CodeMania

“WEB BASED MAIL CLIENT MODEL”

Software Requirements Specification

Version <1.0>

Submitted in Partial Fulfillment for the Award of Degree of Bachelor of Technology in Information Technology from Rajasthan Technical University, Kota

MENTOR:

Mrs. Astha Joshi

(Dept. of Information Technology)

COORDINATOR:

Dr. Priyanka Yadav

(Dept. of Information Technology)

SUBMITTED BY:

ABHYUDAYA SARASWAT
21ESKIT004

ANUSHKA SHARMA
21ESKIT018

DEPARTMENT OF INFORMATION TECHNOLOGY

SWAMI KESHWANAND INSTITUTE OF TECHNOLOGY, MANAGEMENT & GRAMOTHAN

Ramnagaria (Jagatpura), Jaipur – 302017

2024-2025

Contents

1. Introduction

1.1 Purpose

The purpose of the Web-Based Mail Client Model project is to develop a web-based email application that allows users to send, receive, store, delete, and manage emails via a user-friendly interface. This system is intended to simplify the email communication process, eliminating the need for dedicated desktop email clients by providing accessibility through web browsers. It aims to offer functionalities such as user authentication, email composition, inbox management, contact management, and user settings.

1.2 Scope

The Web Based Mail Client Model will be a web-based system designed to simulate core email system functionalities. This will include:

User Registration and Authentication: Secure user sign-up and login functionalities.
Inbox Management: Users can view emails, manage folders, and sort through messages.
Email Composition: Users can draft, send, and attach files to emails.
Email Retrieval: Users can retrieve messages from their inbox.
Contact Management: Users can manage a list of saved contacts.
User Preferences: Settings like password change, notifications, and themes.
File Attachments: Attachments for emails, with restrictions on file types or sizes.
The system will be built with a focus on ease of use, security, and scalability.

1.3 Definitions, Acronyms and Abbreviations

- **SRS**: Software Requirement Specification
- **UI**: User Interface
- **SMTP**: Simple Mail Transfer Protocol (used to send emails)
- **IMAP**: Internet Message Access Protocol (used to retrieve emails)
- **HTTP**: Hypertext Transfer Protocol

- **HTML**: HyperText Markup Language
- **CSS**: Cascading Style Sheets
- **JS**: JavaScript
- **API**: Application Programming Interface

1.4 References

- IEEE Standard for Software Requirements Specifications
- RFC 5321: SMTP Protocol Standard
- RFC 3501: IMAP Protocol Standard
- Web-based frameworks documentation (e.g., Django, Flask, React, Angular)
- Existing mail client system designs and research papers
- Documentation for database technologies like MySQL, PostgreSQL, or MongoDB

1.5 Technologies to be used

The system will use the following technologies and tools:

1. **Frontend Development**:
   - HTML, CSS, JavaScript
   - Frontend Frameworks like React.js or Angular
2. **Backend Development**:
   - Java with Springboot or Hibernate
3. **Database**:
   - MySQL
   - PostgreSQL
   - SQLite (for lightweight prototyping)

4. **Email Protocols Integration**:
    - SMTP for sending emails
    - IMAP for email retrieval
5. **Security Features**:
    - SSL/TLS encryption
    - OAuth 2.0 or custom token-based authentication
6. **Server Infrastructure**:
    - Apache/Tom Cat
    - Cloud Hosting (AWS, Azure, or Heroku)

1.6 Overview

This SRS document is structured systematically into distinct sections, each detailing specific facets of the "Web based mail client model" platform. It includes an executive summary, system overview, detailed descriptions of functional and non-functional requirements, user interfaces, system constraints, assumptions, dependencies, and technical specifications. Additionally, it incorporates diagrams,use cases, and mockups to provide a comprehensive understanding of the system's architecture and functionality.

