A MINI PROJECT REPORT ON Email Sync – Automated Email and Event Management System

as a part of

MCA223007: Mini Project

Submitted by

Sumedh Ahire 03

Madhura Ashture 04

Vedant Bhosale 09

Pratik Kankarej 33

Guided by

Prof. Archana L. Rane

Master of Computer Applications
(Under Faculty of Engineering)

Department of MCA
K. K. Wagh Institute of Engineering Education and Research
Hirabai Haridas Vidyanagari, Amrutdham, Panchavati,
Nashik – 422 003

K. K. WAGH INSTITUTE OF ENGINEERING EDUCATION AND RESEARCH, NASHIK - 422003

CERTIFICATE

This is to certify that

Sumedh Ahire
Madhura Ashture
Pratik Kankarej
Vedant Bhosale

has successfully completed the Mini Project on

Email Sync –
Automated Email and Event Management System

as a part of

MCA223007: Mini Project

during the Academic Year 2024 – 25

Prof.**Archana Rane**	**Dr. V. C. Bagal**	**Dr. K. N. Nandurkar**
Project Guide	I/c Head,	Director,
Dept. of MCA	Dept. of MCA	KKWIEER, Nashik

ABSTRACT

In today’s fast-paced digital world, effective email and event management is crucial for both personal and professional productivity. The “Email Sync - Automated Email and Event Management System” project aims to address this need by developing a robust, automated system that streamlines the management of emails and events. This system is designed to notify users of new emails, track the status of emails (accepted or rejected), and maintain a comprehensive record of all sent and received emails. By leveraging advanced technologies and a well-structured database, the project seeks to enhance user efficiency and organization.

The "Email Sync - Automated Email and Event Management System" project investigates the need for an efficient email and event management system to enhance productivity and organization. The goal is to develop a robust, automated system that notifies users of new emails, tracks email statuses, and maintains a comprehensive record of all email interactions.

The system is built around a relational database with tables for Posts, Notifications, Structure, Status, and History. The backend is developed using Java, Spring Boot, and Hibernate, providing a scalable and robust architecture. The frontend, developed with HTML, CSS, and JavaScript, offers a user-friendly interface. Integration with external email servers is achieved through APIs, and deployment on cloud platforms ensures scalability and reliability.

The system efficiently handles large volumes of emails, providing real-time notifications and updates. It enhances user productivity by reducing the time and effort needed to manage emails and events. The preliminary results indicate robust performance under various conditions without degradation.

The "Email Sync - Automated Email and Event Management System" revolutionizes email and event management through automation, providing a comprehensive, user-friendly solution that significantly enhances productivity and organization.

Keywords:

Automation, Email Management, Event Management, Java, Real-time Notifications, Spring Boot, User Productivity, Email History , Status.

TABLE OF CONTENTS:

LIST OF FIGURES

LIST OF TABLES:

ABBREVIATIONS AND SYMBOLS

- SDLC : Software Development Life Cycle

- STLC : Software Test Life Cycle

Risk Severity :

Color	Severity
🟩	Low
🟨	Moderate
🟥	High

1. INTRODUCTION

The "Email Sync" project is an innovative endeavour aimed at developing an automated system to streamline the management of emails and events. Recognizing the challenges posed by handling large volumes of emails manually, this system is designed to notify users of new emails in real-time, track the status of these emails (such as accepted, rejected, or pending), and maintain a detailed, comprehensive record of all email interactions. This approach not only enhances user productivity but also ensures better organization and efficiency in managing communications.

The core of the project lies in its robust relational database, which is structured with five key tables: Post, Notify, Structure, Status, and History. Each table plays a vital role in ensuring the smooth functioning of the system. The Post table handles email details, while the Notify table manages notifications sent to users. The Structure table maintains the organizational framework of the emails, and the Status table keeps track of the email statuses. The History table logs all interactions, providing a complete record for future reference.

By leveraging advanced technologies and a meticulously designed database, the "Email Sync" project aims to revolutionize email and event management, offering a comprehensive, user-friendly solution that meets the demands of modern digital communication. This project promises to significantly reduce the time and effort required for managing emails, thereby boosting overall efficiency and productivity for users.

Objective :

The "Email Sync - Automated Email and Event Management System" project aims to address this necessity by developing a robust system that automates the management of emails and events, enhancing user efficiency and organization through advanced technologies.

1.1 Detailed Problem Definition

The primary issue addressed by this project is the inefficiency in managing large volumes of emails and events. Traditional email systems often lack automation, leading to missed emails, poor organization, and a lack of comprehensive records. Users struggle with keeping track of the status of emails (accepted, rejected, pending) and are overwhelmed by the manual effort required to maintain email histories and notifications.

1.2 Current Market Survey

A market survey indicates a growing demand for automated email and event management solutions. Existing systems often provide basic functionalities but lack the sophistication required for real-time notifications, status tracking, and comprehensive record management. Major players in the market, such as Google and Microsoft, offer email services but do not fully integrate advanced automation tailored to individual needs. This gap presents an opportunity for innovative solutions like the "Email Sync" system.

1.3 Need of the System

The need for the "Email Sync - Automated Email and Event Management System" is underscored by the increasing volume of digital

communication and the necessity for efficient management tools. Users require a system that not only notifies them of new emails in real-time but also tracks the status of these emails and maintains a detailed history of all interactions. Such a system would significantly enhance productivity by reducing the manual effort involved in managing emails and events.

1.4 Advances / Additions / Updates to the Previous System

The "Email Sync" project brings several advancements over traditional email management systems:

- Automation: Automated email notifications and real-time status tracking.
- Integration: Seamless integration with external email servers via APIs.
- User Interface: A user-friendly dashboard for managing emails and events.
- Scalability: Deployment on cloud platforms like AWS or Azure ensures scalability and security.
- Data Management: A robust relational database to maintain comprehensive records of all email interactions.

1.5 Organization of the Report

This report is organized into the following sections:

- Analysis:

  Detailed project plan, requirement analysis, and team structure.
- Design:

  Software requirement specification, risk assessment, and discussion of the project plan.
- Modelling :

  UML diagrams, DFDs, ERD, and database normalization.
- Implementation:

  Description of the development process, tools used, and code snippets.
- Testing and Validation:

  Test plans, test cases, results, and validation procedures.
- Conclusion:

  Summary of findings, recommendations, and lessons learned from the project.

2.REQUIREMENT ANALYSIS

Requirement Analysis is the initial phase in software development that gathers the user requirements and analysis. It gathers all the functional and non-functional requirements from user, stakeholders or client.

Requirements from User's Point of View

1. Necessary Functions

- Real-time Email Notifications: Users need to be notified of new emails as soon as they arrive.
- Email Status Tracking: Ability to track the status of emails (accepted, rejected, or pending) in real-time.
- Comprehensive Email Records: Maintain detailed records of all sent and received emails.
- User-friendly Interface: An intuitive and easy-to-navigate dashboard for managing emails and events.
- Secure Data Management: Ensure that all email data is stored securely and complies with privacy regulations.

2. Desirable Functions

- Customizable Notifications: Ability to customize notification settings based on user preferences.
- Integration with Calendar Systems: Seamless integration with calendar systems for event management.
- Analytics and Reporting: Provide insights and reports on email interactions and engagement.
- Web Application: Availability of a web app for managing emails and events on-the-go.

3. Other Requirements

- Multi-language Support: Support for multiple languages to cater to a diverse user base.
- Accessibility Features: Ensure the system is accessible to users with disabilities.
- Regular Updates: Continuous updates based on user feedback.

Table 2.2.1 depicts all the requirements which are modified by the developer referring the user's requirement .

Modified Requirements from Developer's Point of View:

1. Necessary Functions

These are those requirements that are basic requirements that is supposed to be present in the application /system.

Table 2.2.1:List Of Necessary Requirements

Sr.no	Function	Implementation	Action needed
1	Real-time Email Notifications	Develop a robust notification system using web sockets or push notifications to alert users instantly.	Security: Encrypt notifications to protect user privacy.
2	Email Status Tracking	Design a tracking system that updates email status in real-time using a relational database.	Scalability: Ensure the tracking system can handle a large volume of email status updates efficiently.
3	Comprehensive Email Records	Use Hibernate ORM for efficient data management and storage.	Security: Implement data encryption and access control measures to protect email records.
4	User-friendly Interface	Develop a responsive frontend using HTML, CSS, and JavaScript frameworks like React or Angular.	Usability Testing: Conduct usability testing to ensure the interface meets user expectations.
5	Secure Data Management	Use Spring Security for authentication and authorization.	Compliance: Ensure the system complies with GDPR and other data privacy regulations.

2. Desirable Functions

The desired functionality is that a user/client expects to be present in system other than the basic requirements. These are some extra functions expected by user.

Table 2.2.2: List of Desired Requirements modified by Developer.

Sr.no	**Function**	**Implementation**	**Action needed**
1	Customizable Notifications:	Provide user settings for customizing notification preferences.	Flexibility: Allow users to choose notification types (email, SMS, push notifications).
2	Integration with Calendar Systems:	Develop APIs to integrate with popular calendar systems like Google Calendar and Outlook.	Syncing: Ensure seamless syncing of events between the email management system and calendar systems.
3	Analytics and Reporting:	Use data visualization tools and libraries to create insightful reports.	Performance: Optimize the analytics engine for real-time data processing.

3. Other Requirements

Table 2.2.3 :List of Other Requirements modified by Developer.

Sr.no	Function	Implementation	Action needed
1	**Multi-language Support:**	Use internationalization (i18n) libraries to support multiple languages.	**Localization:** Ensure the system adapts to local cultures and formats.
2	**Accessibility Features:**	Follow WCAG guidelines to make the system accessible	**Testing:** Conduct accessibility testing to ensure compliance.
3	**Regular Updates:**	Establish a continuous integration and continuous deployment (CI/CD) pipeline for regular updates.	**Feedback Loop:** Collect user feedback to inform updates and improvements. .

3.ANALYSIS:

Project analysis is to comprehensively understand the requirements and constraints of the "Email Sync - Automated Email and Event Management System" project, ensuring that the development process is well-informed and aligned with user needs and technical feasibility.

3.1 Project Plan:

The project plan outlines the roadmap for developing the "Email Sync - Automated Email and Event Management System." It includes detailed timelines, milestones, and resource allocations to ensure the project stays on track and meets its objectives.

The plan is divided into several phases:

Phase 1: Initiation **(**2 weeks)

1. **Define Project Scope and Objectives**
   - Identify project goals, deliverables, constraints, and success criteria.
   - Outline the project scope and key objectives.
2. **Stakeholder Identification and Engagement**
   - Identify all stakeholders (users, business owners, developers).
   - Develop a stakeholder engagement plan.
3. **Project Charter**
   - Create a project charter outlining the project's purpose, scope, and participants.
   - Obtain approval from key stakeholders.

