**EmailEssence SRS**

**v 1.0  
Authors : Emma Melkumian Joseph Madigan Shayan Shahla Ritesh Samal**

**Change History**

| **Date** | **ver** | **Summary of Changes** | **Author(s)** |
| --- | --- | --- | --- |
| **10/16/24** | **0.1** | **Initial Document Review** | **All** |
| **10/21/24** | **0.2** | **Requirements Review** | **All** |
| **10/24/24** | **1.0** | **First iteration** | **All** |
| **11/05/24** | **1.1** | **Pre-Midterm Revisions : AI Integration** | **JM** |
| **02/27/24** | **1.2** | **Mid-Year Revision: Updates** | **EM & SS** |
|  |  |  |  |
|  |  |  |  |

## 1. Introduction

### 1.1 Purpose

The purpose of this Software Requirements Specification (SRS) document is to detail the requirements for the development of EmailEssence, an email companion software aimed at improving the email management experience. The software helps users prioritize important emails at a glance, streamline their workflow through automatic email summarization, and reduce inbox clutter. This document serves as a guide for the development team, any stakeholders, and users, defining both functional and non-functional requirements to ensure that EmailEssence meets its goals of enhancing productivity and providing a seamless user experience.

### 1.2 Scope

This document outlines the complete specifications for the **EmailEssence** project, covering:

* Functional and Non-functional Requirements
* Systems Features
* External Interfaces
* Performance Expectations
* Design Constraints

It serves as a comprehensive guide for the development, ensuring the successful implementation of EmailEssence. The software is designed to enhance email management by providing features such as email summarization, a clutter-free reader view, and a customizable dashboard.

Throughout this document, references will be made to the **Minimum Viable Product (MVP)**, which outlines the essential features and design elements that are necessary for the initial launch. These features are considered non-negotiable to meet the core objectives of the project. Features or design elements that go beyond the MVP and are considered more ambitious or stretch goals are marked as **Feature Complete (FC)**, indicating they may be included in future iterations based on feasibility and time constraints.

### 1.3 References *-* [*IEEE Recommended Practice for Software Requirements Specifications*](http://www.math.uaa.alaska.edu/~afkjm/cs401/IEEE830.pdf) *-* [*Gmail API*](https://developers.google.com/gmail/api/guides) *-* [*IMAP Protocol*](https://docs.python.org/3/library/imaplib.html) *-* [*NLP Python Libraries*](https://www.geeksforgeeks.org/nlp-libraries-in-python/) *-* [*Typescript*](https://www.typescriptlang.org/docs/) *-* [*React*](https://legacy.reactjs.org/docs/getting-started.html) *-* [*Express.js*](https://expressjs.com) *-* [*Electron.js*](https://www.electronjs.org/docs/latest)

### 1.4 Definitions and Abbreviations

**AI**: Artificial Intelligence

**API**: Application Programming Interface

**ERD**: Entity-Relationship Diagram

**FC**: Feature Complete

**IMAP**: Internet Message Access Protocol

**ML**: Machine Learning

**MVP**: Minimum Viable Product

**NLP**: Natural Language Processing

**OAuth2**: Open Authorization 2.0

**RESTful**: Representational State Transfer

**UI**: User Interface

**UX**: User Experience

## 2. Overall Description

### 2.1 Product Perspective

EmailEssence fits within the broader ecosystem of email management tools by integrating directly with popular email services via **OAuth2** authentication and **APIs** (e.g., Gmail API, IMAP). The software’s main focus is to enhance email organization and productivity through **Natural Language Processing** (NLP) and **Machine Learning** (ML) techniques.

EmailEssence utilizes **AI-powered** summarization to automatically condense email content, focusing on relevance and clarity. By integrating **large language models** (LLMs), the system generates high-quality summaries that help users quickly understand important messages without reading each email in full. This feature, along with **NLP-based categorization**, aims to provide users with a simplified, clutter-free email experience.

For the **Minimum Viable Product** (MVP), the core functionality will focus on:

* Basic email retrieval.
* A **reader view** for simplified email display.
* Fundamental email summarization using **AI integration**.
* User authentication.
* Basic UI to display emails.

The Feature Complete (FC) version will extend this functionality to include **keyword analysis**, **advanced summarization** with additional options, and **cross-platform support** to ensure a consistent experience across both web and desktop applications.

**EmailEssence** will interact with external systems (e.g., Gmail API) for email retrieval and processing. These interactions depend on **secure API access**, and all email data will be processed either client-side or server-side with strong encryption to protect user privacy.

