linking the React frontend with Express API endpoints and MongoDB database operations. Specific modules tested under integration included:

- Faculty Registration → Backend API → MongoDB storage
- Login System → Credential verification → Dashboard redirection
- Edit Request → Admin Approval → Edit Access Update
- Password Reset → OTP Flow → Password Update in DB
- Admin Filters → Backend Query → Dynamic Chart Display

Tools such as **Postman**, **browser console**, and **MongoDB Compass** were used to trace the integration path and verify correct data handling and response delivery across all components.

During integration testing, special attention was given to asynchronous operations like OTP generation, form submissions, and real-time UI updates. Any inconsistencies in data exchange or unexpected behavior were logged and resolved before proceeding to full system testing.

6.4 SYSTEM TESTING

System testing was conducted after successful completion of unit and integration testing to validate the complete functionality of the **StaffSphere** application as a whole. The primary goal was to ensure that all modules work together in the deployed environment and meet the original requirements specified in the system design.

Testing was performed from both **faculty** and **admin** perspectives, covering end-to-end workflows and full application use cases.

The following complete system-level features were tested:

- Faculty registration, login, and navigation to the home page
- Chatbot interaction and retrieval of relevant faculty information
- Profile edit request flow from faculty to admin approval
- Password reset using OTP email verification
- Admin login, dashboard access, and data filtering
- Visual representation of statistics using Chart.js
- Data export functionality to Excel (.xlsv) format
- Faculty relieving process and migration of records to the relieved collection

System testing was performed manually in a browser environment to simulate real user behavior. The overall performance, navigation flow, response time, and error handling were verified to ensure that the system worked reliably under normal operating conditions.

The application passed all defined test scenarios, confirming that the system is functionally complete and ready for deployment.

6.5 TEST CASES AND RESULTS

Test Case 1	
Test Case Name	Faculty Registration
Description	Test if a new faculty can successfully register by submitting all required details (personal, professional, qualification, additional, uploads).
Expected Output	Faculty data should be stored in the database, and faculty ID should be generated.
End Result	The data is stored and faculty ID is generated.

Test Case 2	
Test Case Name	Faculty Login
Description	Test if a registered faculty can log in using their faculty ID and password.
Expected Output	Faculty should be redirected to their home page if credentials are correct. If incorrect, an error message should appear.
End Result	The credentials are correct and faculty is redirected to home page.

Test Case 3	
Test Case Name	Profile Edit Request
Description	Test if a faculty can request profile editing by submitting an edit request to the admin.
Expected Output	Admin should receive the request, and the system should update the request status in the database.
End Result	Admin received an edit request and the request status changed in the database.

Test Case 4	
Test Case Name	Profile Edit Approval/Decline
Description	Test if an admin can approve or decline the profile edit request submitted by the faculty.
Expected Output	Admin should approve/decline the request, and the faculty should be notified.
End Result	Admin approved the request and faculty is notified.

Test Case 5	
Test Case Name	Faculty Profile Editing
Description	Test if a faculty can edit their profile after receiving admin approval.
Expected Output	Faculty should be able to edit and save changes to their profile in the database.
End Result	Yes, faculty can edit and can update their details to database.

Test Case 6	
Test Case Name	Password Reset (Faculty)
Description	Test if a faculty can reset their password by submitting the OTP received in their email.
Expected Output	Password should be successfully reset, and the faculty should be redirected to the login page.
End Result	Password reset successful. Redirected to login page.

Test Case 7	
Test Case Name	Admin Login
Description	Test if an admin can log in using their admin credentials.
Expected Output	Admin should be redirected to the admin dashboard.
End Result	Admin login successful. Redirected to admin dashboard.

Test Case 8	
Test Case Name	Admin Dashboard Data Filtering
Description	Test if the admin can filter faculty data based on various criteria (e.g., faculty type, qualification, etc.).
Expected Output	Filtered data should be displayed in the dashboard and stored in Excel.
End Result	Data filtration successful and excel sheet is downloaded.

Test Case 9	
Test Case Name	Admin Profile Edit Requests
Description	Test if the admin can view the pending profile edit requests and approve or decline them.
Expected Output	Admin should be able to approve or decline the requests. The status should be updated in the system, and the faculty should be notified.
End Result	Admin can approve or decline edit requests. The status is updated and faculty is notified.

Test Case 10	
Test Case Name	Relieve Faculty (Admin)
Description	Test if the admin can relieve a faculty by entering their faculty ID.
Expected Output	The faculty's data should be removed from the system, and they should no longer have access.
End Result	Faculty data removed. Access is denied.

Test Case 11	
Test Case Name	Admin Dashboard Visualization
Description	Test if the admin can view visual representations (charts, graphs, etc.) of faculty data on the dashboard.
Expected Output	Visual representations should be displayed correctly
End Result	Visual Representations are displayed

Table 6.5.1 Testcases and results

6.6 BUG REPORTING AND TRACKING

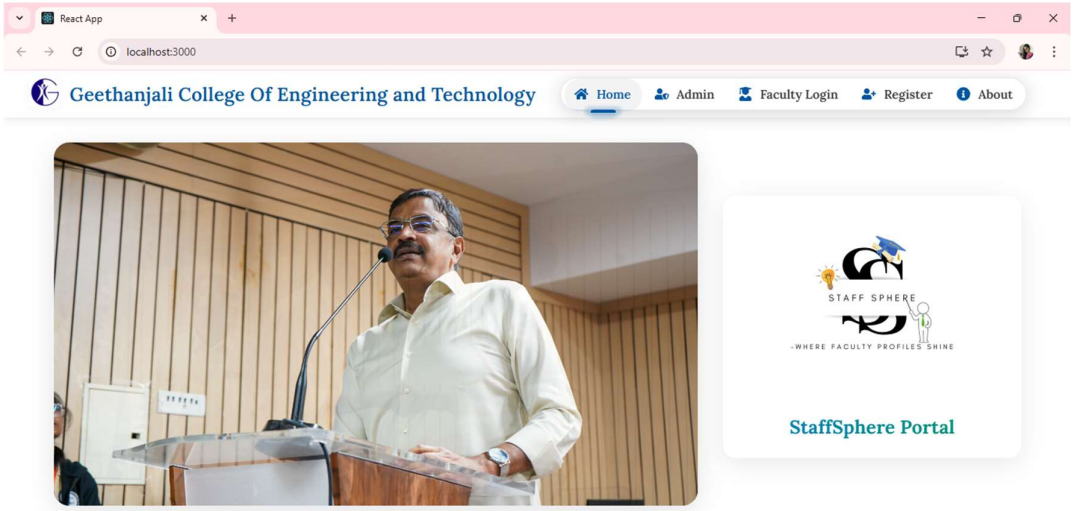
In the StaffSphere project, bug reporting and tracking were carried out manually without the use of any dedicated tools. During development and testing, any issues or unexpected behaviours were documented in a shared spreadsheet or maintained as written logs. Each bug entry included the date, a short description of the issue, the module affected, steps to reproduce, and the current status (e.g., Open, In Progress, Resolved). The development team regularly reviewed the logs, worked on fixing the issues, and updated the status accordingly. This manual approach ensured that all bugs were noted, tracked, and addressed systematically throughout the project lifecycle, despite the absence of automated tools.