Phase 2: Planning **(**4 weeks)

1. **Requirement Gathering**

   The requirements specified in Table 2.2.1 , Table 2.2.2 and Table 2.2.3 is referred along with requirement gathering ways is considered.
   - Conduct stakeholder interviews, surveys, and workshops.
   - Document detailed functional and non-functional requirements.
2. **Feasibility Study**
   - Perform technical, operational, and economic feasibility studies.
   - Document the findings in a feasibility report.
3. **Project Plan Development**
   - Develop a detailed project plan including timelines, milestones, and deliverables.
   - Create a Work Breakdown Structure (WBS) to detail project tasks.

4. **Risk Management Plan**

    - Identify potential risks and develop mitigation strategies.
    - Document the risk management plan.

5. **Resource Allocation**

    - Identify required resources (team members, hardware, software).
    - Allocate resources and assign tasks to team members.

Phase 3: Design **(**6 weeks)

1. **System Architecture Design**

    - Define the overall system architecture (frontend, backend, database, APIs).
    - Document the architectural design.

2. **Detailed Design**

    - Create detailed design documents for each module (User, Email, Notification, Status, History).
    - Develop Data Flow Diagrams (DFDs) and Entity-Relationship Diagrams (ERDs).

3. **User Interface Design**

    - Develop wireframes and mock-ups for the user interface.
    - Conduct usability testing and incorporate feedback.

4. **API Design**

    - Define API endpoints, request/response formats, and authentication mechanisms.
    - Document API specifications.

5. **Component Design**

    - Create class diagrams and sequence diagrams for each component.
    - Specify interactions between components.

Phase 4: Development **(**12 weeks)

1. **Setup Development Environment**

    - Set up the development environment with necessary tools and software.
    - Configure version control using Git.

2. **Backend Development**

    - Develop the backend using Spring Boot and Java.

- Implement database interactions using Hibernate.

3. **Frontend Development**

   - Develop the user interface using HTML, CSS, and JavaScript.
   - Implement responsive design using Thymeleaf.

4. **API Integration**

   - Develop and test RESTful APIs for communication with external email servers.
   - Ensure secure and reliable API interactions.

5. **Unit Testing**

   - Write and execute unit tests for each component using JUnit.
   - Ensure high code coverage and fix any identified defects.

Phase 5: Testing **(**4 weeks)

1. **Integration Testing**

   - Test the integration of frontend, backend, and APIs.
   - Ensure seamless communication between components.

2. **System Testing**

   - Conduct end-to-end testing of the entire system.
   - Verify that the system meets all functional and non-functional requirements.

3. **Performance Testing**

   - Test the system's performance under various conditions (load, stress, scalability).
   - Optimize performance based on test results.

4. **Security Testing**

   - Conduct security testing to identify vulnerabilities.
   - Implement security measures to protect user data.

5. **User Acceptance Testing (UAT)**

   - Engage users in testing the system.
   - Collect feedback and make necessary adjustments.

Phase 6: Deployment **(**2 weeks)

1. **Prepare Deployment Plan**

- Develop a detailed deployment plan outlining steps for go-live.
- Ensure that all dependencies are addressed.

2. **Deploy to Production**

- Deploy the system to the production environment (AWS or Azure).
- Perform final checks to ensure everything is functioning correctly.

3. **Monitor and Support**

- Set up monitoring tools to track system performance.
- Provide user support and address any post-deployment issues.

Phase 7: Maintenance and Updates **(ongoing)**

1. **Continuous Monitoring**

- Monitor system performance and user feedback.
- Address any issues that arise promptly.

2. **Regular Updates**

- Plan and implement regular updates based on user feedback and new requirements.
- Ensure that the system remains up-to-date and relevant.

3. **Documentation**

- Maintain comprehensive documentation of the system, including user manuals and technical guides.
- Update documentation with each new release.

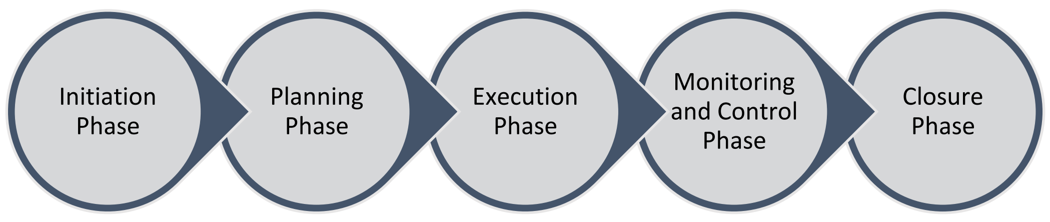


Fig.3.1 : Project Planning Phase

3.2 Requirement Analysis:

Requirement analysis involves gathering and documenting the needs and expectations of stakeholders to ensure the project delivers a valuable solution. Key requirements for the "Email Sync - Automated Email and Event Management System" include:

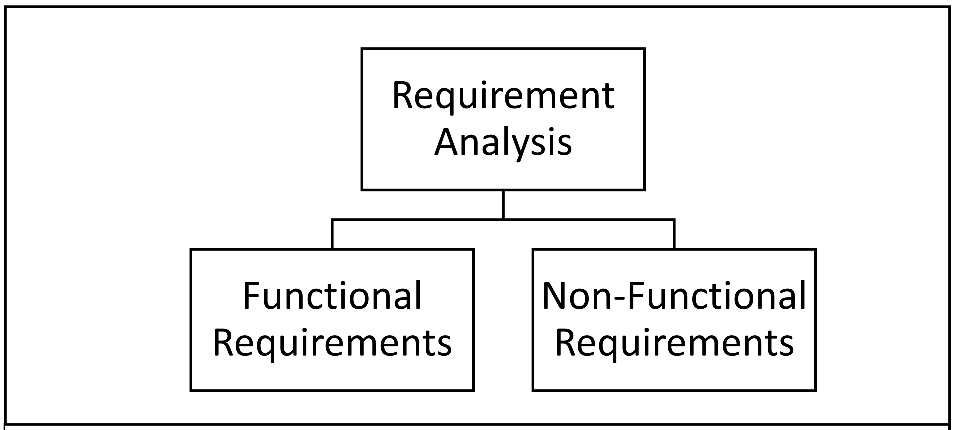


Figure 3.2: Types of Requirements

- Functional Requirements:

1. Email Notification System: Notify users of new emails in real-time.
2. Email Status Tracking: Track the status of emails (accepted, rejected, pending).
3. Email Record Management: Maintain a comprehensive history of all sent and received emails.
4. User Interface: Develop a user-friendly dashboard for managing emails and events.
5. Integration with Email Servers: Seamlessly send and receive emails through APIs.

- Non-Functional Requirements:

1. Scalability: Ensure the system can handle large volumes of emails without performance degradation.
2. Reliability: Guarantee consistent system performance and availability.
3. Security: Protect user data and ensure compliance with data privacy regulations.
4. Usability: Provide an intuitive and easy-to-navigate interface.

2.3 Team Structure:

Key roles include:

- Project Manager: Oversees the entire project, manages timelines and resources, and ensures stakeholder communication.
- Backend Developers: Develop the backend system using Java, Spring Boot, and Hibernate.
- Frontend Developers: Create the user interface using HTML, CSS, and JavaScript.

4. DESIGN:

The design phase is critical in defining the architecture, components, interfaces, and data for the "Email Sync" system. This phase ensures that the system is well-organized, efficient, and scalable.

4.1 Software Requirement Specification (SRS)

The Software Requirement Specification (SRS) document serves as a detailed description of the functional and non-functional requirements for the "Email Sync - Automated Email and Event Management System."

Functional Requirements:

- Email Notification System:
  - Notify users of new emails in real-time.
  - Provide a user interface to view notifications.
- Email Status Tracking:
  - Track the status of emails (accepted, rejected, or pending).
  - Update status in real-time.
- Email Record Management:
  - Store comprehensive records of all sent and received emails.
  - Allow users to search and retrieve historical emails.
- User Interface:
  - Dashboard for managing emails and events.
  - Intuitive design for ease of use.

Non-Functional Requirements:

- Scalability: Handle large volumes of emails efficiently without performance degradation.
- Reliability: Ensure consistent system performance and high availability.
- Security: Protect user data and ensure compliance with data privacy regulations.
- Usability: Provide an easy-to-navigate interface that requires minimal training

4.2 Risk Assessment:

Risk assessment involves identifying potential risks, assessing their likelihood and impact, and developing strategies to mitigate them.

Table 4.1:Risk Assessment Matrix

Sr.no.	Risk Identified	Likelihood	Impact	Mitigation
1	Data Quality Issues	High	High	Implement robust data validation and cleaning processes.
2	Skill Gaps	Medium	High	Provide training programs to enhance team skills in AI and machine learning.
3	Regulatory Changes	Medium	Medium	Stay updated on regulations and ensure compliance.
4	Cybersecurity Threats	Low	High	Implement stringent security measures and regular audits.
5	Vendor Reliability	Medium	Medium	Establish service level agreements (SLAs) with vendors and have contingency plans.

Table 4.1 depicts how the risk is analyse , its severity and how a risk is mitigated in a software development phase.

4.3 Major Milestones and the Work Done as Per It:

Project Plan:

The initial project plan submitted in Semester I outlined the roadmap for developing the "Email Sync - Automated Email and Event Management System." The plan included key milestones, resource allocation, and timelines.

Major Milestones:

1. Requirement Gathering (Weeks 1-2):
   1. Conduct stakeholder interviews.
   2. Document requirements.
2. Design Phase (Weeks 3-6):
   1. Develop SRS document.
   2. Create system architecture and database design.
3. Development Phase (Weeks 7-14):
   1. Implement backend with Java, Spring Boot, and Hibernate.
   2. Develop frontend with HTML, CSS, and JavaScript.
   3. Integrate APIs for external email servers.
4. Testing Phase (Weeks 15-18):
   1. Conduct unit, integration, and system testing.
   2. Ensure compliance with all functional and non-functional requirements.
5. Deployment Phase (Weeks 19-20):
   1. Deploy the system on computer / PC.
   2. Monitor initial performance and make necessary adjustments.

Figure 4.1 depicts how the process of implementation is done. A chart representing the flow of the development is given below .