### 2.2 Product Features

The following are the **high-level features** for **EmailEssence**:

**Minimum Viable Product** (MVP)**:**

* **Email Retrieval**:
  + Retrieves emails securely from external services using the **Gmail API** or **IMAP**.
* **AI-Powered Email Summarization**:
  + Automatically generates concise summaries for emails using an **AI integration**, allowing users to quickly understand core content.
* **Minimal Web Interface**:
  + **Login Page:**
    - Basic login page for user authentication.
  + **Dashboard:**
    - Basic view of retrieved emails and summaries.
  + **Email List View:**
    - Users can see a list of emails with basic metadata.
  + **Reader View:**
    - Users can access a simplified **Reader View** for each email that removes unnecessary formatting to focus on the core content.
  + **Basic User Preferences:**
    - Minimal configuration options allowing users to control how frequently emails are fetched.

**Feature Complete** (FC)**:**

* **Extended Web Interface**:
  + **Modular Dashboard:**
    - A more dynamic and customizable dashboard, allowing users to organize and manipulate their view of their dashboard with modules.
  + **Full Email interface**
    - Users can view, reply, and interact directly with their inbox.
  + **Expanded Summarization:**
    - Advanced **AI-based summaries** with additional features such as keyword highlighting, and topic identification.
  + **User Settings and Customization**:
    - Users have access to **UI customizations** like themes and dashboard layouts.
* **Keyword Analysis**:
  + Uses **NLP** to analyze emails and identify key phrases or topics to further highlight important messages for the user.
* **OAuth 2.0 Authentication**:
  + Securely integrates with external email services using **OAuth 2.0**, allowing users to connect their email accounts without exposing login credentials.
* **Cross-Platform Desktop Support**:
  + The application will be designed to function both as a **web app** and a **desktop application** (via **Tauri**), ensuring cross-platform compatibility.

Beyond **Feature Complete** (FC):

* **Feature Rich Web Interface:**
  + **Interactive Email List:**
    - Provides filtering and sorting options with tags, priorities, and custom user set categories (See Advanced Categorization) along with the capability to reply and edit emails within the interface.
  + **Enhanced Reader View:**
    - Richer customization in the Reader View, including key section highlighting, setting follow-up reminders, and markdown notes output.
  + **User Settings and Customization:**
    - Users can set advanced preferences, including custom summary lengths, specific email rules (e.g., always summarize emails from a particular sender).
* **Multi-Account Support:**
  + Users will be able to connect multiple email accounts and manage them from a unified interface.
* **Advanced Categorization via ML:** 
  + Using machine learning, the system will classify emails into categories (e.g., work, personal, promotions).
* **Spam Detection:** 
  + The system will detect and flag spam emails using custom algorithms and external data sources.

### 2.3 User Classes and Characteristics

**General User**:

* These users receive a large volume of emails and are looking for a simplified way to navigate their inboxes. They value features such as **email summarization** and **clutter-free views** to reduce the time spent managing unnecessary emails. For general users, the primary focus is on **convenience and time savings**, with data security being important but not as critical as for business users.

**Business User**:

* Business users require more advanced features, including quick access to critical work-related emails and a strong emphasis on **data security**. In addition to efficient email summarization, business users expect robust security features such as **OAuth 2.0 authentication**, **end-to-end encryption**, and strict access control to protect sensitive business information.

### 2.4 Operating Environment

**Frontend:**

* **React**: For building the web interface, ensuring a responsive and user-friendly experience that adapts to users needs.

**Backend:**

* **Python**: Used for handling core backend functions, including NLP, email parsing, keyword analysis, and summarization.
* **NLP Libraries**: These include spaCy, NLTK, or transformers for handling natural language processing tasks.

**Middleware**:

* **Express.js**: Middleware for handling HTTP requests, facilitating communication between the frontend and backend.

**Desktop Application:**

* **Tauri**: Used to package the application as a desktop app that can run on **Windows, macOS,** and **Linux.**

**Email Services:**

* **Gmail API** or **IMAP**: External services for retrieving and managing data.

**Authentication:**

* **OAuth2.0**: For secure authentication and access to users’ email accounts.

### 2.5 Design and Implementation Constraints

**Dependency on External APIs:**

* Due to the nature of the service, APIs such as Gmail Email API will be necessary to handle email retrieval. This dependency on the email provider API can lead to constraints such as
  + **API Rate Limits**: Most APIs come with a limited number of requests per day which could affect the applications scalability
  + **API Availability**: Any API downtime or maintenance would effectively stop the service from functioning

**Processing Limitations:**

* Applications will face constraints when handling requests from limited resource environments such as low end pcs and spotty connections.
* Users may attempt to connect with large inboxes and the service must be able to handle their requests without performance degradation. This can be accomplished by implementing **incremental fetching** which would allow the service to load data in batches

**Asynchronous Processing:**

* For services such as email inboxes asynchronous processing allows tasks to be executed in parallel without the webpage losing responsiveness. This will be especially useful when the service may be making a database operation but the user will still be able to scroll through their inbox during the operation. These needs will be met through the JavaScript **async/await** or Python frameworks such as **asyncio**.

**Caching:**

* Operations such as summarizing large blocks of text could require large amounts of computation and can hinder optimization. Therefore implementing caching will alleviate some of the performance needs by allowing the application to operate once and cache the information for later retrieval.