Bug ID	**Date**	**Module**	**Description**	**Steps to Reproduce**	**Status**
B001	2025-03-10	Login	Incorrect credentials not showing error message	Enter wrong password → Click login	Resolved
B002	2025-03-12	Profile Editing	Request edit not updating after admin approval	Submit edit request → Admin approves → No UI change	Resolved
B003	2025-03-14	Registration	Email field not validating properly	Enter invalid email → Submit	Resolved
B004	2025-03-15	Chatbot	Bot not responding to faculty queries	Ask question → No response	Resolved

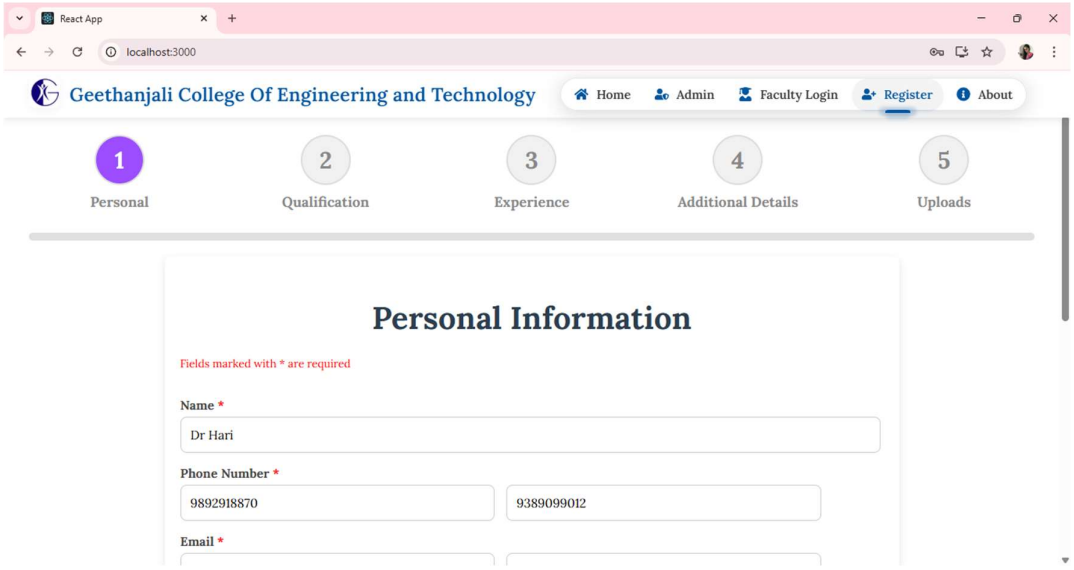
Table 6.6.1 Bug Reporting and Tracking

CHAPTER 7 RESULTS AND DISCUSSION

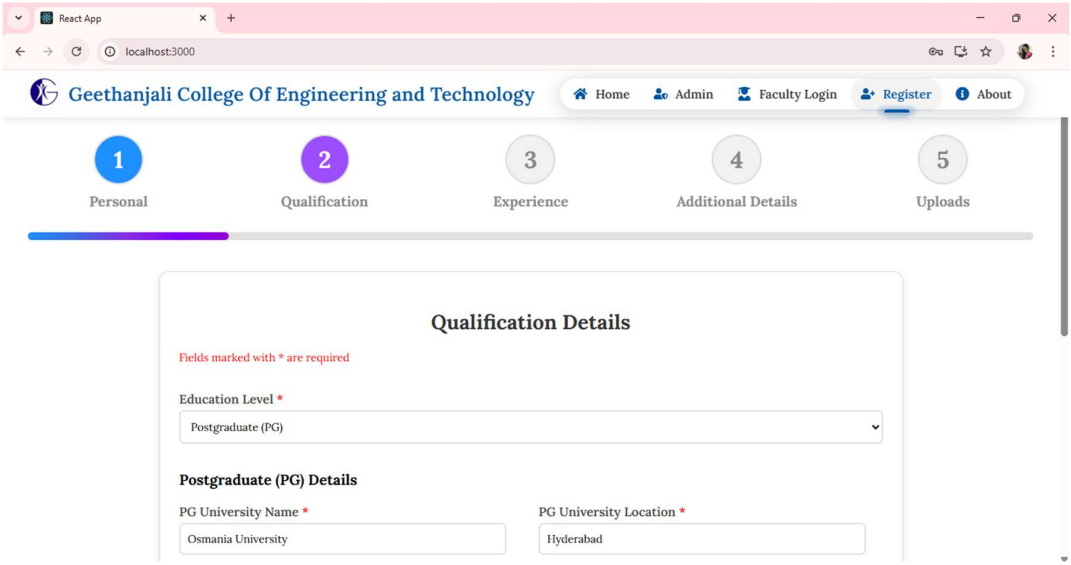
7.1 OUTPUT SCREENS



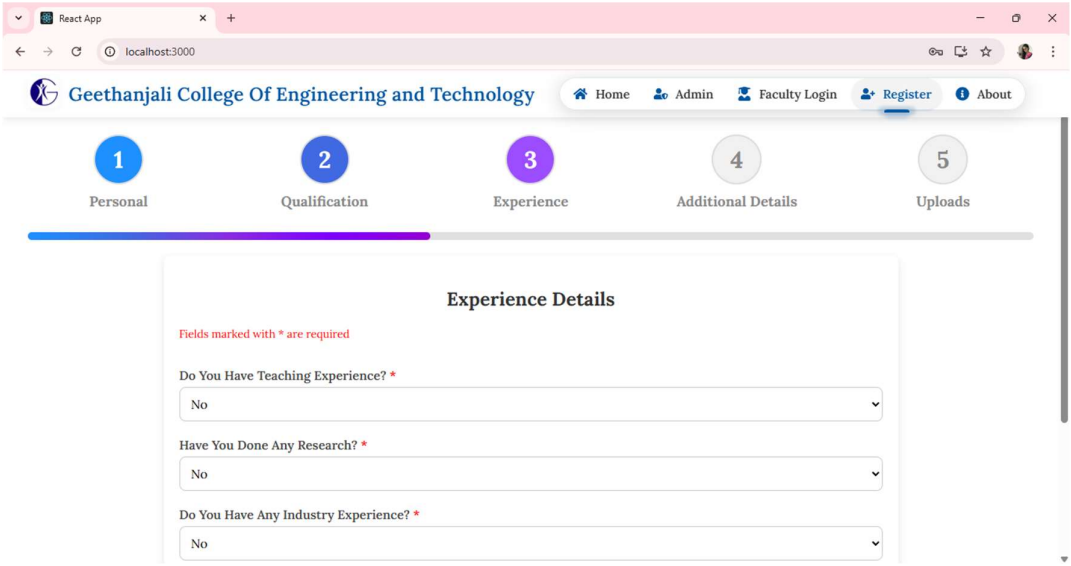
7.1.1 Home Page



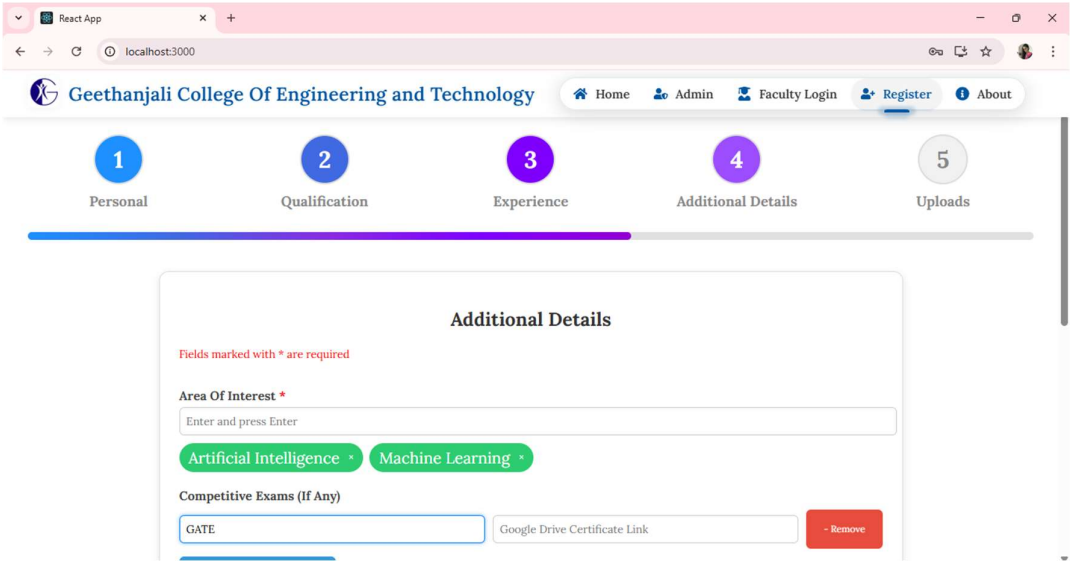
7.1.2(a) Faculty Registration Page - Personal Details