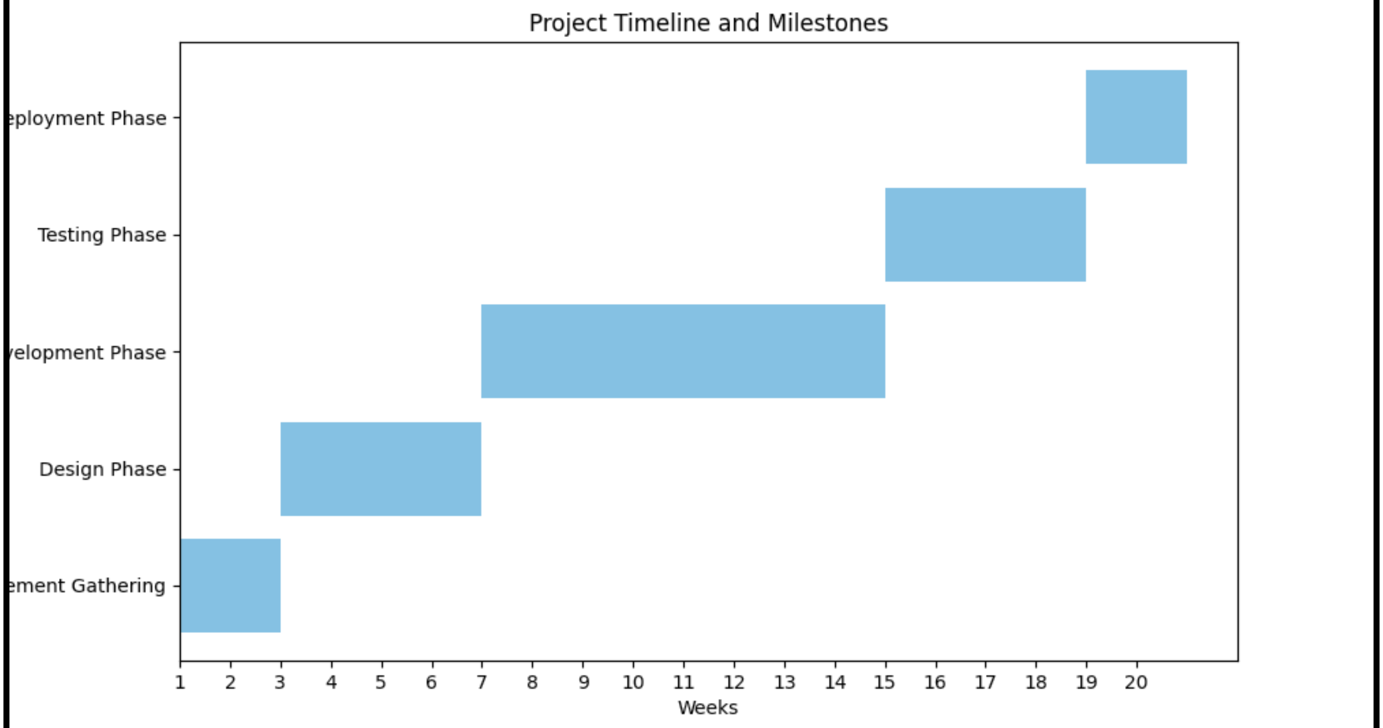


Figure 4.1 : Gantt Chart for the project timeline and milestones.

Work Done as Per the Plan:

- Requirement Gathering: Completed with detailed documentation of functional and non-functional requirements.
- Design Phase: Finalized the SRS, system architecture, and database design.
- Development Phase: Backend and frontend development completed as planned. APIs integrated successfully.
- Testing Phase: Initial testing conducted with high accuracy and performance. Identified and fixed minor bugs.
- Deployment Phase: Deployed with successful initial performance monitoring.

5.MODELING

This phase of software development involves implementing and creating the system’s components. Here, first the blueprint of system is used which is then given as the reference for creating the actual design of system.

5.1 UML Diagram:

UML (Unified Modelling Language) Diagrams are typically created first. They are used to model the overall system architecture and its interactions, focusing on how different components interact with each other. Here are some common UML diagrams:

- **Class Diagrams**:

  Describe the structure of the system by showing its classes, attributes, and relationships. Fig. 5.1.2 shows how the classes are present in this project and how they are related to one another.

- **Sequence Diagrams**:

  Detail the sequence of interactions between objects in different scenarios.

- **Activity Diagrams**:

  Model the workflow of the system's operations.

- **Use Case Diagrams**:

  Captures the functional requirements of the system.Fig.5.1.1 shows the use case diagram for this project.

Actors in Use Case Diagram:

User, Co-ordinator, Guest

Use Cases in Use Case Diagram :

1. Compose Email
2. Send Email
3. Receive Email
4. Notify User
5. Track Email Status
6. Maintain Email History
7. Manage Notifications

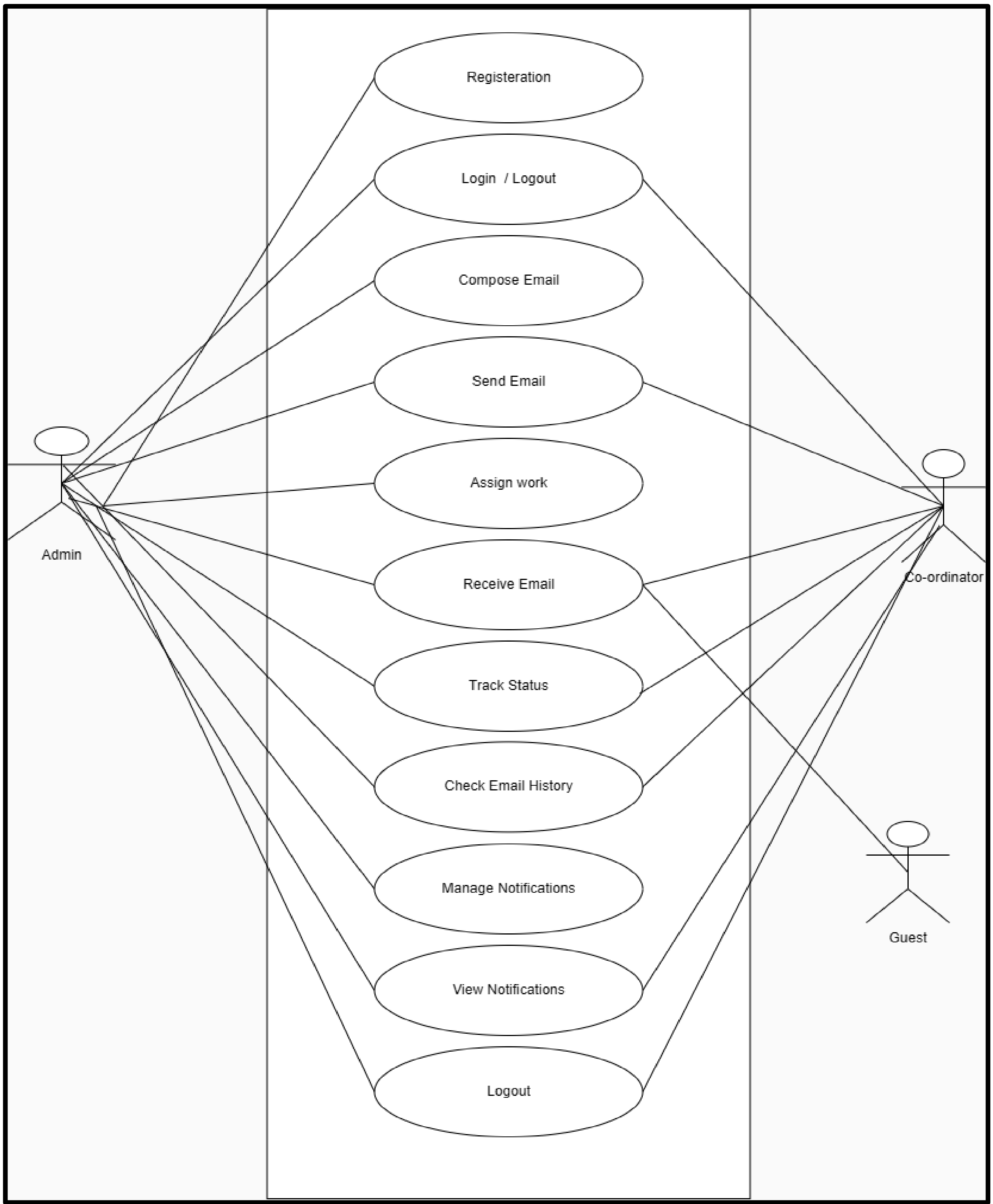


Figure 5.1.1: Use Case Diagram

B) Class Diagram:

List of Classes in a diagram :

1)User,

2) Email,

3)Notifications,

4)History,

5)Status

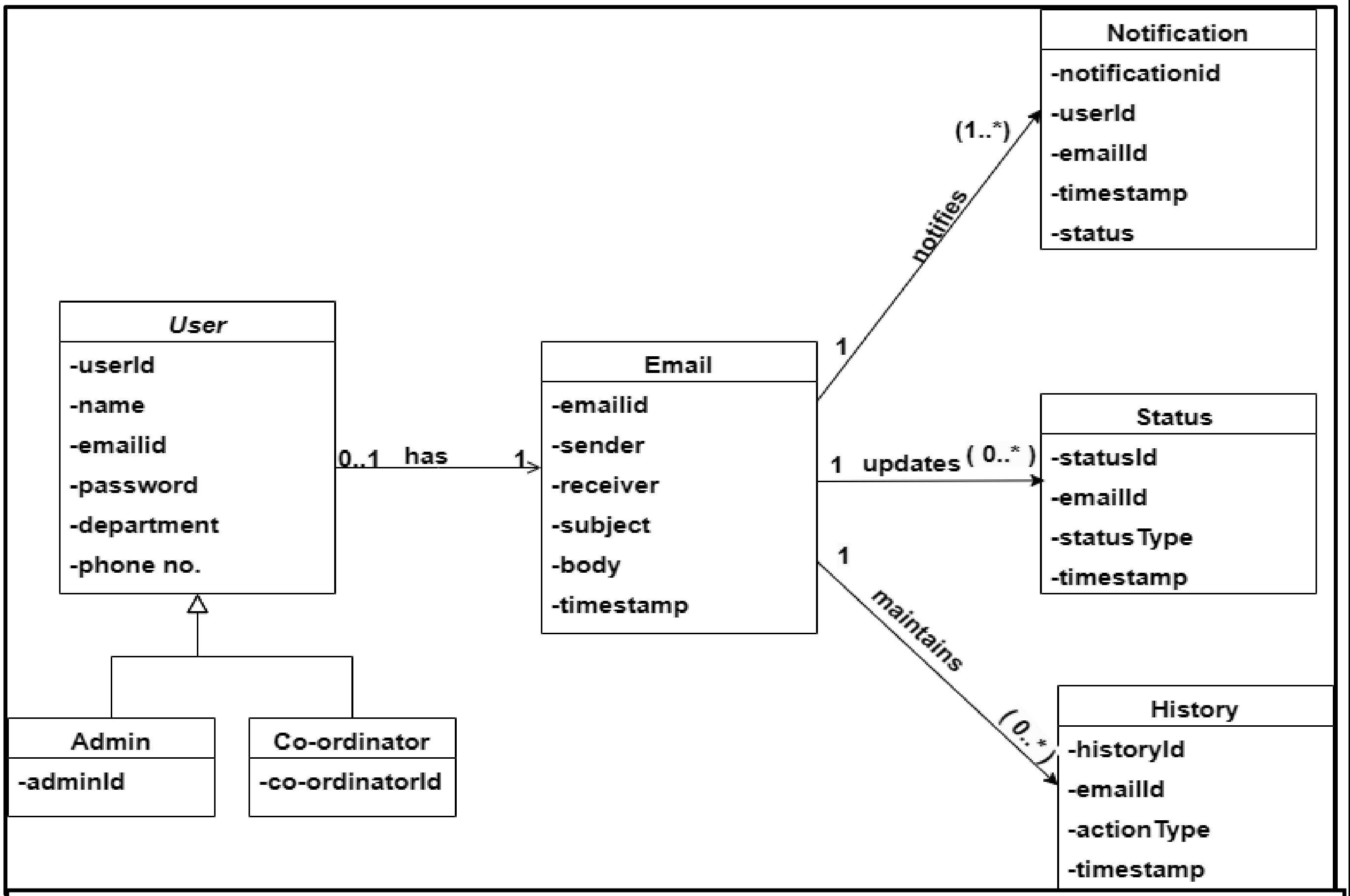


Fig.5.1.2 : UML Diagram : Class Diagram

5.2 Data Flow Diagrams (D.F.D.s):

The Data Flow Diagrams specify the flow of the system, here the processes are defined and the entities performed the processes. There are different levels of the DFD.