**Security and Privacy Regulations**:

* **Data Encryption**: All data must be encrypted at rest and in transit, which entails that the service must use End-to-End encryption

**Data Compliance:**

* The service must follow all data privacy regulations such as CCPA since it is handling sensitive user data. This constraint entails implementation of features such as user consent and right to data erasure

**Authentication and Access Control:**

* To prevent unauthorized access and ensure that only authorized users can perform actions within the system, OAuth 2.0 will be implemented for authentication. Since each user will only access their own email data, complex access control systems such as RBAC are not required at this stage. The system will focus on maintaining individual data privacy through encryption and secure API interactions.

**Cross-Platform Support:**

* System is required to function on web environments(via React.js) and when FC, most desktop environments(via Tauri). Updates, performance, and design should be consistent across both versions.

**Third-Party Libraries**:

* **EmailEssence** relies on several third-party libraries and frameworks. This introduces risks related to library updates, vulnerabilities, and potential deprecations.

**Resource Constraints**:

* Given that the initial development team consists of a small group of college-level students, there are resource limitations in terms of development time, expertise, and infrastructure. This constrains the complexity of features that can be implemented in the MVP. Features such as multi-account support, advanced categorization, and enterprise-level security may need to be deferred to future iterations based on available resources and development capacity.

## 3. Specific Requirements

### 3.1 Functional Requirements

**Minimum Viable Product (MVP)**

* **FR01 - Email Retrieval**:
  + **Requirement**: The system must retrieve emails securely using the Gmail API or IMAP. Users must authenticate via OAuth 2.0 to allow secure access to their email accounts.
  + **User Story**: As a user, I want to securely connect my Gmail or other email account so that I can view my emails in the application.
  + **Acceptance Criteria**: The system retrieves emails after the user authorizes access via OAuth 2.0 and displays them in the inbox view.
* **FR02 - AI-Powered Email Summarization**:
  + **Requirement**: The system must automatically generate high-quality email summaries using an AI-driven model (LLM) to condense email content into 1-2 sentences, capturing the essential points for user convenience.
  + **User Story**: As a user I want my emails summarized using AI, so that I can quickly understand important information without needing to read the full email.
  + **Acceptance Criteria**: Upon fetching an email, the system uses the integrated AI model to generate a concise summary.
    - Summaries are displayed in the inbox list view with key points visible at a glance.
* **FR03 - Minimal Web Interface**:
  + **Requirement**: The system must provide a minimal web interface, including a Dashboard, Reader View, and Basic User Preferences.
    - **FR03.1 - Dashboard**: Users should see a tailored overview of their emails, displaying the sender or short summary.
      * **User Story**: As a user, I want to see an overview of my inbox so that I can quickly decide which emails to read.
      * **Acceptance Criteria**: The dashboard displays a list of emails with sender information, and timestamps.
    - **FR03.2 - Reader View**: Users must be able to open an email in a simplified reader view that removes extraneous formatting.
      * **User Story**: As a user, I want to open emails in a clean, easy-to-read format without unnecessary formatting.
      * **Acceptance Criteria:** The system displays emails in a simplified format when opened.
    - **FR03.3 Basic User Preferences**: Users should be able to configure basic settings, such as frequency of email fetches, whether summaries are generated automatically, and theme.
      * **User Story**: As a user, I want to control how often emails are fetched and the appearance of the application.
      * **Acceptance Criteria**: Users can modify preferences through the settings interface.
    - **FR03.4 Login:** Users should be able to login in a secure manner, connecting with their Gmail Account.
      * **User Story**: As a user, I want to be able to login into EmailEssence knowing that my data is secure.
      * **Acceptance Criteria**: Users can access EmailEssence dashboard once they have been authenticated and their credentials are securely stored.
* **FR04 - Extended Web Interface**:
  + **Requirement**: The system must provide a more dynamic, modular dashboard, allowing users to customize their inbox views and settings.
    - **FR04.1 Modular Dashboard**:
      * **User Story**: As a user, I want to customize my dashboard view by organizing emails based on importance, categories, or tags.
      * **Acceptance Criteria**: Users can rearrange and customize dashboard elements to suit their preferences.
    - **FR04.2 User Settings and Customization:**
      * **User Story**: As a user, I want to adjust my UI settings (e.g., themes, dashboard layouts) to personalize my experience.
      * **Acceptance Criteria:** Users can modify UI settings, including layout, themes, and notification preferences.
* **FR05 - Keyword Analysis**:
  + **Requirement**: The system must analyze email content using NLP to highlight key phrases or topics for the user.
  + **User Story**: As a user, I want my emails to be analyzed so that important keywords or topics are highlighted.
  + **Acceptance Criteria**: The system identifies and highlights important keywords within the email summary or reader view.
* **FR06 - OAuth 2.0 Authentication:**
  + **Requirement**: The system must securely integrate with external email services using OAuth 2.0 for authentication.
  + **User Story**: As a user, I want to connect my email account securely without exposing my login credentials.
  + **Acceptance Criteria**: The system uses OAuth 2.0 to authenticate users and access their email data securely.
* **FR07 - Cross-Platform Desktop Support**:
  + **Requirement**: The system must function as both a web app and a desktop application using Electron.js, ensuring consistent cross-platform compatibility.
  + **User Story**: As a user, I want to access my emails via both the web and desktop application with a consistent experience.
  + **Acceptance Criteria**: The web and desktop versions of the app offer a unified experience, with consistent features across platforms.