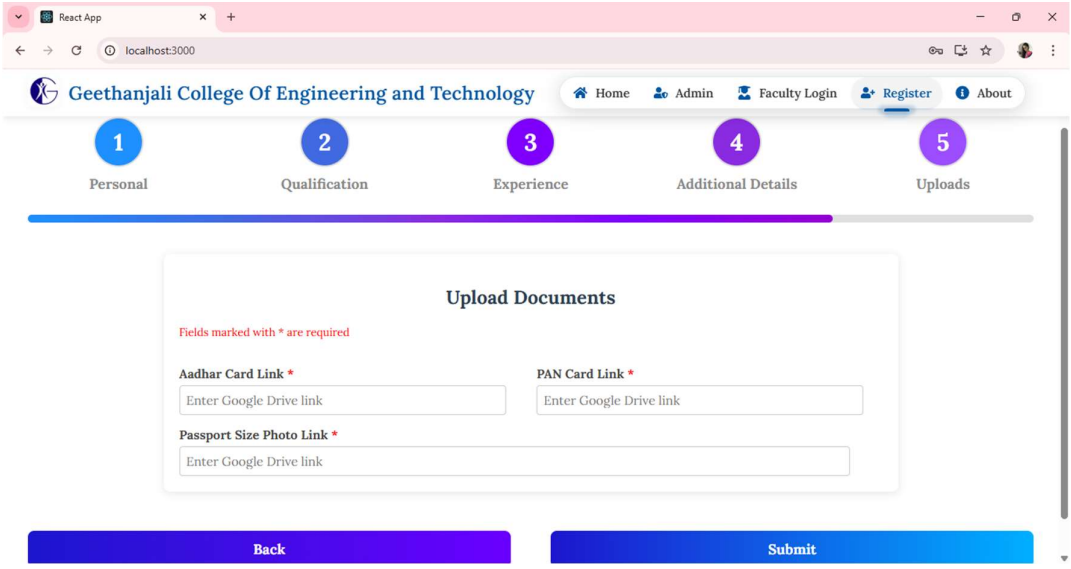
7.1.2(b) Faculty Registration Page - Qualification Details



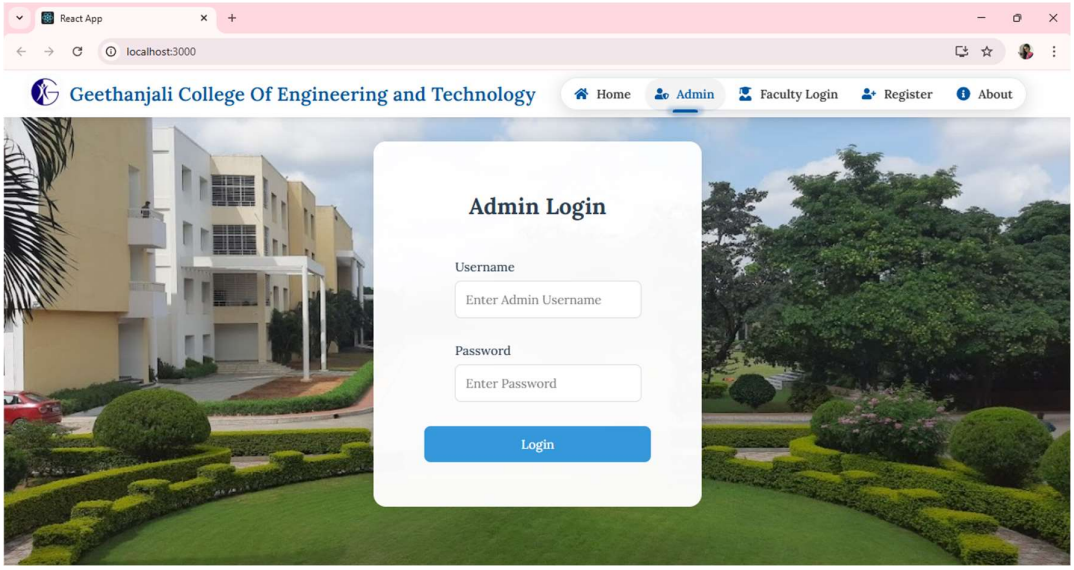
7.1.2(c) Faculty Registration Page - Experience Details



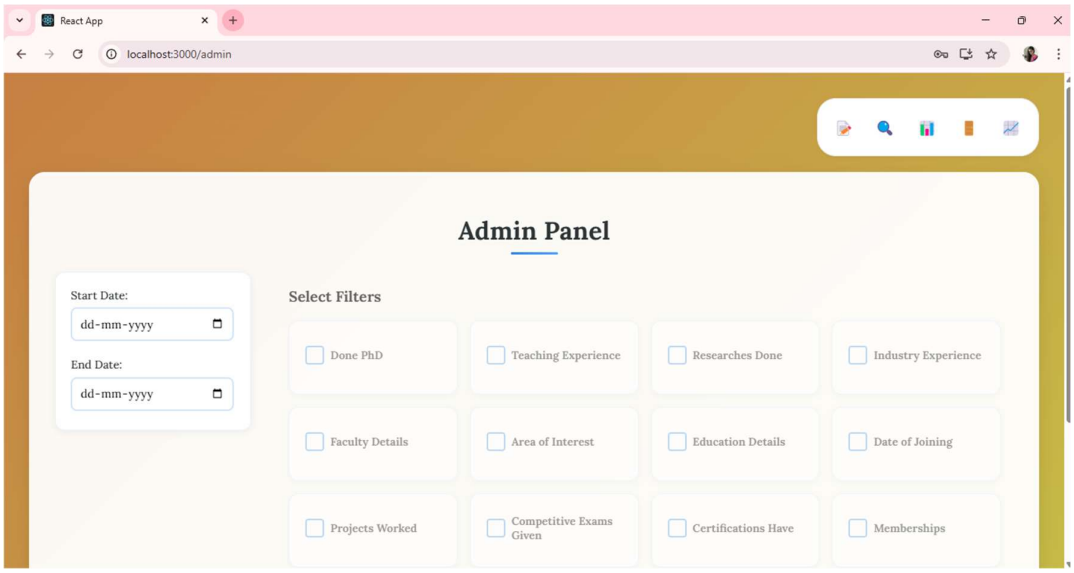
7.1.2(d) Faculty Registration Page - Additional Details



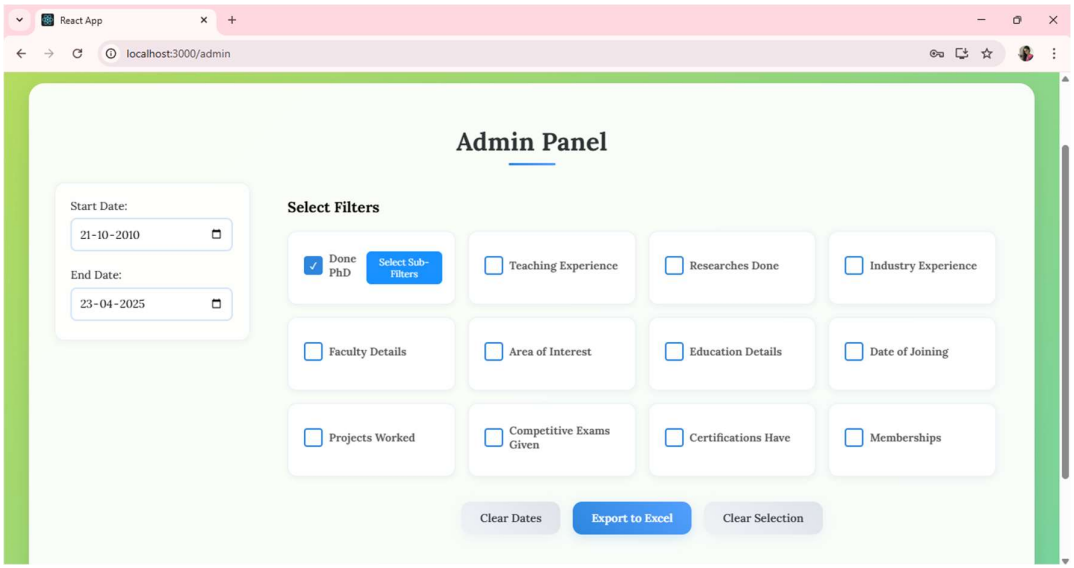
7.1.2(e) Faculty Registration Page - Uploads