For example :

Level 0 DFD, Level 1 DFD, Level 2 DFD, etc

Level 1 DFD:

Processes:

1. Send Email:

User inputs email details and System updates Email Store

2. Receive Email:

External Email Server delivers email to System and System updates Email Store

3. Notify User:

System generates notification and System updates Notification Store

4. Track Email Status:

System tracks email status and System updates Status Store

5. Maintain Email History:

System logs email actions and System updates History Store

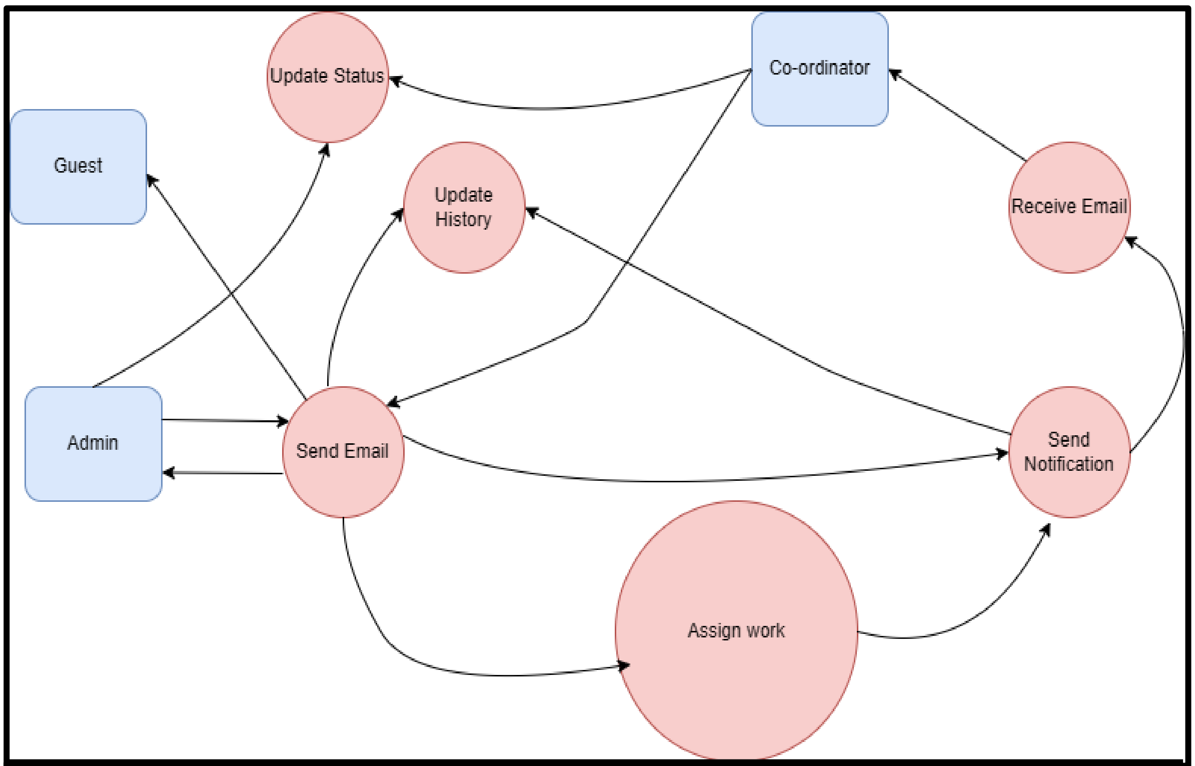


Fig.5.2 Data Flow Diagram (Level 0)

5.2 ER-Diagram:

ER (Entity-Relationship) Diagrams come after the initial UML diagrams. They focus on modelling the database structure, showing how data entities are related to each other within the system. ER diagrams help in defining and organizing the data requirements, which can then be translated into a physical database as

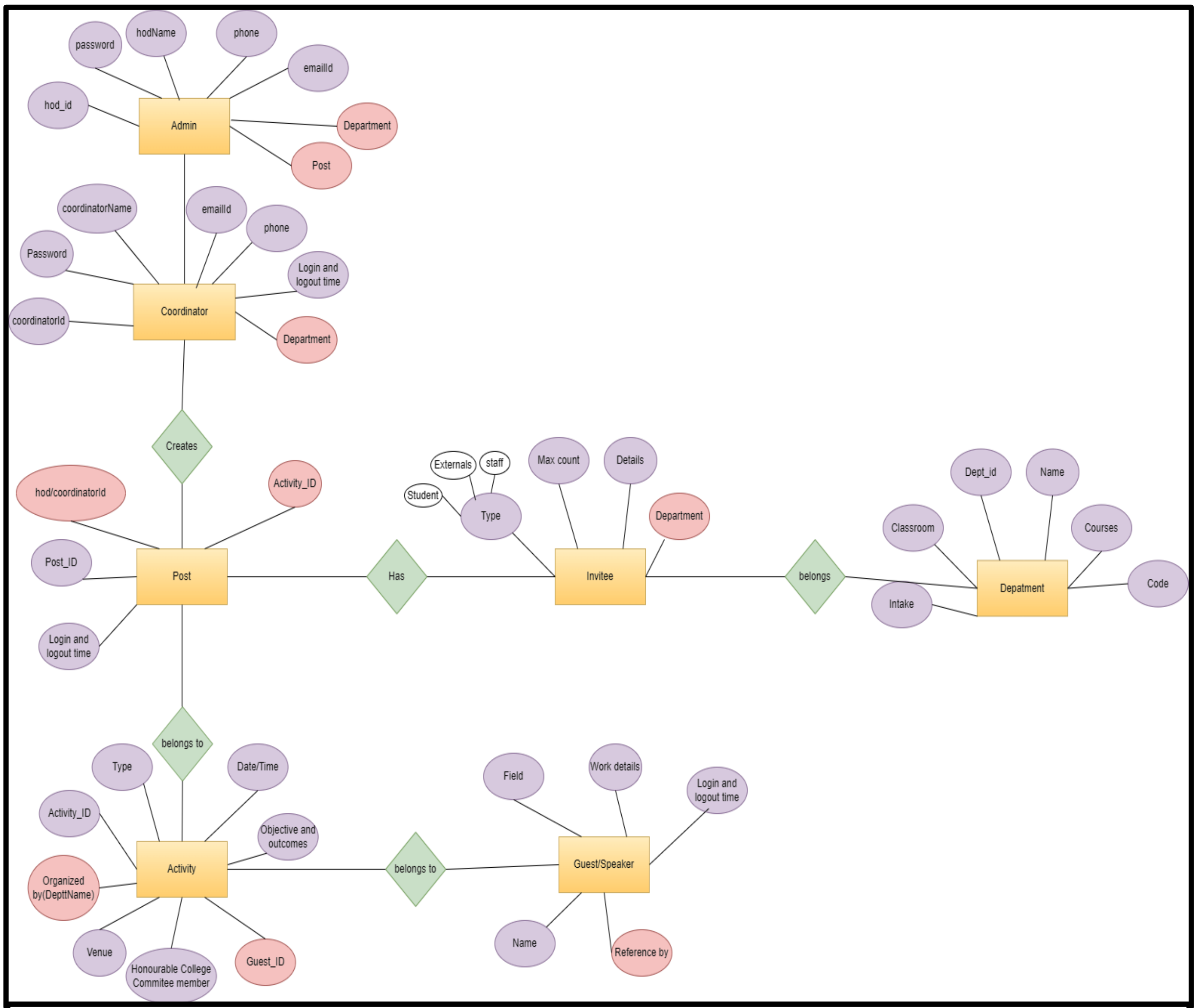


Fig.5.3 ER-Diagram

6.METHODOLOGY

Email Sync - Automated Email and Event Management System" was developed, detailing the materials, subjects, and equipment used, as well as the methods and procedures followed. By providing enough detail, this section ensures that the work can be replicated.

6.1 Software Used

- **IDE: IntelliJ IDE**
  IntelliJ IDEA was chosen as the primary Integrated Development Environment (IDE) for Java development due to its advanced support for the Spring Boot framework, extensive debugging tools, and productivity features such as code suggestions, refactoring tools, and version control integration. Its plugin ecosystem also allows for further customization to fit development needs.

- **Database: MySQL**
  MySQL was used as the relational database to store critical data, including email metadata, user information, notifications, and history of email transactions. MySQL's structured query language and robust transaction support make it ideal for managing large volumes of data with high reliability and consistency.

- **Backend Framework: Spring Boot**
  Spring Boot served as the backbone of the backend development, providing a structured framework that simplifies the creation of RESTful APIs, security configuration, dependency injection, and seamless database integration through Spring Data JPA. Spring Boot's ecosystem enables easy scaling and maintenance of the application.

- **Frontend Technologies: HTML, CSS, JavaScript**
  The user interface was created using HTML, CSS, and JavaScript. HTML structures the content, CSS provides styling to create an intuitive and visually appealing user experience, and JavaScript enables dynamic interactions, improving the responsiveness and usability of the web application.

- **APIs: RESTful APIs for Email Integration**
  RESTful APIs were utilized to integrate external email servers, allowing the system to manage incoming and outgoing emails seamlessly. These APIs handle various tasks, such as fetching new emails, sending notifications, tracking email status updates, and retrieving email history data, ensuring smooth communication between the backend and the email servers.

- **Version Control: Git**
  Git was used for source code management to track changes, manage versions, and facilitate collaboration. With Git, the development team could maintain a robust history of code changes, manage branches, and efficiently merge updates.

6.2 Hardware Specification

- **Development Machines**
  Development was carried out on Intel Core i5 machines equipped with 16GB of RAM

and 512GB SSD storage. These specifications ensure smooth performance during development, particularly for handling multiple processes in Java and running a MySQL database instance locally.

- **Servers: AWS EC2 Instances**
  For deployment, the backend services may be hosted on AWS EC2 instances configured with similar specifications. AWS EC2 provides scalable computing resources and is well-suited for backend applications that require stability, scalability, and availability.

- **Networking**
  High-speed internet was required to support efficient cloud deployment and stable API integration with external email servers. This ensured quick response times during development and seamless interaction with cloud services for deployment.

6.3 Programming Languages

- **Java**
  Java was selected as the primary programming language for backend development. Known for its portability, scalability, and compatibility with Spring Boot, Java provides robust support for managing server-side logic, handling HTTP requests, and ensuring data security.