*Beyond FC Features pending Functional Requirements ahead of implementation.*

### 3.2 Non-Functional Requirements

* **Performance Requirements**
  + ***Processing Time****: The system should limit the time taken to process emails for summaries and reader views.*
    - ***Metric****:* 90% of API responses should be completed within 3 seconds. The remaining 10% should complete within 10 seconds, with clear feedback provided to the user in cases of extended processing time.
  + **Incremental Email Handling for Large Inboxes:** *The system should efficiently handle large inboxes by processing emails incrementally, ensuring performance remains consistent regardless of inbox size.*
    - ***Priority Handling****:* Prioritize processing for emails received within the last 14 days to improve relevance and responsiveness*.*
    - ***Caching****:* Utilize caching mechanisms to avoid redundant fetching of email data, improving both performance and response times for previously processed content.
  + ***On-Demand Reader View Processing****: Emails should only be processed into reader view when explicitly requested by the user, preventing unnecessary resource consumption.*
    - ***Metric****:* Reader view processing should occur in real-time, with most requests processed in under 2 seconds.
  + ***Asynchronous Processing****: The system should implement asynchronous processing for email fetching and summarization, allowing the system to handle large volumes of emails and traffic without blocking other processes.*
    - ***Concurrency****:* The system should support thousands of concurrent users with no significant degradation in performance.
    - ***Scalability****:* The architecture should scale seamlessly with increased user traffic and be capable of handling exponential growth in the number of emails processed, ensuring performance remains stable as email load increases.
* **Security Requirements**
  + **Authentication**: The system must use OAuth 2.0 for secure authentication with email providers. This ensures that users’ credentials are never exposed to the system, only the tokens required to access their emails.
  + **Data Encryption**: Sensitive data such as email summaries, access tokens, and user preferences should be encrypted both at rest and in transit using strong encryption standards (e.g., AES-256 for data at rest and TLS/SSL for data in transit).
  + ***Access Control****:* The system must implement strict access control mechanisms to ensure that only authenticated and authorized users can access email data.
  + ***Data Retention and Deletion****:* Users should have control over their data, including the ability to delete email summaries or revoke access to their email accounts at any time. The system should comply with relevant data protection regulations (e.g., GDPR) by ensuring that data is promptly deleted upon user request.
* **Usability Requirements***:*
  + ***Intuitive User Experience (UX)****:* The system should provide an intuitive user interface that allows new users to navigate and understand the core features within 2 minutes of first use, without the need for external guidance or tutorials.
  + ***User Onboarding****:* The onboarding process should be simple and guide users through connecting their email provider(s) with minimal friction.
    - ***Metric****:* Users should be able to connect their email account within 30 seconds of starting the onboarding process*.*
  + ***Customization****:* Users should be able to easily customize basic settings, such as email filtering preferences and dashboard layout, without going through complex steps.
    - ***Metric****:* Key settings should be accessible within 1-2 clicks from the main interface.
  + **Quick Access***:* The system should be designed to minimize layers of navigation, ensuring that common actions (such as viewing email summaries or retrieving emails) can be completed in no more than 1 click from the main dashboard.
* **Accessibility Requirements:**
  + **Cross-Platform Support**: The system must operate seamlessly on major desktop operating systems (Windows, macOS, and Linux) without requiring additional software installations or complex configurations.
  + **Browser Compatibility**: If accessed via a web interface, the system must support the latest versions of all major browsers, including Chrome, Firefox, Safari, and Edge.
  + **Screen Reader and Keyboard Navigation Support:** The system should be designed to comply with accessibility standards (e.g., WCAG 2.2), ensuring that users with disabilities can effectively navigate the interface using screen readers and keyboard shortcuts.
    - **Metric**: All interactive elements (e.g., buttons, links) should be fully accessible using keyboard navigation and compatible with popular screen readers.
  + **High Contrast and Customizable Fonts**: Users should be able to switch to a high contrast mode and adjust fonts to meet their individual accessibility and preferential needs.
    - **Metric**: These options should be available in the settings menu within 2 clicks and should be applied system-wide.