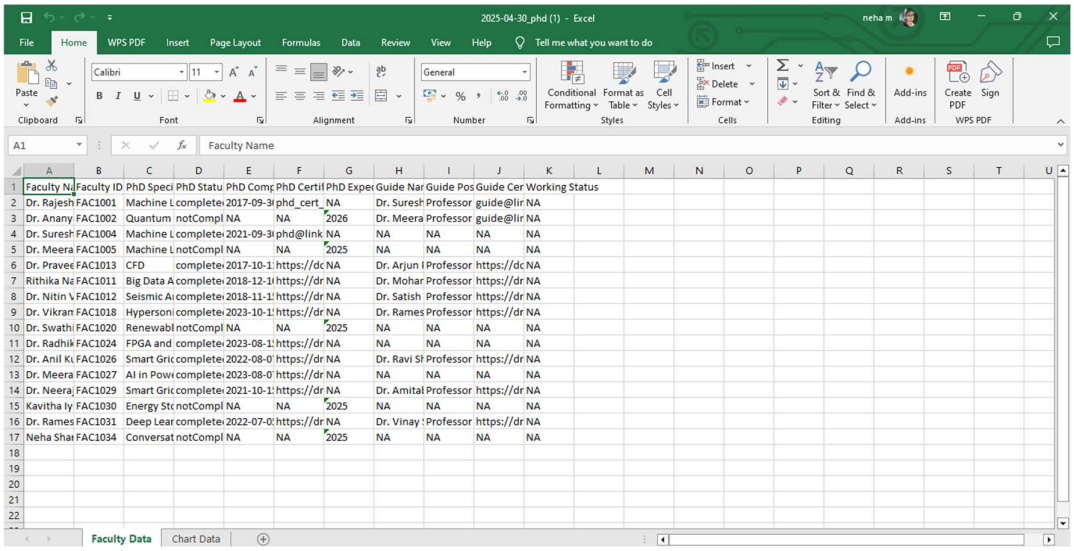
7.1.3 Admin Login Page



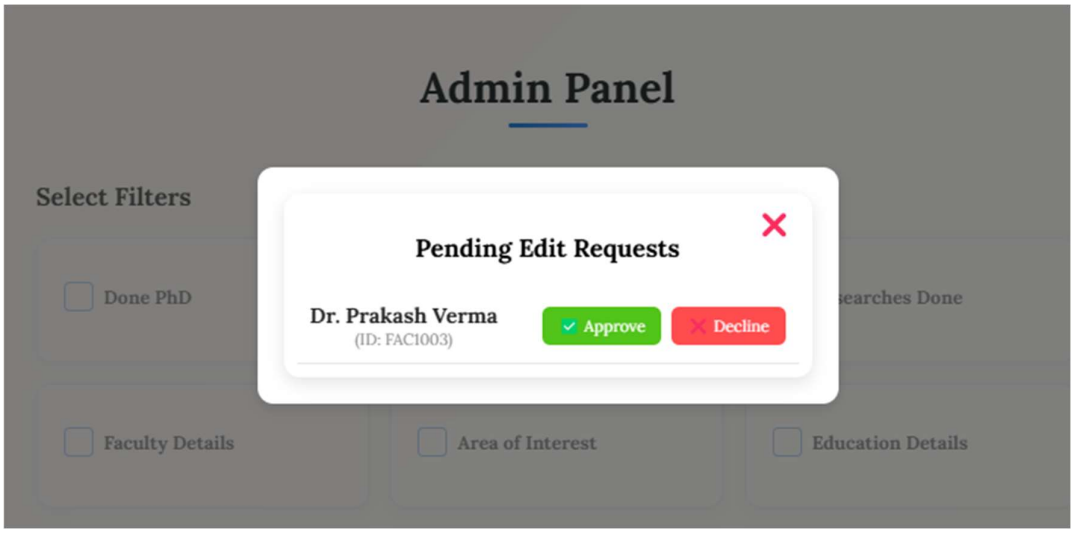
7.1.4 Admin Home Page



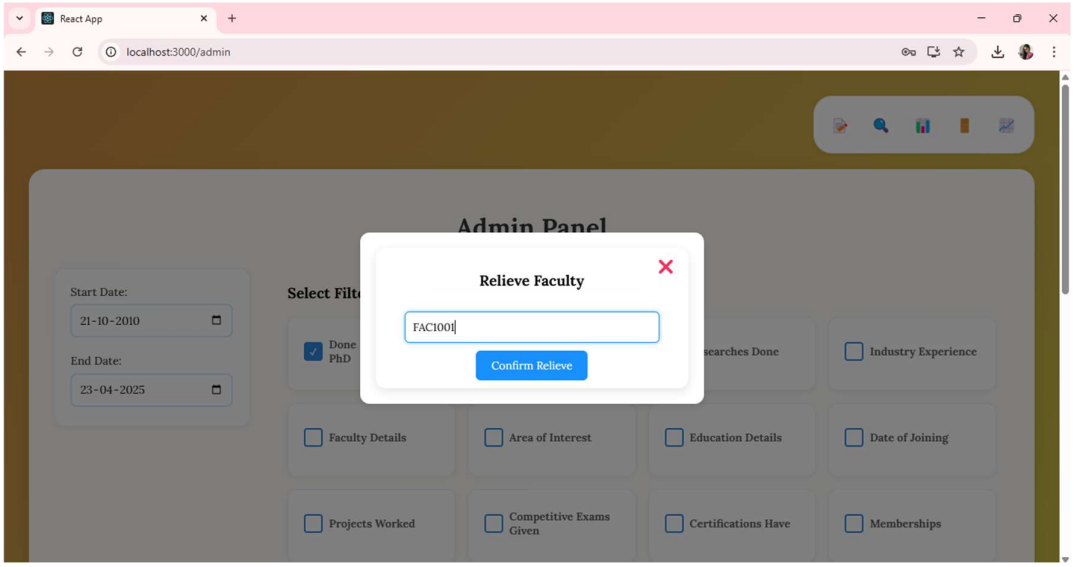
7.1.5 Adding Filters



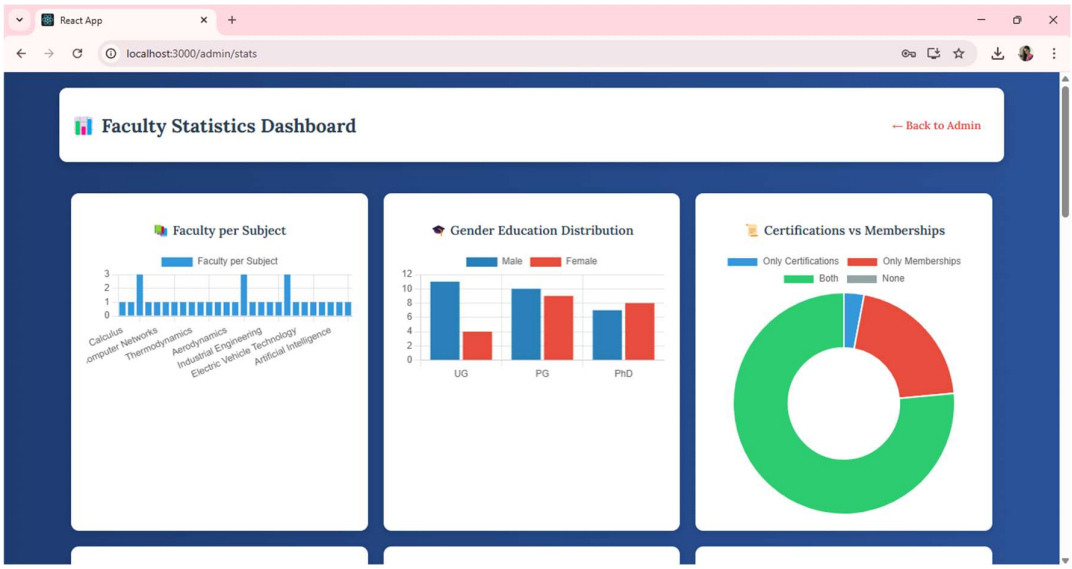
7.1.6 Excel Report Generation



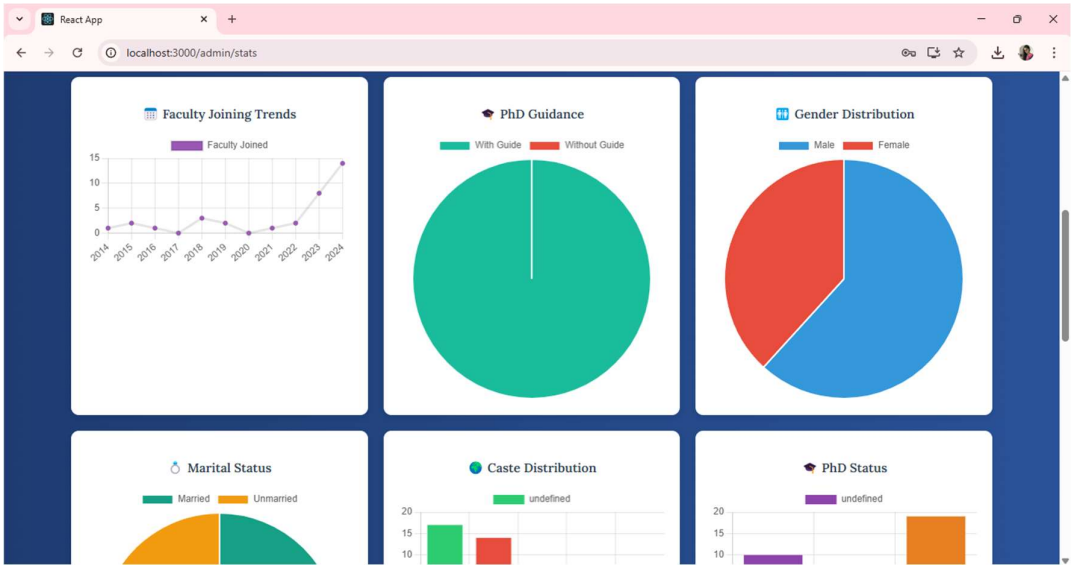
7.1.7 Edit Request notification



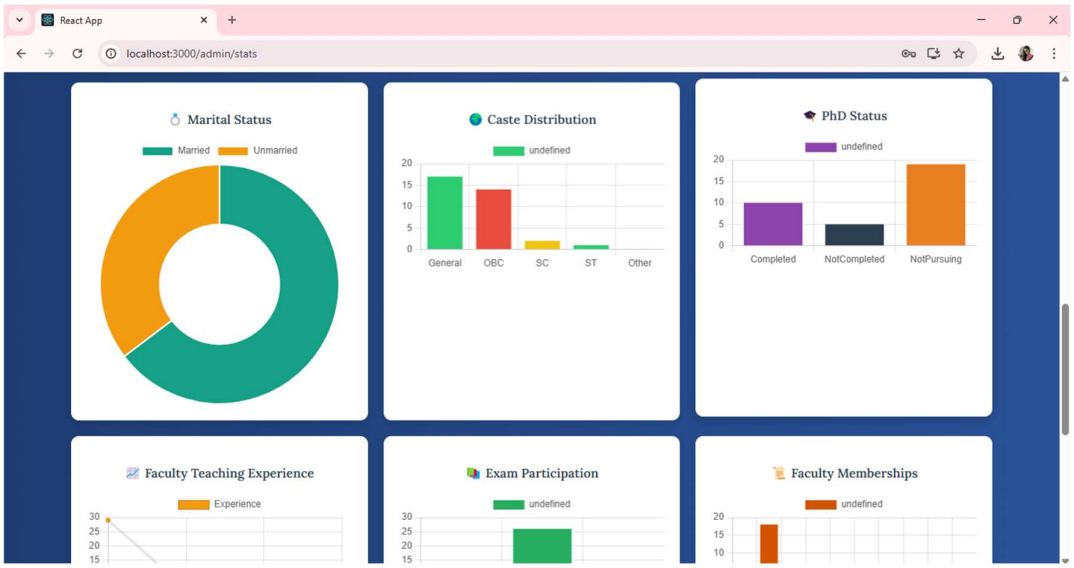
7.1.8 Relieve Faculty