2. Literature survey

1. Web-Based Communication Systems

Web-based communication systems have become integral to modern digital interaction, enabling seamless exchanges of information across devices. These systems leverage internet protocols like HTTP, SMTP, and IMAP to facilitate real-time communication and data sharing. Unlike traditional desktop applications, web-based systems offer platform independence, accessibility, and a centralized approach to data management. Examples such as Gmail, Slack, and Microsoft Teams illustrate how web-based systems combine ease of use with robust capabilities like multimedia support, multi-device synchronization, and integration with third-party tools. The rise of these systems emphasizes the importance of scalability, security, and a user-centric interface, all of which serve as guiding principles for the proposed mail client.

2. Mail Client Applications and their Applications

Mail client applications have evolved from basic desktop programs to sophisticated web-based platforms that offer unparalleled convenience and functionality. Early email clients like Microsoft Outlook and Eudora relied on local storage and offline accessibility, but their limitations in synchronization and cross-device usability posed challenges in an increasingly mobile world. With the advent of cloud technology and internet advancements, email services transitioned to web-based platforms such as Gmail and Yahoo Mail. These systems brought cloud storage, enhanced synchronization, and an intuitive user interface, providing users with accessibility from any device with internet connectivity.

3. Research Papers and Publications

Research papers and publications have significantly influenced the design principles and functionality of web-based mail clients. Studies on communication protocols like SMTP and IMAP highlight their pivotal role in ensuring reliable email delivery and retrieval while addressing challenges like latency and security vulnerabilities. Research on user authentication and data protection emphasizes techniques such as two-factor authentication and encryption to safeguard against unauthorized access and data breaches. User experience research has demonstrated the importance of responsive design and intuitive navigation in improving user satisfaction and efficiency. Additionally, the incorporation of artificial intelligence in email systems, as discussed in recent studies, has shown potential in areas such as predictive categorization, auto-completion, and spam detection. These insights contribute to the design of a comprehensive, intelligent, and secure mail client.

4. Conclusion

The literature survey highlights the transformative journey of web-based communication systems and mail client applications, alongside key findings from research publications. While current systems have achieved remarkable advancements in accessibility, security, and usability, there remain opportunities for innovation, particularly in enhancing automation, personalization, and security features. The *Web-Based Mail Client Model* aims to build upon these insights, creating a platform that meets modern user demands while addressing existing challenges. By integrating state-of-the-art technologies and best practices, this model aspires to deliver a robust, efficient, and user-friendly email solution.

3. Specific Requirements

3.1 Functional Requirements

The functional requirements of the *Web-Based Mail Client Model* define the core operations and features the system must perform:

1. **User Authentication**: Secure user login and registration using email and password with optional two-factor authentication (2FA).
2. **Inbox Management**: Display a list of received emails with options for sorting, searching, and categorizing messages.
3. **Email Composition**: Allow users to draft, send, and attach files to emails.
4. **Email Retrieval**: Support fetching and syncing emails in real-time using protocols such as IMAP.
5. **Contact Management**: Provide functionalities for adding, editing, and deleting contacts in a personalized contact list.
6. **Spam Filtering**: Detect and move spam emails to a separate folder using intelligent algorithms.
7. **User Settings**: Enable customization of themes, notification preferences, and account security settings.

3.2 Non-Functional Requirements

Non-functional requirements focus on the performance, scalability, and usability of the system:

1. **Performance**: The system must support up to 10,000 concurrent users with minimal latency.
2. **Scalability**: Ability to expand the system for additional features or user growth without performance degradation.
3. **Security**: Ensure data encryption (SSL/TLS) for all communications and storage.
4. **Usability**: Provide an intuitive and responsive user interface compatible with various devices and browsers.
5. **Availability**: Maintain 99.9% uptime, ensuring high availability for users.

.