## 4. External Interface Requirements

*List and describe the external interfaces, including user interfaces, hardware interfaces, software interfaces, and communication interfaces. Provide mock-ups of the user interfaces (UI) and give each interface a unique ID*

**User Interface:**

* **Dashboard (EIR01):** 
  + Displays urgent email summaries based on AI determined priority.
  + Users can click on the view more button to open the complete email in a separate view.
* **Reader View (EIR02):**
  + Offers a simplified view of the emails by removing irrelevant content, like extra images or formatting.
  + Activated by a toggle button above the email view.
* **Summaries (EIR03):** 
  + Provides concise overview of email content next to each email title in the email list.
  + Allows users to understand key messages at a glance without opening the full email.

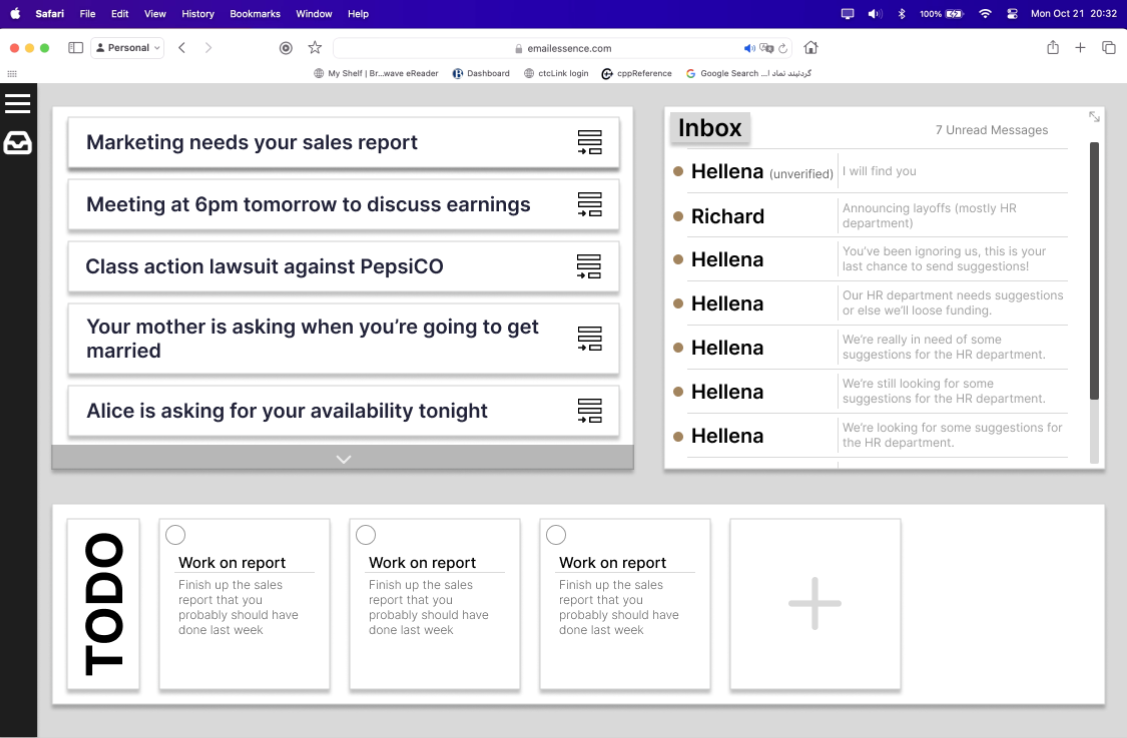
**Software Interface:**

* **OAuth2 (EIR04):** 
  + Used for secure authentication and token management to connect with users email accounts.
  + Integrates with supported email providers (like Gmail) for secure access.
* **Gmail API (EIR05):** 
  + Provides access to Gmail accounts for email retrieval, management, and manipulation .
* **IMAP Protocol (EIR06):** 
  + An alternative to Gmail API for email retrieval from IMAP supported email servers.
  + ensures compatibility with various email providers