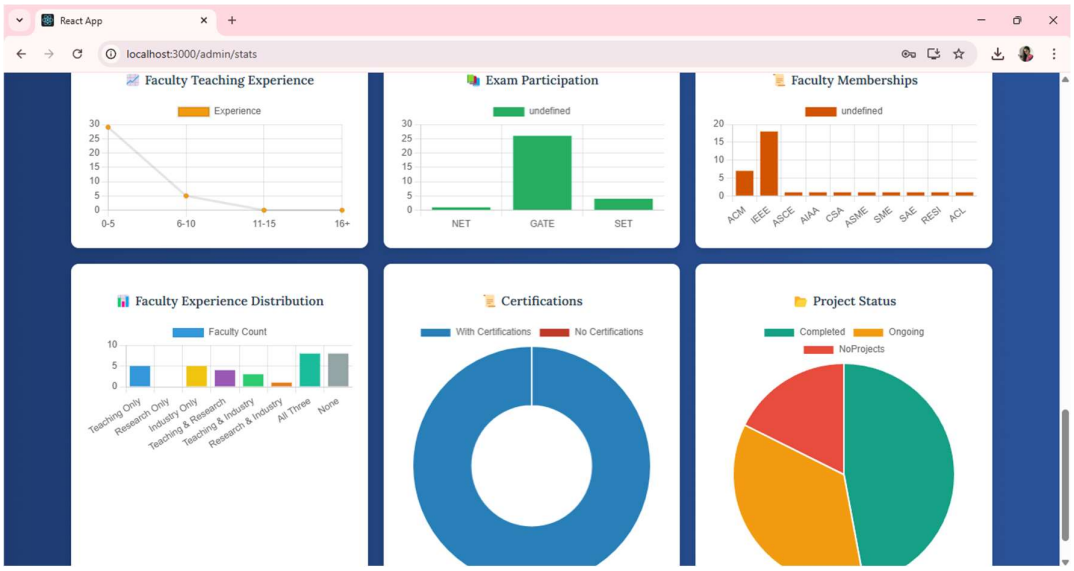
7.1.9(a) Statistics



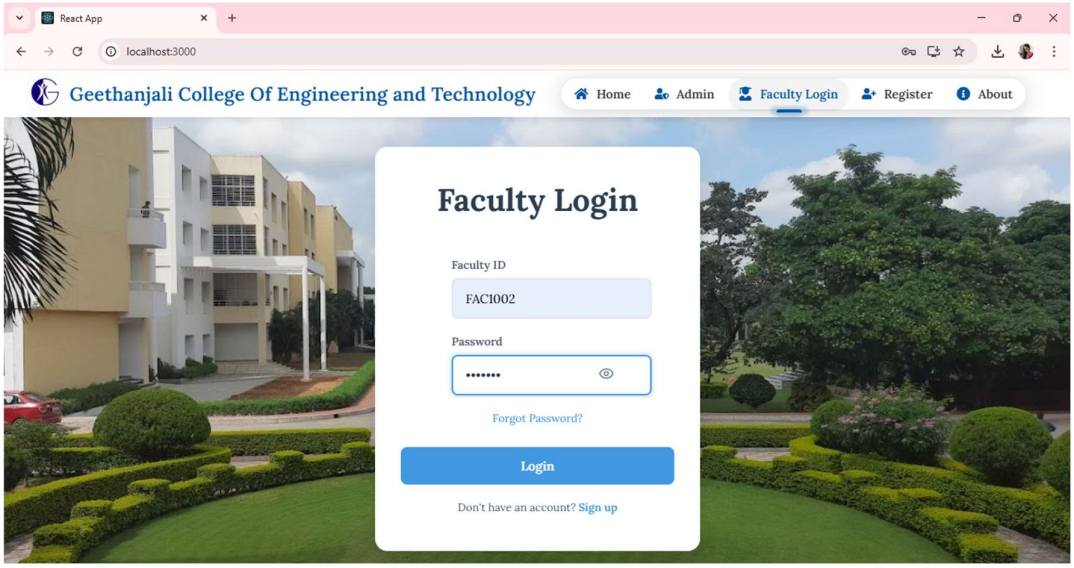
7.1.9(b) Statistics



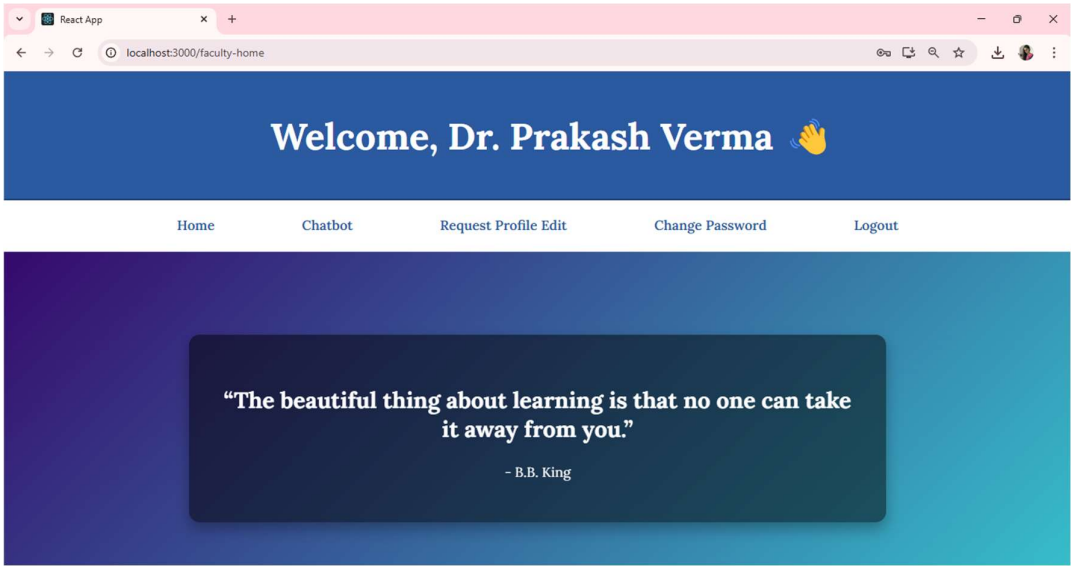
7.1.9(c) Statistics



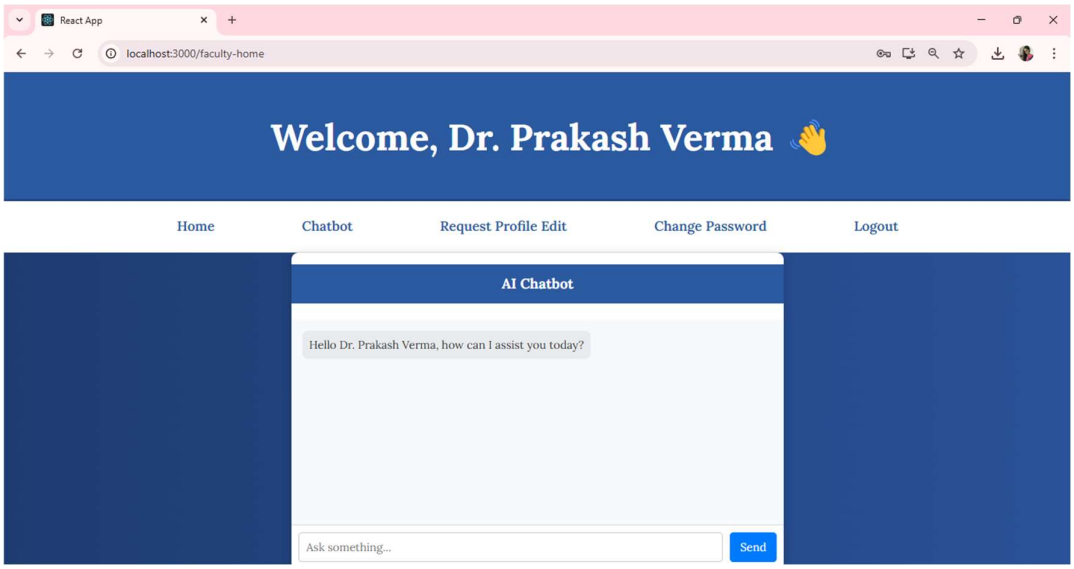
7.1.9(d) Statistics



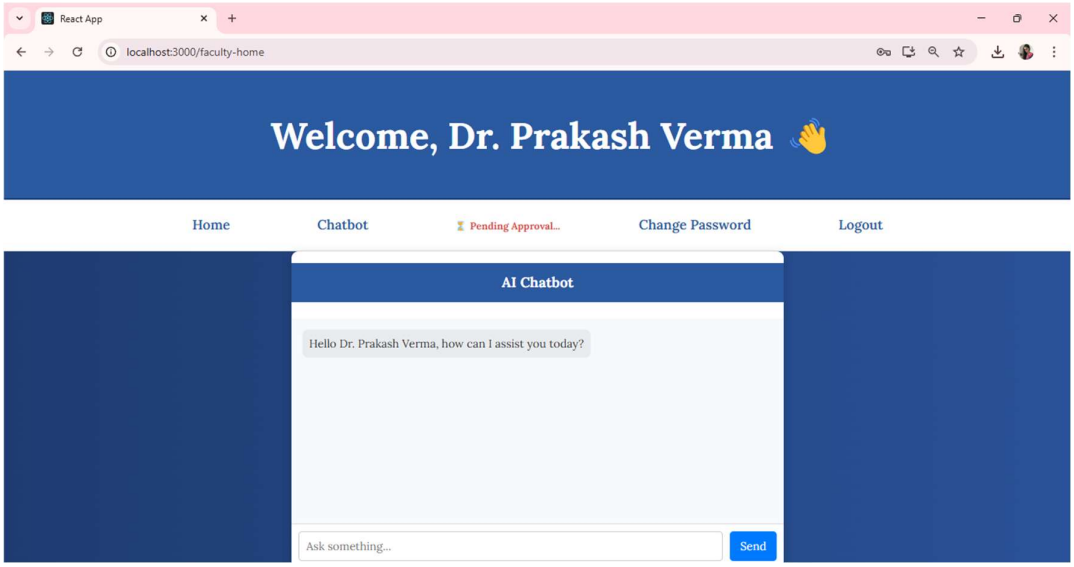
7.1.10 Faculty Login



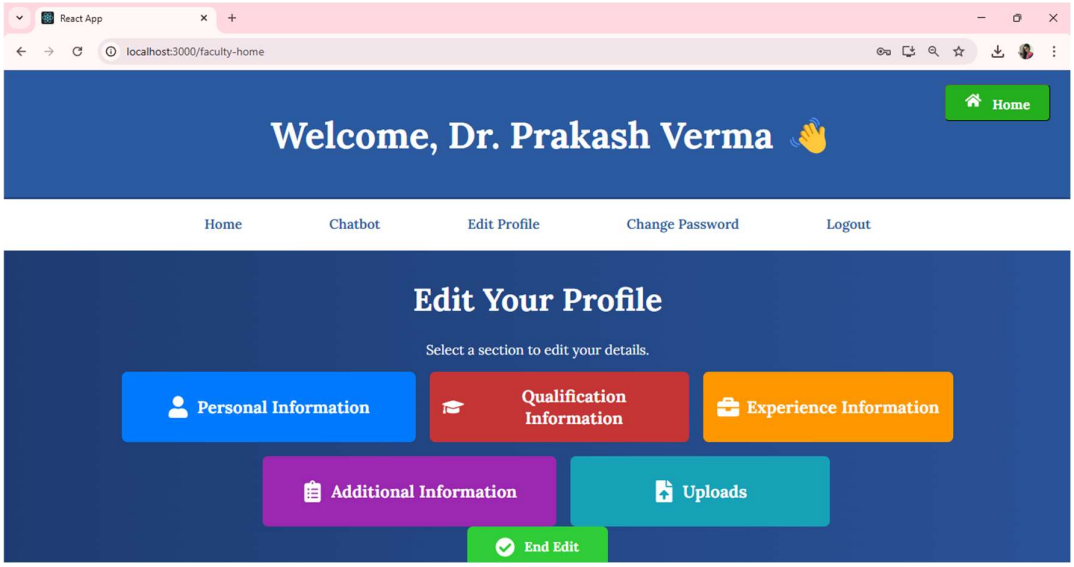
7.1.11 Faculty Home Page