3.3 Hardware Requirements

The hardware requirements specify the necessary infrastructure for hosting and accessing the system:

1. **Server-Side Requirements**:
   - Processor: Quad-core or higher
   - RAM: 16 GB or higher
   - Storage: SSD with at least 1 TB capacity
   - Network: High-speed internet connection (1 Gbps or more)
2. **Client-Side Requirements**:
   - Device: Desktop, laptop, or mobile device with internet access
   - Browser: Latest versions of Chrome, Firefox, Safari, or Edge
   - Minimum Screen Resolution: 1024x768

3.4 Software Requirements

The software requirements include:

- **Frontend:** HTML, CSS, JS with Angular
- **Database:** SQL, MongoDB
- **Authentication:** Firebase Authentication
- **Development Tools:** Git

3.5 Agile Methodology

The project will adopt an Agile development methodology to ensure iterative and incremental delivery. Key components include:

1. **Sprints**: Development cycles of 2-4 weeks to deliver functional increments.
2. **Daily Standups**: Short meetings to discuss progress, challenges, and goals.
3. **Backlog Management**: A prioritized list of features and fixes to guide the development process.
4. **Continuous Integration and Testing**: Regular integration of code changes with automated testing for quality assurance.

3.6 Business Process Model

The business process model for the *Web-Based Mail Client Model* includes the following:

1. **User Interaction**:
   - Users sign up, log in, and manage their emails via an intuitive web interface.
2. **Backend Processing**:
   - Server-side processes handle email communication, data storage, and protocol management.
3. **Data Flow**:
   - Emails are sent and received securely via SMTP and IMAP protocols, while user data is stored in a relational database.
4. **Administrative Management**:
   - Admins oversee system health, monitor user activity, and manage user accounts.

3.7 Supplementary Requirements

Additional requirements for the project include:

1. **User Documentation**: Provide a user manual and help resources for onboarding and troubleshooting.
2. **Localization**: Support multiple languages to cater to a global audience.
3. **Analytics**: Integrate reporting tools to track user activity and system performance.
4. **Support Services**: Ensure 24/7 technical support and system maintenance.

4. System Architecture

4.1 Client-Server Architecture

The Cuisine Popular and festivals uses a client-server architecture to ensure efficient, centralized data processing and secure access across multiple user devices. This architecture provides a clear separation of responsibilities between the client side (frontend interface) and the server side (backend processing and database management).

- **Client Side (Frontend)**: The client side is built with HTML, CSS, and React, providing an intuitive user interface accessible through web browsers on desktops, tablets, and mobile devices.
- **Server Side (Backend)**: The backend, built using Firebase handles business logic, authentication, and data processing. It includes routes for various system functionalities, such as login, record entry, and case management, and interacts with the firebase database to store, retrieve, and manage criminal records.
- **Database**: The Firebase database stores structured data, including food records, user information. It ensures data consistency, reliability, and easy retrieval.
- **Security and Role-Based Access Control**: Role-based access control restricts data access according to user roles, ensuring that only authorized personnel can view or modify sensitive information.
-

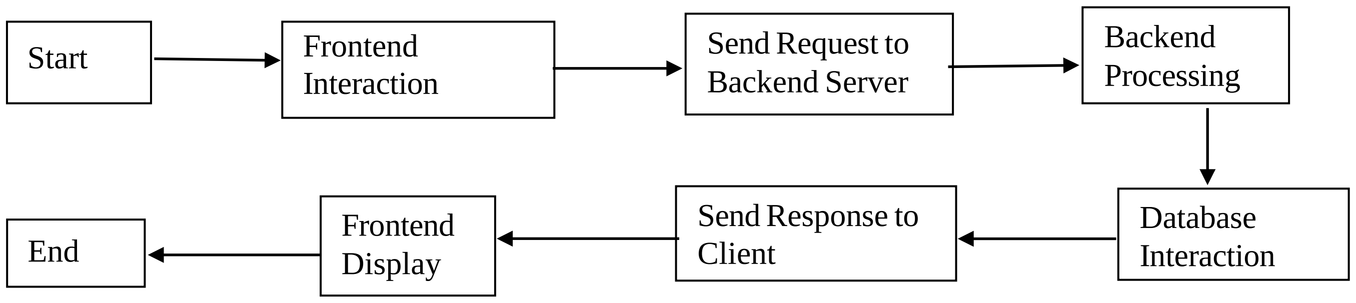


Figure 4.1: Client Server Architecture

4.2 Communication Interfaces

The communications interface**s** of the food delivery system facilitate secure, efficient data exchange between the client and server, as well as interoperability with other law enforcement systems. The primary communication protocols and interfaces are as follows:

- **HTTP/HTTPS Protocols**: The System uses HTTP/HTTPS protocols for secure client-server communication. All API requests from the client side to the server are sent over HTTPS to encrypt data in transit, protecting against unauthorized access or interception. HTTPS also ensures data integrity and authentication between the client and server.
- **RESTful API**: The backend server provides a RESTful API interface for handling requests from the client. The API follows standard REST principles for CRUD operations (Create, Read, Update, Delete), enabling consistent and predictable communication between the frontend and backend. The API includes endpoints for login, order management, search functionality, and statistical data retrieval, allowing smooth interaction and data exchange between client and server.
- **Authentication Interface**: The system uses Firebase for user authentication. When users log in, they receive a JWT, which is included in each subsequent request for secure authentication. The server validates the token, ensuring only authenticated users access the system.

5. Overall Description

5.1 Product Feature

The key features of the *Web-Based Mail Client Model* are as follows:

1. **User Authentication**
   - Secure login with support for multi-factor authentication (MFA).
   - Password recovery and account management options.
2. **Inbox Management**
   - Display and organize emails using folders, labels, and filters.
   - Advanced search functionality to quickly locate emails based on keywords, dates, or senders.
3. **Email Composition and Sending**
   - Rich text editor for formatting email content.
   - Ability to attach files, images, and documents.
   - Save drafts for later completion and sending.
4. **Real-Time Synchronization**
   - Seamless integration with IMAP and SMTP protocols for real-time email synchronization.
   - Automatic updates to reflect changes across devices.
5. **Spam Filtering and Security**
   - Intelligent spam detection using machine learning algorithms.
   - End-to-end encryption to protect user data during transmission.
6. **Contact Management**
   - Add, edit, delete, and group contacts for efficient communication.
   - Import/export contact lists for easy management.

7. **User Settings and Customization**
    - Personalize themes, layouts, and notification preferences.
    - Language and accessibility settings for an inclusive user experience.
8. **Integration with External Tools**
    - Calendar integration for scheduling and event management.
    - Compatibility with third-party productivity tools via APIs.
9. **Mobile and Responsive Design**
    - Fully responsive interface compatible with desktops, tablets, and smartphones.
    - Optimized for different screen sizes and browsers.
10. **Analytics and Reporting**
    - Email usage statistics, such as the number of emails sent, received, and categorized.
    - Reports on storage usage and performance.
11. **Administrative Controls**
    - Role-based access for user and admin accounts.
    - System monitoring tools for maintaining health and security.

5.2 ER Diagram

The Entity-Relationship (ER) Diagram for the Food ordering site represents relationships between entities such as Customer, Menu , Orders and Bank payments etc**.**

- **Entities:**
    - User: Attributes include userID, name, role, and contactInfo.
    - Order Record: Attributes include orderID, CustomerName,total.
    - Food record: Attributes include IsPrepared, IsDelivered, Remarks,Quantity.
    - Menu: Attributes include MenuID,Price ,Availability.

- **Relationships:**
    - User-Orders Record: One-to-many, with users authorized to add/update records.

- Order-Food Record: many-to-many, associating orders with food records.