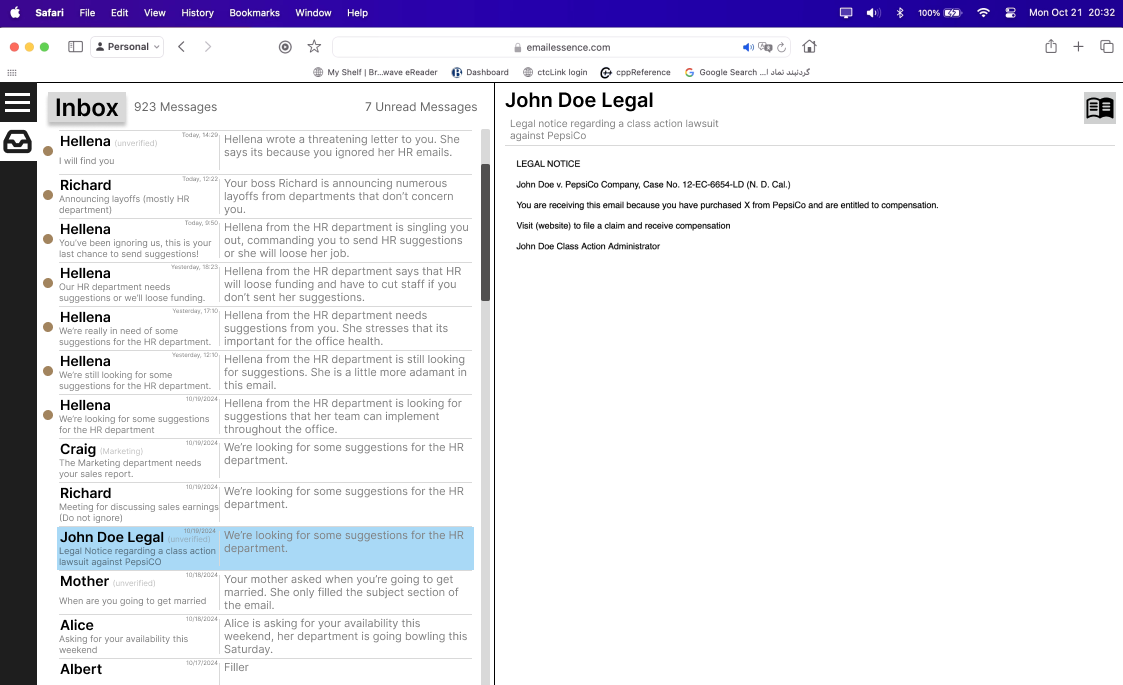
**Communication Interface:**

* **Express.js (EIR07):** 
  + Will act as middleware between frontend and backend, creating a RESTful APIfor handling requests and responses.

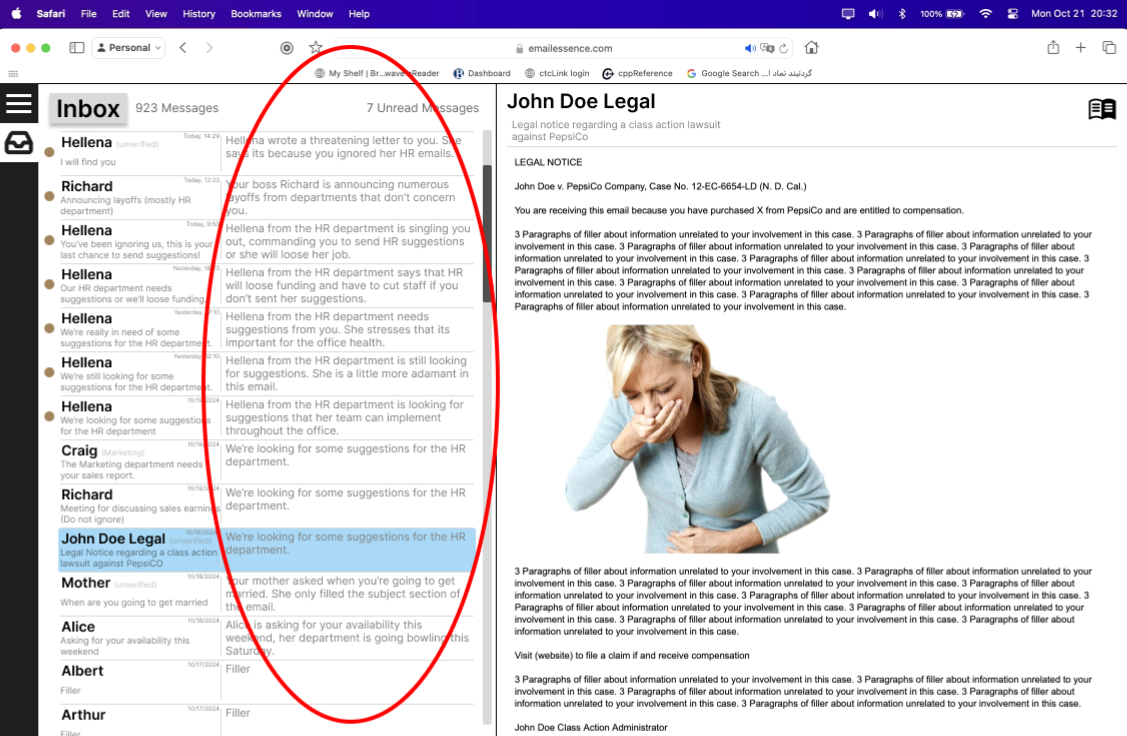
**EIR01:** In the Dashboard we will have a small message/summary from emails that our model deems urgent. The urgency of these emails will be determined on a weight that will be set for all emails. Along with the message/summary there will be an icon indicating to open up the full email that will be click/tappable, opening the full email in the inbox tab. A drop arrow will exist underneath to allow the user to see more emails based on this weighted system. The Dashboard will separate each email summary into a box.



**EIR02:** A Reader view will use our model to filter out any unnecessary space, images, and text that does not relate to the main topics of the email. The Reader view will be accessible by clicking/tapping the toggleable Reader view button above the email. It will only be available when the user has opened the full email.