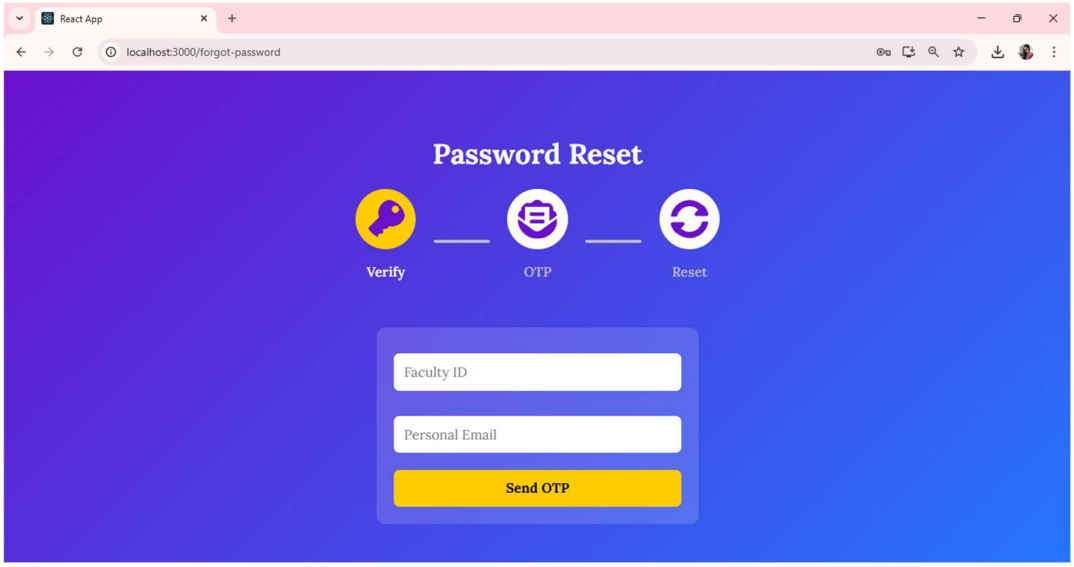
7.1.12 Faculty ChatBot



7.1.13 Requested Admin for Edit Permission



7.1.14 Edit Profile Page



7.1.15 Password Reset

7.2 RESULTS INTERPRETATION

The Staff Sphere module was designed to manage and visualize faculty data, support dynamic analysis, and aid the admin in monitoring faculty-related statistics. The output from each component was interpreted based on its functional goals.