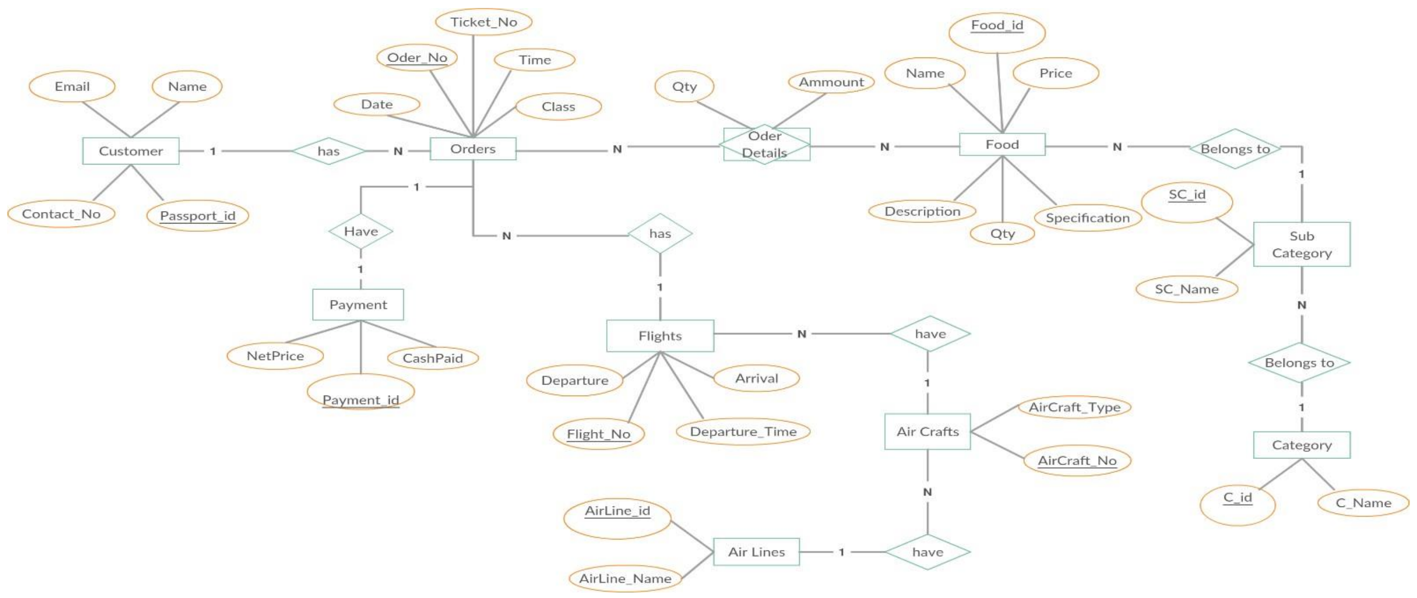


Fig 5.2.1: ER Diagram

5.3 Class Diagram

The **Class Diagram** outlines the structure and interactions of main classes in the Food delivery system:

- **Classes**:
    - User: Attributes include userID, name, role, and contactInfo.
    - Order Record: Attributes include orderID, CustomerName,total.
    - Food record: Attributes include IsPrepared, IsDelivered, Remarks,Quantity.
    - Menu: Attributes include MenuID,Price ,Availability.
- **Associations**:
    - User interacts with Menu Record and Order.

- **Methods**:
    - User methods include Send_Order(), Receive_Order().
    - MenuRecord methods include UpdateMenu(), DeleteMenu(),GetMenu().
    - Order methods include Confirm(), Cancel().