- **HTML, CSS, JavaScript**
  HTML, CSS, and JavaScript were used for frontend development to create the web interface. These technologies enable the construction of a responsive, interactive UI, enhancing user experience and making the system accessible across devices.

6.4 Platform

- **Operating System: Ubuntu 20.04 LTS / Windows**
  Development was conducted on both Ubuntu 20.04 LTS and Windows operating systems. Ubuntu’s stability and compatibility with Java make it ideal for backend development and cloud deployment, while Windows provides ease of use and supports Java development with IntelliJ IDEA.

- **Cloud Platforms: AWS and Azure**
  AWS and Azure were chosen for their scalability, reliability, and security. AWS EC2 was primarily used for backend deployment, while other cloud services, such as AWS S3 or Azure Blob Storage, could be considered for storing email attachments or other static files, ensuring efficient data management.

6.5 Components

- **Database: MySQL**
  MySQL stores critical data, including email records, notifications, user preferences, and the history of email transactions. Its relational structure supports complex queries, allowing for efficient data retrieval and data integrity.

- **Backend: Spring Boot**
  Spring Boot handles the backend logic and API management, supporting functionality

such as email notifications, event tracking, and status updates. Its powerful dependency management, along with features like Spring Security and Spring Data JPA, allows for a well-organized and secure backend.

- **Frontend: HTML, CSS, JavaScript**
  The frontend component provides users with a responsive and interactive interface to view emails, receive notifications, track email statuses, and manage their settings. This component communicates with the backend through REST APIs to fetch and display real-time data.

- **APIs: RESTful APIs**
  RESTful APIs were developed to interact with external email servers, enabling the system to fetch, send, and update email information. The APIs facilitate seamless integration between the email servers and the backend, enhancing automation in email management.

6.6 Tools

- **Version Control: Git**
  Git provided robust version control for tracking code changes, managing branches, and supporting team collaboration. It ensured code integrity and allowed for efficient code review and merging processes, enabling team members to work on different features simultaneously.

- **Project Management: GitHub**
  GitHub was used for project management and task tracking. Its project boards, issue tracking, and pull request features helped streamline workflows, track progress, and document bug reports, ensuring efficient team coordination.

- **Testing Frameworks**

  - **JUnit:** JUnit was utilized for unit testing of backend components, ensuring each function in the backend performs as expected. Unit tests validate individual methods and services, minimizing the likelihood of errors in the core logic.
  - **Selenium:** Selenium was chosen for automated UI testing. Selenium scripts were created to test user flows, form submissions, and notifications, ensuring the interface performs well across different devices and screen sizes.

6.7 Coding Style Followed

- **Code Conventions**
  The project adheres to Java coding standards, which includes practices such as naming conventions (e.g., CamelCase for class names and methods, ALL_CAPS for constants), indentation, and spacing to ensure readability. Consistent use of standard practices helps avoid bugs, simplifies code review, and enhances overall code quality.

- **Documentation**
  Each class and method is thoroughly documented using Javadoc comments, which provide clear descriptions of functionalities, parameters, return values, and exceptions.

Inline comments are also used to explain complex logic within the code, making it easier for future developers or collaborators to understand and maintain.

- **Modular Design**
  The code is organized into distinct, reusable modules, ensuring separation of concerns and allowing for easier maintenance. Each feature is encapsulated in its own module or class, following the Single Responsibility Principle (SRP) of software design. This modularity enables independent development, testing, and scaling of each component.

6.8 Methods and Procedures

The project followed a structured development lifecycle, broken down into the following stages:

1. Requirement Analysis

- **Stakeholder Interviews**
  Interviews were conducted with potential users and stakeholders to gather insights into their needs and expectations. This helped define core features, such as email notifications, tracking status updates, and a history of email interactions.
- **Requirements Documentation**
  Both functional and non-functional requirements were documented. Functional requirements focused on specific features like email notifications, tracking, and database integration. Non-functional requirements included system performance, security, and usability standards.

2. Design Phase

- **UML Diagrams**
  Unified Modelling Language (UML) diagrams, such as class diagrams(Fig 5.1.2) and sequence diagrams, were created to model the system's architecture. These diagrams provided a visual representation of the interactions between components and helped identify dependencies.
- **Entity-Relationship Diagram (ERD) and Database Normalization**
  An ERD (Fig 5.3 :ER Diagram) was developed to define the structure of the database. Normalization was applied to the database to the Third Normal Form (3NF), reducing redundancy and ensuring data integrity. Tables were organized to store emails, user information, notifications, and history records in a relational structure.

3. Development Phase

A. Backend Development

The backend was built with Java, Spring Boot, and Hibernate to implement the following functionalities:

- **Email Notification System**
  A notification service was implemented to alert users when a new email is received. This system periodically checks the connected email servers using RESTful APIs and triggers notifications via the backend to update the frontend in real time.

- **Email Status Tracking**
  Each email's status—such as accepted, rejected, or pending—is tracked and stored in the database. This information is displayed on the frontend, enabling users to see the status of each email at a glance.
- **Email Record Management**
  All sent and received emails are stored in the database with metadata (e.g., sender, recipient, timestamp). The system ensures data consistency and integrity by enforcing transactions, allowing for a reliable history of interactions.

B. Frontend Development

The frontend was developed using HTML, CSS, and JavaScript to provide an intuitive and responsive user experience. Key functionalities include:

- **User Interface for Email Management**
  A dashboard allows users to view, send, and manage their emails. This interface communicates with the backend via AJAX or Fetch API requests, ensuring seamless updates.
- **Notification Alerts for New Emails**
  Real-time notifications are displayed on the user's dashboard to inform them of new emails. This feature enhances user experience by providing immediate feedback.
- **Status Updates and History Tracking**
  Users can view the status of their emails (e.g., accepted or rejected) and access a complete history of email interactions. This functionality is essential for tracking communications and maintaining records.

Example UI Components:

- **Email Dashboard**: Lists received emails with actions such as view, accept, reject, or delete.
- **Notification Panel**: Displays real-time notifications of incoming emails.
- **Email History**: Provides a detailed history of all email interactions, allowing users to track previous communications.

C. Integration

- **APIs**
  Integration with external email servers (e.g., SMTP and IMAP protocols) was achieved through RESTful APIs. These APIs enable the system to retrieve new emails, update statuses, and manage the flow of messages between the system and the email server.
- **Cloud Deployment**
  The system is deployed on cloud platforms like AWS (using services such as AWS EC2 and AWS RDS) or Azure to ensure reliability, scalability, and security. The use of Docker containers simplifies deployment and provides a consistent environment across different platforms.

4. Testing Phase

- **Unit Testing (JUnit)**
  Unit tests were created for backend functions to ensure that each method works correctly. JUnit was used to validate individual modules, focusing on core functionalities like email notification and status tracking.
- **Integration Testing**
  Integration testing was performed to ensure that all components work together seamlessly. This testing verified that the frontend and backend communicate effectively and that API integrations with external email servers' function correctly.
- **Automated UI Testing (Selenium)**
  Selenium was used to automate testing of the user interface. Tests were conducted for key workflows, such as viewing and managing emails, receiving notifications, and tracking statuses. Automated tests ensure that the frontend operates as expected across different devices and browsers.

5. Deployment Phase

- **Backend Deployment on AWS EC2**
  The backend was deployed on AWS EC2 instances, providing a scalable environment. Security configurations, such as firewalls and IAM roles, were applied to protect the application.
- **Database Configuration on AWS RDS**
  The MySQL database was hosted on AWS RDS, which offers high availability and automated backups, ensuring data reliability.
- **Frontend Deployment**
  The frontend was hosted on a cloud-based web server, allowing users to access the application through a browser. The frontend is configured to communicate with the backend securely over HTTPS.
- **Security Configurations**
  Access control, encryption, and other security measures were implemented to protect data in transit and at rest, ensuring compliance with best practices.

6. Monitoring and Maintenance

- **Monitoring Tools**
  Monitoring tools were implemented to track system performance, detect potential issues, and identify bottlenecks. AWS CloudWatch or similar services may be used to monitor server performance metrics and database health.
- **Feedback Loop for Continuous Improvement**
  A feedback loop was established to collect user feedback and bug reports. This information is used for continuous improvement, ensuring the application evolves to meet user needs.

7. RESULT

Result refers the section in which present the findings and outcomes of the work conducted. This section provides evidence to support whether the project objectives and goals were achieved. It typically includes data, analysis, and interpretations.

7.1 Functional Outcomes

Email Notification System:

- Users receive real-time notifications for new emails.
- Notifications are customizable based on user preferences (e.g., email, SMS, push notifications).
- The system accurately tracks the status of emails (accepted, rejected, pending) and updates them in real-time.

Email Management:

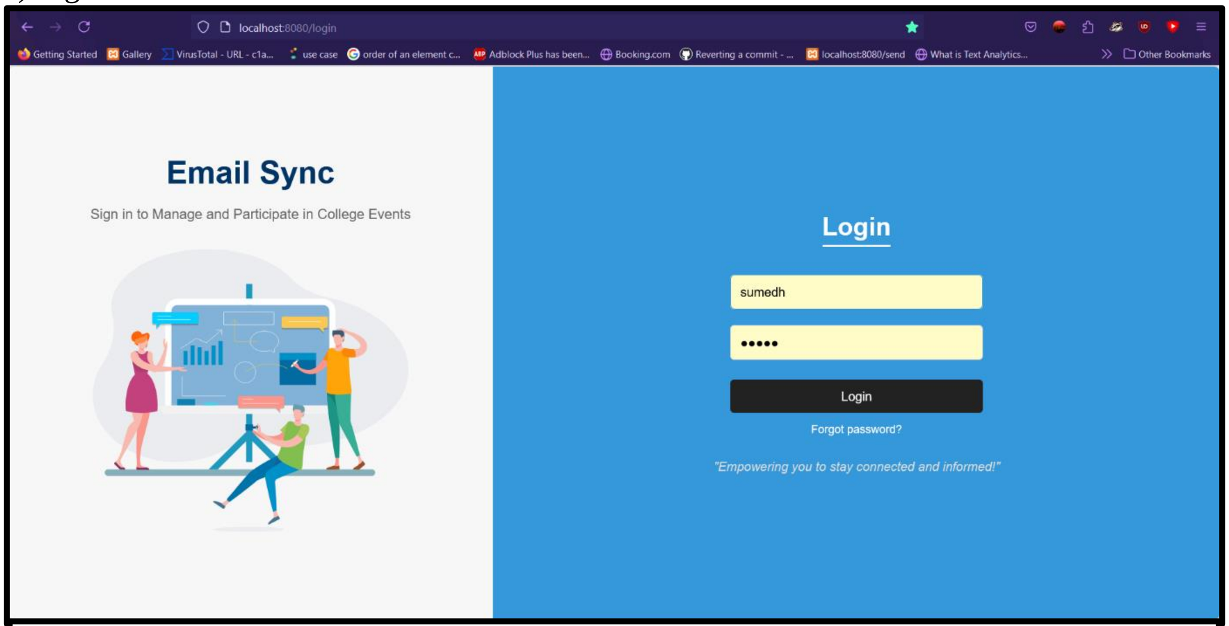
- Comprehensive records of all sent and received emails are maintained.
- Users can search and retrieve historical emails efficiently.
- The system integrates seamlessly with external email servers, ensuring reliable email sending and receiving.