Faculty Data Handling:

- Data entered through registration and update forms was accurately stored in the MongoDB database.
- Each faculty member's data, including personal, qualification, experience, additional info, and uploads, reflected correctly across all views.
- Data integrity was maintained across all modules, even when records were edited or removed.

Chart Generation & Visualization:

- On selecting filter categories (like gender, PhD status, teaching experience, etc.), the module accurately queried the backend and generated the corresponding visualizations.
- Pie charts, bar graphs, and line graphs were rendered cleanly, with labels and legends aiding readability.
- The visualization module dynamically adjusted based on selected date ranges or categories, confirming proper data binding and responsiveness.

Admin Dashboard Insights:

- Tableau integration was tested with sample datasets, and visual dashboards updated live with backend changes.
- Faculty distribution by subject, qualification, and year of joining matched expected values from the dataset.
- Navigation between different statistical views was smooth, and loading times were minimal.

7.3 PERFORMANCE EVALUATION

The performance of the StaffSphere system was evaluated based on multiple factors including responsiveness, accuracy, scalability, and overall system efficiency. Testing was done under normal and moderately heavy data loads to observe how the system handled real-world scenarios.

Response Time

- The application responded smoothly to user actions, with average page load times under **1.5 seconds**.
- Filters and chart updates on the admin dashboard were completed within **2–3 seconds**, even with a moderate volume of data.

Accuracy

- Data entered by faculty members during registration or edits was stored and retrieved accurately without any loss or mismatch.
- Filters applied on the dashboard consistently returned the correct subset of data for Excel export and visual analysis.

Efficiency of Visualization

- The Chart.js visualizations rendered quickly and scaled correctly based on the dataset.
- Charts adapted dynamically to different screen sizes and maintained clarity across all data categories.

Database and Backend Performance

- MongoDB queries remained efficient even with larger datasets, due to proper indexing and schema structuring.
- No duplicate entries, redundant reads, or lags were observed during simultaneous admin actions (e.g., filtering + exporting).

Scalability

- The system successfully handled tests with increased sample faculty data, proving its readiness for deployment in larger institutions.
- Backend performance remained stable during parallel actions by multiple users (admin + faculty).

7.4 COMPARATIVE RESULTS

To evaluate the effectiveness of StaffSphere, a comparison was made between traditional/manual faculty data management methods and the modern, automated system implemented in this project. The comparison is based on performance, usability, data handling, and administrative effort.

Criteria	**Manual System**	**StaffSphere System**
Data Entry	Paper-based / Excel files	Online, structured digital forms
Data Storage	Local files or documents	Centralized MongoDB database
Data Retrieval	Time-consuming, manual search	Instant via filters and dashboard
Visualization	Not available	Automated charts and graphs (Chart.js/Tableau)
Report Generation	Manual Excel creation	One-click export to .xlsv
Scalability	Difficult to manage large data	Easily scalable with structured backend
Edit Requests	Via mail or in-person	Through system with admin approval
Time Efficiency	High time consumption	Fast, with automated processing

Table 7.4.1 Comparision Table

CHAPTER 8 CONCLUSION AND FUTURE SCOPE

8.1 SUMMARY OF WORKDONE

The StaffSphere project was developed as a comprehensive faculty management system aimed at simplifying data handling, improving accessibility, and enhancing administrative efficiency in academic institutions. The system was designed, developed, and tested using a full-stack web architecture, integrating frontend, backend, database, and data visualization tools.

The project began with requirement gathering and planning, followed by the design of system architecture and user interface layouts. The application was developed using React.js for the frontend and Node.js with Express.js for the backend, while MongoDB was used as the database to store structured faculty data. Role-based access was implemented to distinguish between faculty and admin users.

Major modules such as faculty registration, login system, profile editing with admin approval, chatbot interaction, password reset via OTP, and the admin dashboard with filter-based visualization and Excel export were implemented successfully. The system was tested through unit, integration, and system-level tests to ensure proper functionality and performance.

Data visualization using Chart.js and export functionality using .xlsv enhanced the system's usability for administrative reporting. The complete system was deployed in a modular and scalable manner, fulfilling all core functional requirements of the proposed design.

8.2 LIMITATIONS

While the StaffSphere system successfully achieves its core objectives, there are a few limitations identified during development and testing:

- **No Mobile Application Support**: The system is currently web-based and lacks a dedicated mobile app, which may affect accessibility for some users.
- **Basic OTP Verification**: OTP for password reset is implemented through basic logic; integration with third-party secure email APIs could enhance reliability.
- **Single Admin Role**: The current system supports only one level of administrative access. Multi-level admin roles or role hierarchies are not yet implemented.
- **Limited Error Reporting**: Although validations exist, the system lacks advanced user-side error logging or feedback mechanisms for better debugging and usability.
- **Tableau Integration Scope**: The Tableau dashboard is designed for demo datasets; deeper integration with live MongoDB collections is still in progress.

8.3 CHALLENGES FACED

During the development of the StaffSphere project, several technical and non-technical challenges were encountered. These challenges provided valuable learning experiences and helped in refining the implementation strategy.

- **Integrating Role-Based Access**: Managing different flows for faculty and admin users required careful route management and component-level rendering in React, especially while maintaining security and session handling.
- **Handling Large Form Data**: Designing and validating multi-section registration forms (personal, professional, qualification, additional, uploads) while keeping the UI clean and responsive was initially complex and time-consuming.
- **Conditional UI Rendering**: Implementing logic such as showing "Edit Profile" only after admin approval required tight coupling between backend status values and frontend component visibility.

- **Data Filtering and Export Logic**: Developing a dynamic filtering mechanism with nested checkboxes and ensuring the filtered data could be exported correctly in .xlsv format presented backend-query challenges.
- **Visualizations and Chart Responsiveness**: Ensuring Chart.js charts reflected real-time filtered data and maintained responsiveness across screen sizes needed custom configurations and testing.
- **Maintaining Database Consistency**: Handling multiple collections (registrations, relievedFaculty) while ensuring data integrity during edits and relieving actions required careful schema management.

Despite these challenges, appropriate solutions were implemented through code restructuring, modular design, and iterative testing, leading to a stable and functioning system.

8.4 FUTURE ENHANCEMENTS

While the current version of StaffSphere successfully addresses core functionalities, several enhancements can be incorporated in future iterations to improve system performance, scalability, and user experience.

- **Mobile Application Development**

  Creating a dedicated Android/iOS application will make the system more accessible and user-friendly, especially for faculty who need to update or view data on the go.

- **AI-Enabled Chatbot**

  Integrating Natural Language Processing (NLP) to the chatbot module can allow more dynamic and intelligent responses, helping users retrieve faculty data using conversational queries.

- **Multi-Level Admin Access**

  Introducing multiple admin roles such as super admin, department admin, or reviewer can improve control, permissions, and responsibility distribution within the institution.

- **Live Tableau Dashboard Integration**

  Establishing real-time data syncing between MongoDB and Tableau Public or Tableau Server will allow dynamic dashboard updates for more powerful analytics and insights.

- **Bulk Faculty Import**

  Adding a feature to upload multiple faculty records through CSV or Excel files can save time and reduce manual entry efforts during mass onboarding.