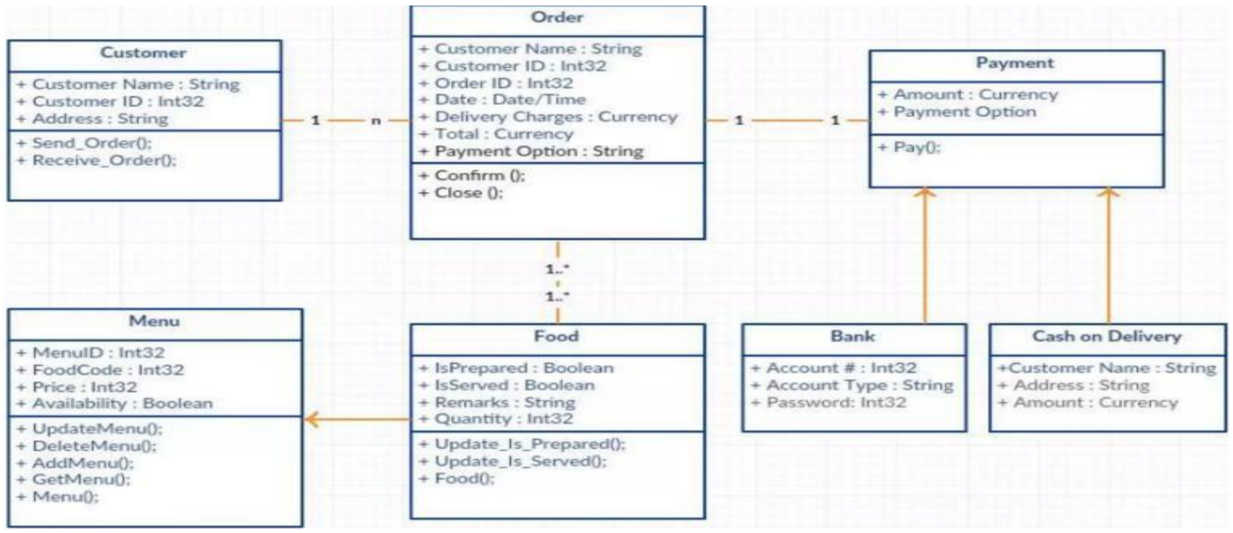


Fig 5.3.1: Class Diagram

5.4 Use-case Model Survey

The Use-Case Model Survey includes primary user actions in site:

- **Login and Authentication**: Users authenticate to access features.
- **Manage Order Records**: Users add, update, and view orders.

5.5 Behavior Diagrams

➢ 5.5.1 Sequence Diagram

Login Process Sequence:

1. User initiates login by submitting credentials.
2. System validates credentials against the database.

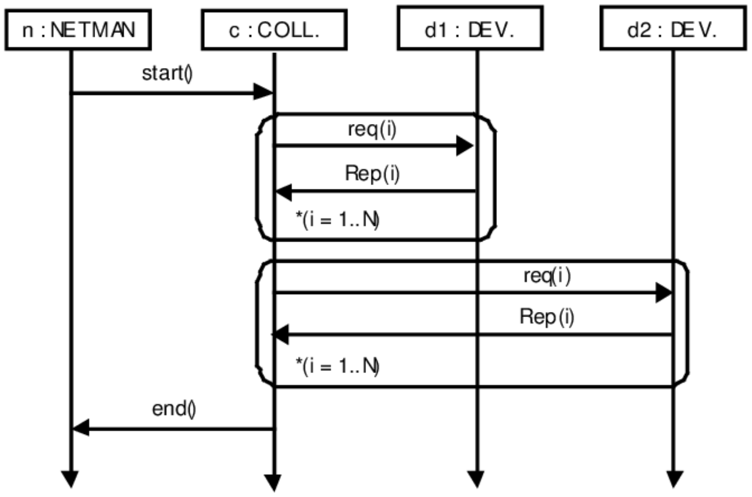


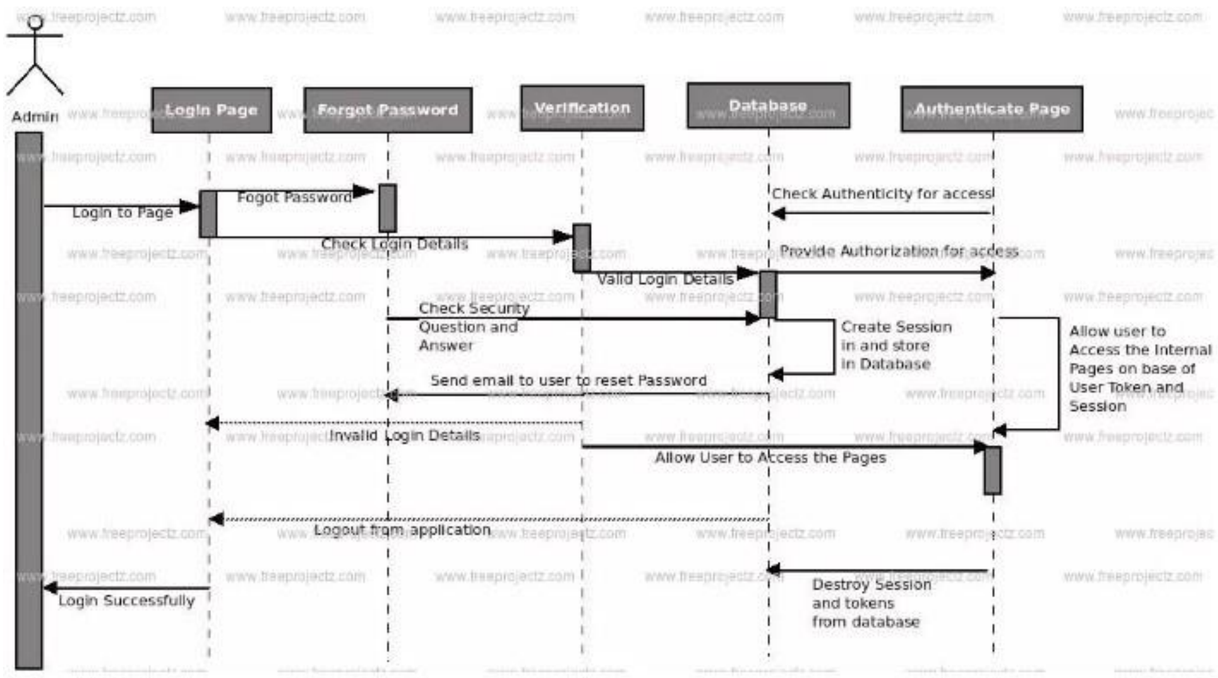
Fig 5.5.1: Sequence Diagram

➢ 5.5.2 Activity Diagram

Record Entry:

1. Start.
2. User selects "Display Menu".
3. Enter details and submit.

Fig5.5.2: Activity Diagram


Admin
Login Page
Forgot Password
Verification
Database
Authenticate Page
Login to Page
Fogot Password
Check Login Details
Check Authenticity for access
Provide Authorization for access
Valid Login Details
Check Security Question and Answer
Create Session in and store in Database
Allow user to Access the Internal Pages on base of User Token and Session
Send email to user to reset Password
Invalid Login Details
Allow User to Access the Pages
Logout from application
Destroy Session and tokens from database
Login Successfully

5.6 Assumptions and Dependencies

5.6.1 Assumptions

1. **User Access to Stable Internet**
   - It is assumed that users will have a stable and reliable internet connection to access the web-based email client.
2. **Browser Compatibility**
   - Users are expected to access the system through modern web browsers (e.g., Chrome, Firefox, Safari, or Edge) that support JavaScript, HTML5, and other essential technologies.
3. **Standard Hardware Resources**
   - Client devices, such as desktops, laptops, or mobile devices, will meet minimum hardware requirements for smooth operation.
4. **Compliance with Security Practices**
   - Users and administrators will adhere to recommended security practices, such as setting strong passwords and enabling multi-factor authentication (MFA).
5. **Email Standards Support**
   - Email providers and servers used by users will comply with standard email protocols (SMTP, IMAP, and POP3) for seamless communication.

5.6.2 Dependencies

1. **Third-Party Email Services**
   - The system depends on external email servers and services to facilitate sending and receiving emails through SMTP and IMAP protocols.
2. **Cloud Infrastructure**
   - Hosting the mail client will require reliable cloud infrastructure to ensure scalability, uptime, and data security. Examples include AWS, Azure, or Google Cloud.
3. **Database Management System**
   - The performance and reliability of the mail client depend on the underlying database system (e.g., MySQL or PostgreSQL) for storing user and email data.
4. **APIs and Libraries**

- The project relies on various APIs and libraries for essential functionalities, such as email communication, file uploads, spam filtering, and encryption.