**EIR03:** Our program will feature Summaries. These Summaries will be visible beside an email and will provide a brief overview on the overall message of the email. We will achieve this by using our model to extract necessary information. Summaries will only be viewable when the full inbox tab is open.



## 5. Constraints

*[List any constraints or limitations that might impact the software development.]*

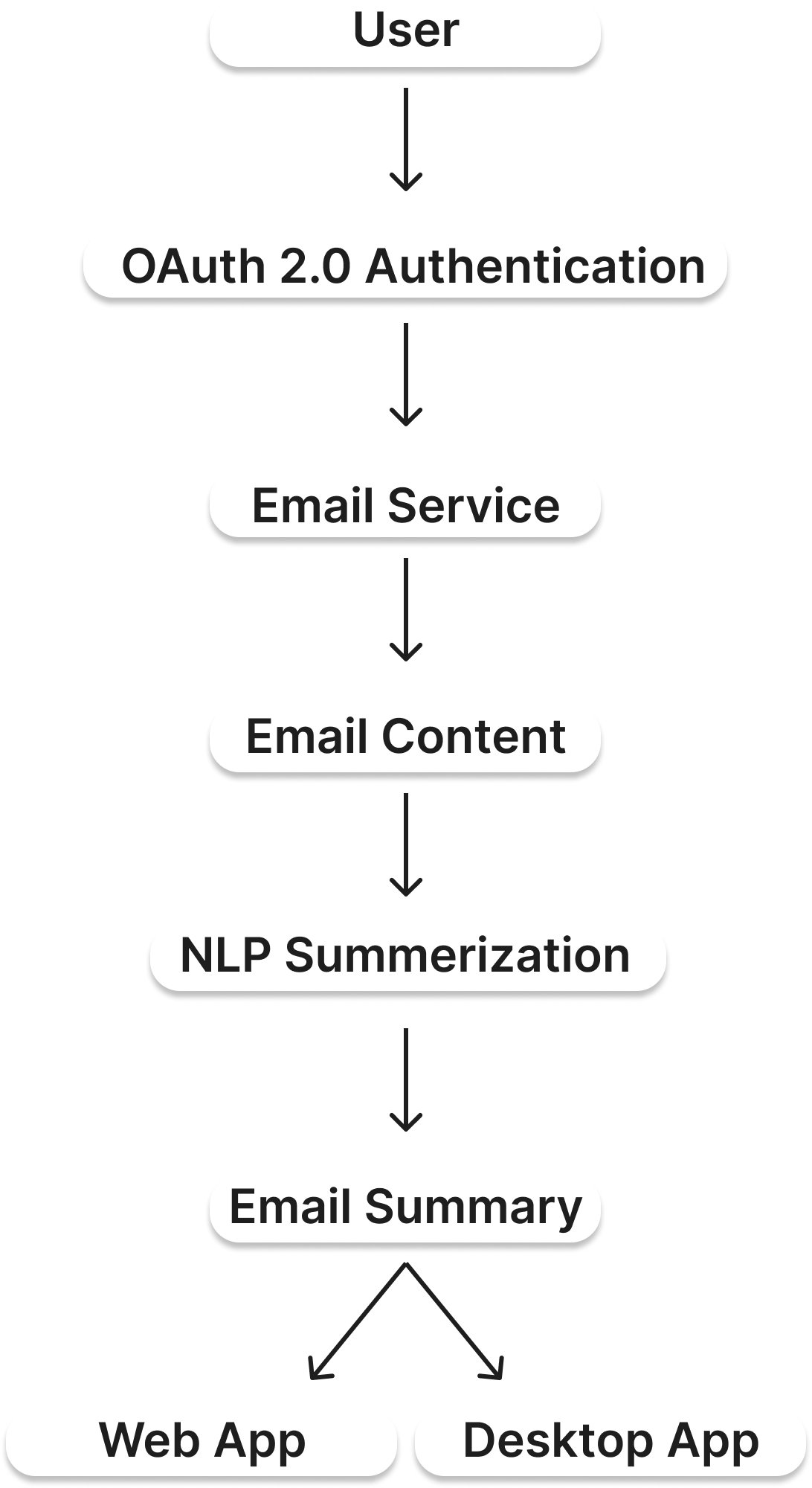
* **Internet Access***:* The system’s core functionality, including email retrieval and summarization, depends on active internet connectivity.
* **Large Inbox Handling**: The system should be capable of efficiently handling inboxes with thousands of emails without significant slowdowns. Processing time for retrieving and summarizing large volumes of emails may be difficult on low end machines and slow internet connections
* **Multi-Account Support Limitations**: Adding support for multiple email accounts (especially from different email providers like Outlook, and Yahoo) could introduce complexity in managing different API integrations
* **API Requests Rate**: Due to high volume of traffic to the web app we could hit the maximum of daily requests that the API allows.