Performance:

- The system handles large volumes of emails without performance degradation.
- Real-time updates and notifications are processed efficiently.

Final Output is as follows:

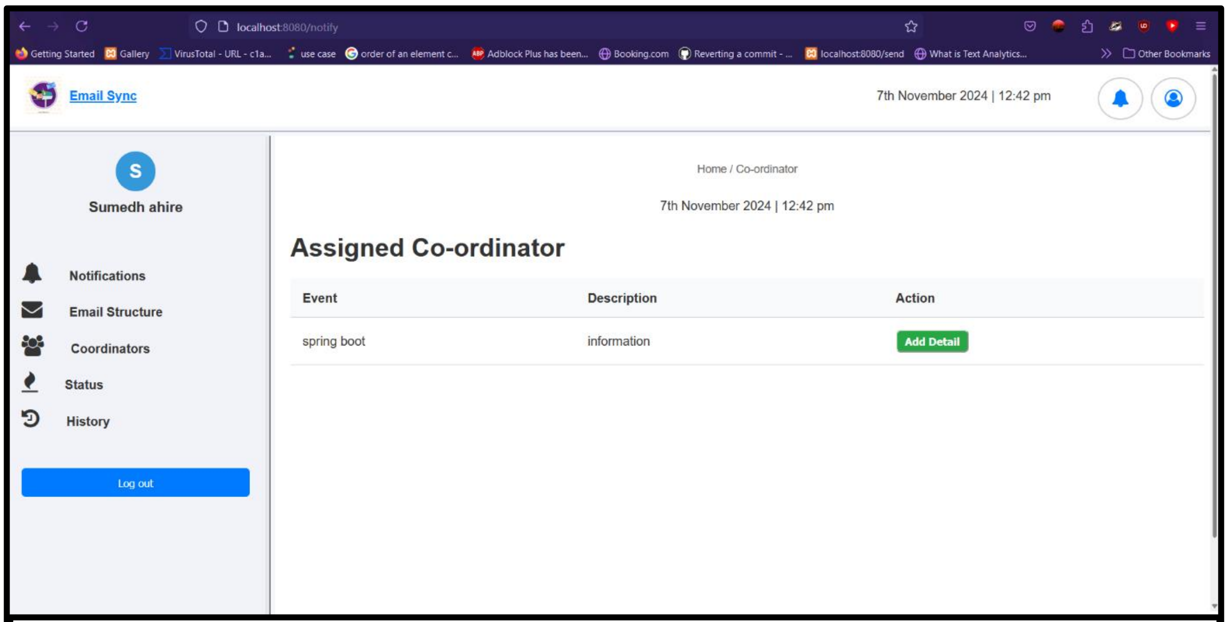
1)Login



localhost:8080/login
Email Sync
Sign in to Manage and Participate in College Events
Login
sumedh
•••••
Login
Forgot password?
"Empowering you to stay connected and informed!"

Fig 7.1: Login page

2)Event is assigned to a coordinator by admin via dashboard ,this is notified to the respective coordinator


localhost:8080/notify

Email Sync

7th November 2024 | 12:42 pm

Sumedh ahire

Home / Co-ordinator

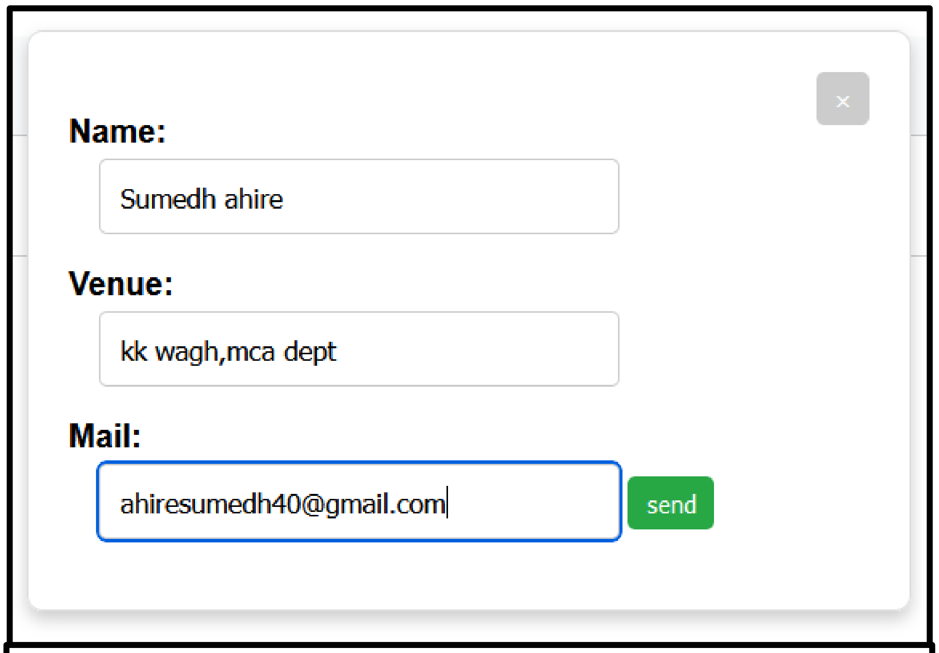
7th November 2024 | 12:42 pm

Assigned Co-ordinator

Event	Description	Action
spring boot	information	Add Detail

Fig 7.2 Notification page

3)Structuring the event


Name:

Sumedh ahire

Venue:

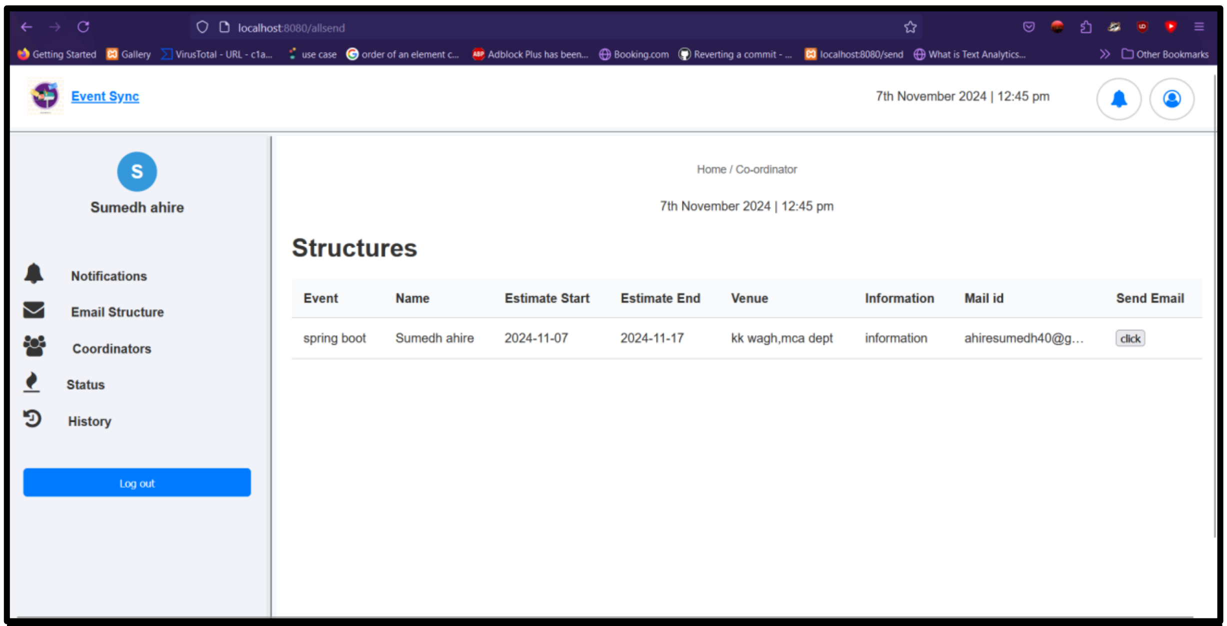
kk wagh,mca dept

Mail:

ahiresumedh40@gmail.com

Fig 7.3 Setting structure for the respective event

4)Structure table record

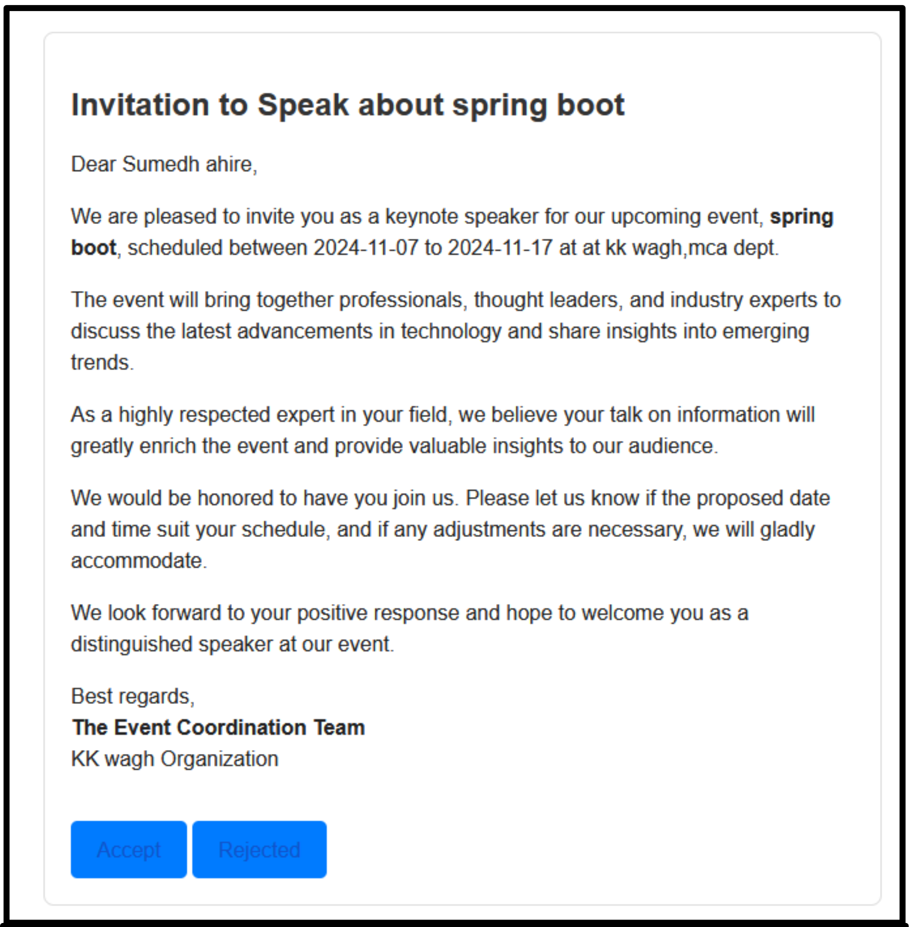


Structures

Event	Name	Estimate Start	Estimate End	Venue	Information	Mail id	Send Email
spring boot	Sumedh ahire	2024-11-07	2024-11-17	kk wagh,mca dept	information	ahiresumedh40@g...	click

Fig 7.4 Structure records

5)Sending the mail by clicking the send email button, the mail is as follow



Invitation to Speak about spring boot

Dear Sumedh ahire,

We are pleased to invite you as a keynote speaker for our upcoming event, **spring boot**, scheduled between 2024-11-07 to 2024-11-17 at at kk wagh,mca dept.