5. **Regulatory Compliance**
   - The system must adhere to legal and regulatory requirements, such as GDPR or CCPA, to ensure user data privacy and compliance with global standards.
6. **Development Tools and Frameworks**
   - The successful implementation of the project is dependent on the availability and compatibility of development tools, such as Django/Flask for the backend and React.js/Angular for the frontend.

6. Supporting Information

6.1 List Of Figures

7. Conclusion and Future Scope

7.1 Conclusion

The *Web-Based Mail Client Model* represents a comprehensive solution to modern email communication needs, combining user-centric design with advanced functionalities. By leveraging web-based technologies, this model ensures platform independence, accessibility, and seamless synchronization across devices. Key features such as real-time email management, spam filtering, contact management, and enhanced security protocols make it a reliable and efficient tool for personal and professional communication. The adoption of Agile methodology and adherence to industry standards ensure that the system is robust, scalable, and user-friendly. This project demonstrates how integrating intelligent automation, secure communication protocols, and responsive design can enhance the user experience and efficiency of email systems.

7.2 Future Scope

The *Web-Based Mail Client Model* has significant potential for future enhancements and integrations to stay aligned with evolving user expectations and technological advancements. Possible areas of growth include:

1. **Artificial Intelligence Enhancements**
   - Introducing advanced AI features such as predictive email drafting, smarter spam detection, and personalized email organization.
2. **Blockchain for Enhanced Security**
   - Leveraging blockchain technology to enhance data integrity and ensure tamper-proof email communication.
3. **Voice Integration**
   - Incorporating voice-to-text functionality and voice commands for hands-free email management.
4. **Mobile App Development**
   - Expanding the system to native mobile applications for iOS and Android to improve accessibility and user experience.
5. **Integration with IoT Devices**

- Enabling email notifications and management through IoT-enabled devices like smartwatches and voice assistants.

6. **Collaboration Tools**
    - Adding shared workspaces, team email threads, and collaborative document editing to enhance team productivity.
7. **Advanced Analytics**
    - Providing deeper insights into email usage, trends, and performance through visualized data dashboards.
8. **Globalization and Localization**
    - Expanding support for multiple languages and cultural formats to cater to a diverse, global user base.

8. Concerns / Queries / Doubts if any:

This section highlights potential concerns, unresolved queries, and areas requiring further clarification or exploration for the successful implementation and operation of the *Web-Based Mail Client Model*.

1. **Scalability Concerns**
    - Will the system efficiently handle a large and growing user base without compromising performance?
    - Are the chosen technologies, such as the database and cloud infrastructure, scalable enough for future demands?
2. **Security Challenges**
    - How will the system address potential threats like phishing, malware attacks, and unauthorized access?
    - Are the implemented encryption methods and authentication mechanisms sufficient to protect sensitive user data?
3. **Compliance with Regulations**
    - How will the system ensure compliance with varying regional and global data protection laws, such as GDPR and CCPA?
4. **Third-Party Dependencies**

- What are the risks associated with relying on third-party APIs, libraries, or email servers for critical functionalities?
- How will these dependencies impact the system's reliability and security in the long term?

5. **User Experience Optimization**
    - How can the interface balance simplicity and functionality for both novice and advanced users?
    - Are there provisions for accessibility features, such as screen readers and high-contrast modes, to support users with disabilities?
6. **Resource Management**
    - Is the allocated budget sufficient for development, testing, deployment, and maintenance?
    - What is the estimated timeline for completing each development phase, and are there potential risks of delays?
7. **System Maintenance and Upgrades**
    - How frequently will updates and patches be applied to address bugs, vulnerabilities, or feature enhancements?
    - What processes are in place to ensure minimal downtime during maintenance or upgrades?
8. **Feedback and Support System**
    - How will user feedback be collected and incorporated into future iterations of the system?
    - What mechanisms are available for providing users with timely technical support?

Addressing these concerns and queries will require detailed planning, effective communication among stakeholders, and a proactive approach to potential risks and challenges. Identifying solutions to these issues early in the development process will help ensure the project's success and user satisfaction.