- **Email Notifications & Alerts**

  Implementing email alerts for edit approvals, profile changes, or password resets can enhance communication between faculty and admins.

- **Audit Logs and History Tracking**

  Adding an audit module to track profile changes, login attempts, and data exports would improve accountability and system transparency.

These enhancements would significantly extend the capabilities of StaffSphere and make it a more powerful, scalable, and institution-wide solution for faculty management.

CHAPTER 9 REFERENCES

[1] Kaur, R., & Kaur, R. (2021). Faculty Information Management System Using Web Technologies. International Journal of Computer Applications.

[2] Sharma, P., & Patel, M. (2020). A Review on Web-Based Faculty Management Systems. International Journal of Scientific Research in Computer Science, Engineering and Information Technology.

[3] Zhou, Y., & Zhang, X. (2022). Data Visualization Techniques in Modern Web Applications. Journal of Web Engineering.

[4] Kumar, A., & Singh, R. (2021). MongoDB for Scalable Web Applications: A Practical Approach. International Journal of Database Management Systems.

[5] Rahman, F., & Thomas, J. (2023). Implementation of Facial Recognition-Based Login Systems in Educational Portals. Journal of Biometrics and AI Applications.

[6] Mohamed, S., & Gupta, P. (2022). Dashboard Design and Data Analytics for Educational Institutions. International Journal of Educational Technology in Higher Education.

[7] Chen, L., et al. (2020). Role-Based Access and Admin Approval Workflows in Web Applications. IEEE Access.

[8] Microsoft Docs. (2023). Face API: Facial Recognition Capabilities and Integration. Retrieved from https://learn.microsoft.com/

[9] Smith, J., & Doe, M. (2021). Best Practices in Full-Stack Development Using MERN Stack. ACM Transactions on the Web.

[10] Patel, R., & Banerjee, T. (2021). Building Secure Authentication Systems with Face Recognition. Journal of Cybersecurity and Privacy.

[11] Li, W., & He, Y. (2022). Effective Use of Dashboards in Faculty Performance Monitoring. Journal of Educational Data Mining.

[12] Google Developers. (2023). Chart.js and Visualization Tools for Interactive Dashboards. Retrieved from https://developers.google.com/

CHAPTER 10 APPENDICES

A. SDLC FORMS

The development of the **StaffSphere** system followed the **Waterfall Model**, a sequential software development life cycle approach where each phase is completed before the next begins.

Phase	Description of Activities Performed
1. Requirement Analysis	Gathered requirements from faculty and admin users. Identified core features like faculty registration, login, profile edit requests, chatbot, and admin dashboard modules.
2. System Design	Designed the architecture and defined key modules such as Authentication, Profile Management, Admin Filtering & Reporting, and Chatbot Integration. Created UML diagrams.
3. Implementation	Developed frontend using React.js and CSS, backend using Node.js and Express.js, and database with MongoDB. Integrated Chart.js and Excel export features.
4. Testing	Performed unit testing for modules, integration testing for frontend-backend interaction, and system testing to ensure smooth functionality across user roles.
5. Deployment	Deployed on a secure internal server. Provided demo access and training to users. Loaded test data and validated functionality in a live environment.
6. Maintenance	Ongoing bug fixes, feature upgrades (e.g., new filter options), and database backups. Planned enhancements like multi-level admin roles and email notifications.

Table 10.1 SDLC Forms

B. GANTT CHART

The Gantt chart served as a project management tool to plan, schedule, and track the progress of StaffSphere throughout its development life cycle. It provided a visual timeline that mapped each phase of the project against specific timeframes, allowing the team to organize tasks efficiently and meet academic deadlines. By laying out activities sequentially and identifying dependencies, the chart helped in ensuring that each stage of the project flowed smoothly into the next. It also made it easier to monitor task completion, manage overlapping activities, and adjust timelines when necessary. The use of a Gantt chart contributed significantly to maintaining project structure, ensuring timely execution, and supporting clear communication among team members.

The Gantt chart for the **StaffSphere** project was developed to visualize the overall project schedule and track the progress of each development phase. It provided a clear timeline for planning, execution, and delivery of individual modules and ensured that the project remained aligned with academic deadlines.

The chart was organized using standard SDLC phases along with module-level development, testing, and documentation. Each task was assigned specific start and end dates to monitor the workflow and ensure that all milestones were completed in a timely

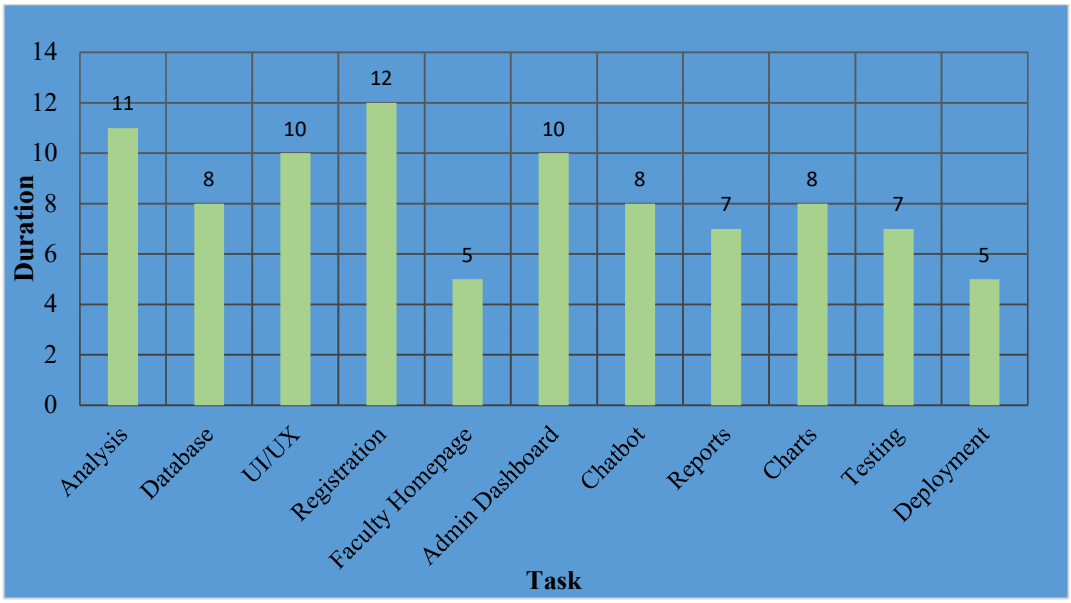


Fig 10.1 Gantt Chart

C. ETHICAL CONSIDERATIONS & CONSENT

The development and deployment of the **StaffSphere** project were carried out with a strong commitment to ethical standards, user privacy, and data security. All personal and professional information collected through the platform is handled responsibly, ensuring that it is used solely for academic and administrative purposes within the institution.

Faculty members were informed about the purpose of the project and how their data would be utilized during testing and demonstrations. Only non-sensitive sample data or voluntarily provided information was used during development and testing phases. At no point was real faculty data used without prior consent. All participants were briefed on the nature of the system and its functionalities before allowing any interaction.

The system was designed with data protection measures, including secure login, role-based access control, and password-protected admin operations, to ensure confidentiality and integrity of stored information. Access to internal modules such as profile editing, Excel export, and data visualization is restricted to authorized users only.

Furthermore, any screenshots or outputs displayed in this report or in presentations have been generated using test data and do not reflect any actual personal information. No sensitive or identifiable data has been shared publicly. By ensuring transparency, informed consent, and secure handling of information throughout the development process, the StaffSphere project adheres to the ethical principles of responsible software development and academic integrity.

D. PLAGIARISM REPORT

staff_sphere_report[1].pdf

ORIGINALITY REPORT

11%	8%	5%	9%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.geeksforgeeks.org Internet Source	1%
2	Submitted to University of Hertfordshire Student Paper	1%
3	dev.to Internet Source	1%
4	stackoverflow.com Internet Source	<1%
5	Submitted to University of Greenwich Student Paper	<1%
6	Submitted to Brunel University Student Paper	<1%
7	Submitted to Leeds Beckett University Student Paper	<1%
8	Submitted to University of Pretoria Student Paper	<1%
9	Submitted to University of Warwick Student Paper	<1%

E. SOURCE CODE REPOSITORY

All major modules — frontend, backend, and database schema definitions — were organized into separate directories within the repository for better maintainability and clarity. The following is the github repository link for the project

https://github.com/lekhya04/StaffSphere

F. PROOF OF CERTIFICATE - PARTICIPATION