## 6. Assumptions and Dependencies

* **User Consent for Email Access**:
  + It is assumed that users will grant the necessary permissions via OAuth 2.0 for the application to access their email accounts (e.g., Gmail, Outlook). The system cannot function without these permissions, and users who deny access will not be able to utilize the application and its corresponding features.
* **Stable API Access*:***
  + The system assumes that external services (e.g., Gmail API, Outlook API) will remain stable and accessible under normal operating conditions. It is expected that these APIs will continue to provide the necessary functionality for email retrieval and access.
* **OAuth 2.0 Authentication***:*
  + The system’s authentication mechanism depends on OAuth 2.0 for secure access to users' email accounts. Any changes or restrictions in OAuth implementations by external services (e.g., new security requirements or scopes) may necessitate updates to the system.
* **Third-Party Libraries and Frameworks**:
  + The backend and frontend rely on third-party libraries (e.g., Node.js, React.js, Express.js, Tauri) for core functionality. Updates, vulnerabilities, or deprecations in these libraries could affect system performance, security, or development timelines.
* **Cloud Infrastructure**:
  + If the system is hosted on cloud services (e.g., AWS, Google Cloud), it depends on the scalability, reliability, and performance of those cloud services. Disruptions in cloud availability or changes in service pricing may impact system costs and performance.

## 7. Appendices

**Use Case Diagram:**



### Glossary

**AI (Artificial Intelligence):** A field of computer science that aims to simulate human intelligence in machines, enabling them to perform tasks like decision-making, problem-solving, and language understanding.

**API (Application Programming Interface):** A set of protocols and tools that allows different software applications to communicate and interact with each other.

**Asynchronous Processing:** A method of processing tasks concurrently, allowing the system to execute other operations while waiting for a specific task to complete, thus improving responsiveness.

**BERT (Bidirectional Encoder Representations from Transformers):** A transformer-based NLP model developed by Google that is widely used for text understanding and summarization tasks.

**CCPA (California Consumer Privacy Act):** A California state law that provides consumers with rights regarding the collection, use, and sharing of their personal information.

**Cross-Platform:** Software that is compatible with multiple operating systems, such as Windows, macOS, and Linux.

**ERD (Entity-Relationship Diagram):** A diagram that shows the relationships between different entities in a system, often used to model database structures.

**Electron.js:** A framework that allows the creation of cross-platform desktop applications using web technologies like HTML, CSS, and JavaScript.

**Express.js:** A web framework for Node.js, used to build APIs and middleware for backend services.

**FC (Feature Complete):** A version of the software that includes all the intended features, going beyond the MVP to offer a richer user experience.

**Frontend:** The client-side part of the application that users interact with directly, typically built using frameworks like React or Angular.

**Gmail API:** A RESTful API provided by Google that allows access to Gmail features, such as reading emails, sending messages, and managing inboxes.

**IMAP (Internet Message Access Protocol):** A protocol used by email clients to retrieve messages from a mail server, enabling access to both read and unread emails.

**Incremental Fetching:** The process of retrieving data in smaller chunks over time, rather than all at once, to reduce load and improve performance.

**MVP (Minimum Viable Product):** The initial version of a product that includes only the core features necessary to launch and validate its purpose.

**ML (Machine Learning):** A subset of AI that enables systems to learn from data and improve performance without explicit programming.

**NLP (Natural Language Processing):** A field of AI that focuses on the interaction between computers and human language, enabling tasks like text analysis, translation, and summarization.

**OAuth 2.0:** An open-standard protocol for secure, token-based authorization, commonly used for logging into third-party applications without exposing user credentials.

**Reader View:** A simplified view of email content that removes unnecessary elements like images, formatting, and ads, focusing on the main message.

**RESTful (Representational State Transfer):** An architectural style for building APIs that use standard HTTP methods (GET, POST, etc.) to enable communication between client and server.

**Screen Reader**: A software program that enables visually impaired users to read text displayed on the screen through speech synthesis or braille displays.

**Scalability:** The capability of a system to handle increased loads or expand its capacity while maintaining performance.

**Summarization:** The process of condensing the content of an email to its key points, using NLP techniques.

**Token:** A secure string of characters used in OAuth2 to grant access to protected resources without exposing login credentials.

**Transformers:** A class of deep learning models designed for NLP tasks like text classification, translation, and summarization, often implemented through libraries like Hugging Face.

**TLS/SSL (Transport Layer Security / Secure Sockets Layer):** Protocols that ensure secure communication over networks by encrypting data in transit.

**UI (User Interface):** The part of a software application that users interact with, including visual elements like buttons, menus, and text fields.

**UX (User Experience):** The overall experience and satisfaction a user has when interacting with a software application, considering aspects like usability, design, and functionality.

**WCAG (Web Content Accessibility Guidelines):** A set of guidelines aimed at making web content more accessible, particularly for people with disabilities.