The event will bring together professionals, thought leaders, and industry experts to discuss the latest advancements in technology and share insights into emerging trends.

As a highly respected expert in your field, we believe your talk on information will greatly enrich the event and provide valuable insights to our audience.

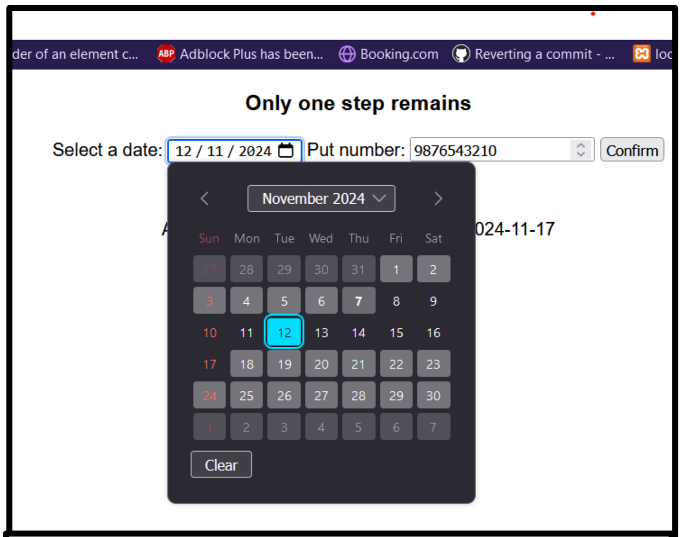
We would be honored to have you join us. Please let us know if the proposed date and time suit your schedule, and if any adjustments are necessary, we will gladly accommodate.

We look forward to your positive response and hope to welcome you as a distinguished speaker at our event.

Best regards,
The Event Coordination Team
KK wagh Organization

Fig 7.5 Mail to the expert with options

6)When expert accepts the offer ,will be redirected to the acceptance page


der of an element c... Adblock Plus has been... Booking.com Reverting a commit - ... loc
November 2024
Sun Mon Tue Wed Thu Fri Sat
Clear

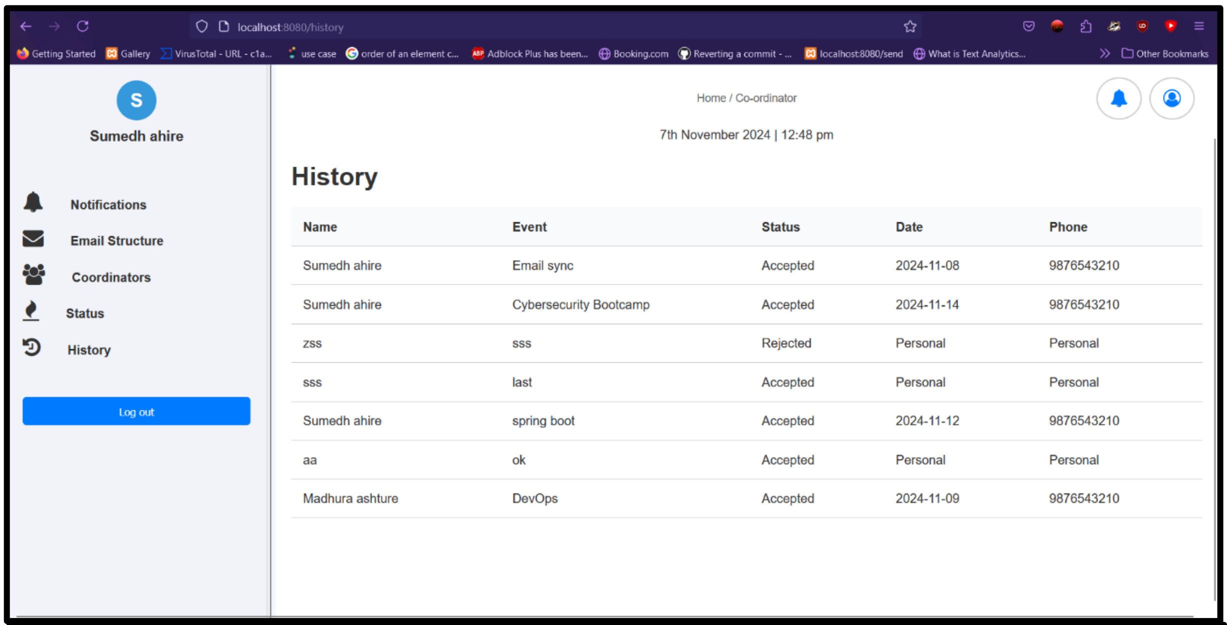
Only one step remains

Select a date: 12 / 11 / 2024 Put number: 9876543210 Confirm

024-11-17

Fig 7.6 Acceptance page for expert

7)History records


localhost:8080/history
Getting Started Gallery VirusTotal - URL - c1a... use case order of an element c... Adblock Plus has been... Booking.com Reverting a commit - ... localhost:8080/send What is Text Analytics... Other Bookmarks
S
Sumedh ahire
Notifications
Email Structure
Coordinators
Status
History
Log out
Home / Co-ordinator

7th November 2024 | 12:48 pm

History

Name	Event	Status	Date	Phone
Sumedh ahire	Email sync	Accepted	2024-11-08	9876543210
Sumedh ahire	Cybersecurity Bootcamp	Accepted	2024-11-14	9876543210
zss	sss	Rejected	Personal	Personal
sss	last	Accepted	Personal	Personal
Sumedh ahire	spring boot	Accepted	2024-11-12	9876543210
aa	ok	Accepted	Personal	Personal
Madhura ashture	DevOps	Accepted	2024-11-09	9876543210

Fig 7.7 Record of history

7.2 User Experience

User Interface:

- The dashboard is user-friendly and intuitive, allowing users to manage emails and events easily.
- Responsive design ensures accessibility across various devices, including desktops, tablets, and smartphones.

Usability:

- Users can customize notification settings to suit their preferences.
- Real-time status tracking provides transparency and keeps users informed about their email interactions.

Feedback and Iteration:

- User feedback has been positive, highlighting the system's efficiency and ease of use.
- Continuous updates based on user feedback ensure the system remains relevant and effective.

7.3 Technical Results

Backend Performance:

- The backend, developed using Java and Spring Boot, is robust and scalable.
- Hibernate ensures efficient database management, maintaining data integrity and consistency.
- Integration with external email servers via APIs works seamlessly.

Frontend Performance:

- The frontend, built with HTML, CSS, and JavaScript, delivers a responsive and smooth user experience.
- Real-time notifications and updates are handled effectively without causing lag.

Security:

- Data encryption and access controls protect user data.
- The system complies with data privacy regulations, ensuring user trust and safety.

7.4 Overall Impact:

The "Email Sync - Automated Email and Event Management System" project has had a profound impact on both personal and professional productivity by addressing key challenges in managing emails and events.

Improved Organization

By maintaining comprehensive records of all email interactions and providing real-time updates, the system ensures that users are better organized. This organization extends to managing events and notifications, making it easier for users to stay on top of their schedules and communications.

User Satisfaction

The user-friendly interface and customizable notification settings have received positive feedback from users, who appreciate the seamless experience and the ability to tailor the system to their preferences. This satisfaction leads to higher engagement and more effective use of the system.

Scalability and Reliability

The deployment on cloud platforms like AWS ensures that the system is scalable and reliable, capable of handling large volumes of emails without performance issues. This scalability means that the system can grow with its user base, continuing to provide high-quality service as demand increases.

8.TESTING

Testing is a crucial phase in developing the "Email Sync - Automated Email and Event Management System." It ensures the system functions correctly, meets specified requirements, and performs efficiently. This phase includes technical reviews to validate design and code quality, and a comprehensive test plan covering unit, integration, system, performance, and security testing. By thoroughly testing each component and the overall system, we ensure reliability, usability, and robustness before deployment.

8.1 Format Technical Reviews

Objective: Technical reviews aim to ensure that the design and code meet the required standards, are free from defects, and adhere to best practices.

Participants:

- Project Manager
- Developers
- QA/Testers
- Business Analyst

Review Process:

1. **Preparation:**
    - Reviewers receive documentation and code ahead of the meeting.
    - Each reviewer examines the materials independently.
2. **Review Meeting:**
    - **Introduction:**
        - Brief overview of the objective and scope of the review.
    - **Walkthrough:**
        - Developers walk through the design/code, explaining the logic and implementation.
    - **Discussion:**
        - Reviewers discuss any issues or concerns.
    - **Action Items:**
        - Identify and assign action items to address any defects or improvements.
3. **Documentation:**
    - Document the findings, discussions, and action items.
    - Store the documentation in the project management tool for reference.

Frequency:

- Conduct technical reviews at major milestones, such as the completion of key modules or phases.

8.2 Test Plan

The test plan aims to outline the testing strategy, objectives, scope, resources, schedule, and deliverables to ensure the system functions correctly and meets user requirements.

Test Strategy:

Unit Testing:

In the Unit Testing , it tests the individual components for correctness.

- Each function or method is tested in isolation to ensure it performs as expected.
- Mock objects and stubs are used to simulate interactions with other components.
- Tools like JUnit and Mockito are commonly used.
- Example: Testing the email sending function to ensure it correctly formats and sends emails.

Integration Testing:

Integration testing integrates components work together as expected.

- Tests are conducted to ensure that different modules or services interact correctly.
- Focuses on the interfaces and data flow between components.
- Example: Testing the interaction between the email module and the notification module to ensure notifications are sent when an email is received.

System Testing:

System testing the entire system to ensure it meets the specified requirements.

- End-to-end testing of the complete system in an environment that closely resembles the production environment.
- Ensures that the system as a whole function correctly and meets the requirements.
- Example: Testing the entire email sync process from composing an email to receiving notifications and tracking status.

Performance Testing:

The software performance testing ensures the system performs well under various conditions.

- Tests are conducted to evaluate the system's performance under different loads and stress conditions.
- Includes load testing, stress testing, and scalability testing.
- Example: Testing how the system handles a high volume of emails being sent and received simultaneously.

Security Testing:

Software security testing identifies the vulnerabilities and ensure data protection.

- Tests are conducted to identify potential security vulnerabilities and ensure that data is protected.
- Includes penetration testing, vulnerability scanning, and security code reviews.
- Example: Testing for SQL injection vulnerabilities in the email storage module.

Test Schedule

A detailed test schedule ensures that all testing activities are planned and executed in a timely manner, allowing for thorough validation of the system before deployment.

Week 1-2: Preparation

- Set up Test Environment: Configure hardware, software, and network settings.
- Create Test Data: Generate sample data for users, emails, notifications, statuses, and history.
- Develop Test Cases: Write detailed test cases for unit, integration, system, performance, and security testing.

Week 3-4: Unit Testing

- Execute Unit Tests: Test individual components using JUnit and Mockito.
- Fix Defects: Address any issues identified during unit testing.
- Review Code Coverage: Ensure high code coverage and refactor code as needed.

Week 5-6: Integration Testing

- Execute Integration Tests: Test interactions between different modules.
- Verify Data Flow: Ensure data flows correctly between components.
- Fix Defects: Address any issues identified during integration testing.

Week 7-8: System Testing

- Execute System Tests: Conduct end-to-end testing of the entire system.
- Validate Requirements: Ensure the system meets all functional and non-functional requirements.
- Fix Defects: Address any issues identified during system testing.

Week 9: Performance Testing

- Execute Performance Tests: Test the system's performance under various conditions using LoadRunner.
- Analyse Results: Identify any performance bottlenecks and optimize the system.
- Fix Defects: Address any performance-related issues.

Week 10: Security Testing

- Execute Security Tests: Conduct security testing using Burp Suite.

- Identify Vulnerabilities: Find and document any security vulnerabilities.
- Fix Defects: Implement security measures to address identified vulnerabilities.

Week 11: User Acceptance Testing (UAT)

- Engage Users: Involve users in testing the system.
- Collect Feedback: Gather feedback from users and make necessary adjustments.
- Fix Defects: Address any issues identified during UAT.

Week 12: Final Review and Deployment Preparation

- Review Test Results: Conduct a final review of all test results.
- Prepare Deployment Plan: Develop a detailed plan for deploying the system to production.
- Final Fixes: Address any remaining issues and ensure the system is ready for deployment.

Test Environment

A well-defined test environment is crucial for ensuring that the "Email Sync - Automated Email and Event Management System" is tested under conditions that closely resemble the production environment. This helps in identifying and resolving issues before deployment.

1. Hardware

- Development Machines: Intel Core i7, 16GB RAM, 512GB SSD
- Test Servers: AWS EC2 instances with similar specifications to production servers
- Networking: High-speed internet connection to support cloud deployment and API integration

2. Software

- Operating System: Ubuntu 20.04 LTS for both development and testing environments
- Database: MySQL for data storage and management
- Backend Framework: Spring Boot for creating a robust and scalable backend
- Frontend Technologies: HTML, CSS, JavaScript for developing the user interface
- APIs: Integration with external email servers using RESTful APIs
- Version Control: Git for source code management
- Testing Tools:
  - JUnit: For unit testing
  - Mockito: For creating mock objects
  - Selenium: For automated UI testing
  - LoadRunner: For performance testing
  - Burp Suite: For security testing

3. Test Data

- Sample User Accounts: Multiple user accounts with different roles and permissions
- Email Data: Sample emails with various attributes (sender, receiver, subject, body, timestamp)
- Notification Data: Sample notifications with different statuses and timestamps
- Status Data: Sample status updates for emails

- History Data: Sample historical records of email interactions

4. Configuration

- Environment Setup:
  - Development environment for initial testing
  - Staging environment that mirrors the production environment for final testing
- Database Configuration:
  - Separate databases for development, testing, and production
  - Regular backups and data refreshes to ensure consistency
- API Configuration:
  - Secure API endpoints for communication with external email servers
  - OAuth 2.0 for authentication

Test Cases:

A test case is a specific set of conditions or variables under which a tester determines whether a system or component is working correctly. It includes inputs, execution conditions, testing procedures, and expected results.

1.Email Notification System:

Verify that users receive real-time notifications for new emails.

Test different notification settings (e.g., email, SMS, push notifications)

Table 8.1: Email Notification System Testcase

Test Id	**Test Case Description**	**Testcase Input**	**Actual O/p**	**Expected Output**	**Status**
101	Send a new email to the user.	Valid email Id	Send email	Send Email	Pass
102	Send email to existing user	Valid Email id	Send Email	Send Email	Pass
103	Send email to invalid email Id	Invalid email id	Email Not sent	Email Not Sent	Pass
104	Send email through unregistered admin email Id	Unregistered Email Id of admin	Email Sent	Email Not Sent	Fail

As shown in Table 8.1 , the test case table provides an organised view of Email Notification System Testcase used to verify this feature of Email Notification. It includes the test case id , test case description , expected output , expected input , status.

2.**Email Status Tracking:**

Verify that email statuses (accepted, rejected, pending) update correctly.

Test status updates under various conditions (e.g., network latency).

Table 8.2 Email Status Tracking Testcase

Test Id	**Test Case Description**	**Testcase Input**	**Actual O/p**	**Expected Output**	**Status**
201	Update Email Status to "Sent"	Valid email	ok	ok	Pass
202	Update Email Status to "Delivered"	Sent email	Delivered	Delivered	Pass
203	Update Email Status to "Pending"	Email sent	Email Not sent	Email Not Sent	Pass

As shown in Table 8.2 , the test case table provides an organised view of Email Status Tracking Testcase used to verify this feature of Email Status Tracking of accept reject or pending. It includes the test case id , test case description , expected output , expected input , status.

Resources:

- **Testers:** QA/Testers
- **Environment:** Development and testing environments set up to mirror the production environment.
- **Tools:** JUnit, Selenium, LoadRunner, Burp Suite

Schedule:

- **Preparation Phase:**
    - Develop test cases and scripts.
    - Set up the testing environment.
- **Execution Phase:**
    - Execute test cases according to the test plan.
    - Record results and log defects.
- **Reporting Phase:**
    - Generate test reports.
    - Review findings and address any issues.

Deliverables:

- Test Plan Document
- Test Cases and Scripts , Test Reports
- Defect Logs , Test Summary Report

CONCLUSION

The results of the “Email Sync - Automated Email and Event Management System” project have demonstrated its effectiveness in improving email, streamline email and event management through automation and event management. The system has enhanced user productivity, provided comprehensive record-keeping, and ensured data security and privacy. The positive user feedback and performance metrics indicate that the system is well-suited for both personal and professional use.

The preliminary success of the system opens up opportunities for future enhancements, such as integrating with calendar systems for event management, developing advanced analytics for email interactions, and creating a mobile application for on-the-go management. These potential improvements will further solidify the system's value and utility.

In conclusion, the "Email Sync - Automated Email and Event Management System" project has delivered a comprehensive and efficient solution for managing emails and events, providing significant benefits in terms of productivity, organization, user satisfaction, scalability, and security. The positive impact on users' daily operations underscores the value of investing in automated, user-centric solutions.

APPENDICES

Appendices provide supplementary material that supports the main text of your project report. This can include detailed data, technical documentation, additional diagrams, code snippets, and any other relevant information.

Appendix A: Detailed Database Schema

- Comprehensive tables, attributes, primary and foreign keys, and relationships.

Appendix B: Full UML Diagrams

- Extended versions of the Use Case, Class, Sequence, and Activity diagrams.

Appendix C: ERD and Normalization

- Entity-Relationship Diagram with detailed annotations.
- Steps and rationale for normalization (1NF, 2NF, 3NF).

Appendix D: Full Test Cases and Results

- Complete list of test cases, including descriptions, steps, expected outcomes, and actual results.
- Detailed test reports for unit, integration, and system testing

Appendix E: Code Snippets

- Key portions of the source code, including explanations.
- Examples of critical functions and methods.
-

Appendix F: Project Management Documents

- Copies of the project plan, risk assessment, and Gantt charts.
- Records of technical reviews and meeting minutes.
-

Appendix G: User Manuals

- Instructions for end-users on how to use the system.
- Troubleshooting guide.

ACKNOWLEDGEMENT

We would like to express our heartfelt gratitude to all those who have supported and guided us throughout the development of the "Email Sync - Automated Email and Event Management System" project.

First and foremost, we would like to thank our project guide, **Prof. Archana Rane** for her invaluable guidance, insightful suggestions, and continuous encouragement. Their expertise and dedication were instrumental in the successful completion of this project.

We are also grateful to our professors and lecturers **K. K. Wagh Institute of Engineering Education and Research, Nashik** for providing us with the foundational knowledge and skills necessary for undertaking this project. Their teachings have greatly contributed to our understanding and application of advanced technologies.

We would like to extend our sincere thanks to our project team members for their hard work, collaboration, and unwavering support. Their contributions, whether through brainstorming sessions, coding, or testing, were crucial to bringing this project to fruition.

Special thanks go to the technical staff at MCA Department of K.K.W.I.E.E.R for their assistance with the hardware and software resources needed for this project. Their support in setting up the development and testing environments was invaluable.

We are also thankful to the users who provided feedback during the testing phase. Their insights and suggestions were pivotal in refining the system and ensuring it meets user needs effectively.

Lastly, we would like to acknowledge the unwavering support of our family and friends. Their encouragement and belief in our abilities kept us motivated throughout this journey.

Thank you all for your contributions and support, which have made this project a success.

Sumedh Ahire

Madhura Ashture

Vedant Bhosale

Pratik Kankarej

Report Documentation					
Report Code: MCA-SY Project 2023-2025				**Report Number: B-01**	
Report Title: **E-mail Sync: Automated Email and Event Management System**					
Address (Details): Department of MCA, K. K. Wagh Institute of Engineering Education and Research, Hirabai Haridas Vidyanagari, Amrutdham, Nashik Pin – 422 003, M.S. INDIA.					
Sumedh Ahire **Address :** Jail Road Nashik **E-mail :** ahiresumedh40@gmail.com **Roll:** 03 **Cell No:-** 9766338826	**Madhura Ashture** **Address:** Upnagar,Nashik **E-mail :** madhuraashture@gmail.com **Roll:** 04 **Cell No:**9370728305	**Vedant Bhosale** **Address:** Nashik Road,Nashik **E-mail:** bvedantuiux@gmail.com **Roll:** 09 **Cell No:-** 8459529977	**Pratik Kankarej** **Address :-** Sinnar **E-mail:** pratikkankarej18@gmail.com **Roll:** 33 **Cell No:-**8805579968		
Year: 2023 – 2025 **Branch: Master of Computer Application (MCA)**					
***Key Words*:** Automation, Email Management, Event Management, Java, Real-time Notifications, Spring Boot, User Productivity, Email history, Status.					
Type of Report: FINAL	*Report Checked By:*	Report Checked Date:	Guides Complete Name: **Archana L. Rane**	Total Copies	

Abstract:

In today's fast-paced digital world, effective email and event management is crucial for both personal and professional productivity. The "Email Sync - Automated Email and Event Management System" project aims to address this need by developing a robust, automated system that streamlines the management of emails and events. This system is designed to notify users of new emails, track the status of emails (accepted or rejected), and maintain a comprehensive record of all sent and received emails. By leveraging advanced technologies and a well-structured database, the project seeks to enhance user efficiency and organization.