Software Requirements Specification

for

<Email Project>

**Version 1.0 approved**

**Prepared by Adam, Johnny, David, Ricardo**

**<Group 3>**

**<09/02/2021>**

# Introduction

## Purpose

The purpose of this document is to build an email web application so users can send and receive messages with attachments, the ability to sort, search, and forward emails, and be able to login to the web application through a portal.

## Document Conventions

This document is written in 11-pt Ariel font using Microsoft Word as a means to write sections of the document collaboratively. Functional requirements are placed in tables to facilitate understanding of their design. The main sections are in enlarged font, bolded, and numbered to provide clarity and to make them distinguishable from each other. Sub-sections are numbered in decimal and also bolded for the same reason.

## Intended Audience and Reading

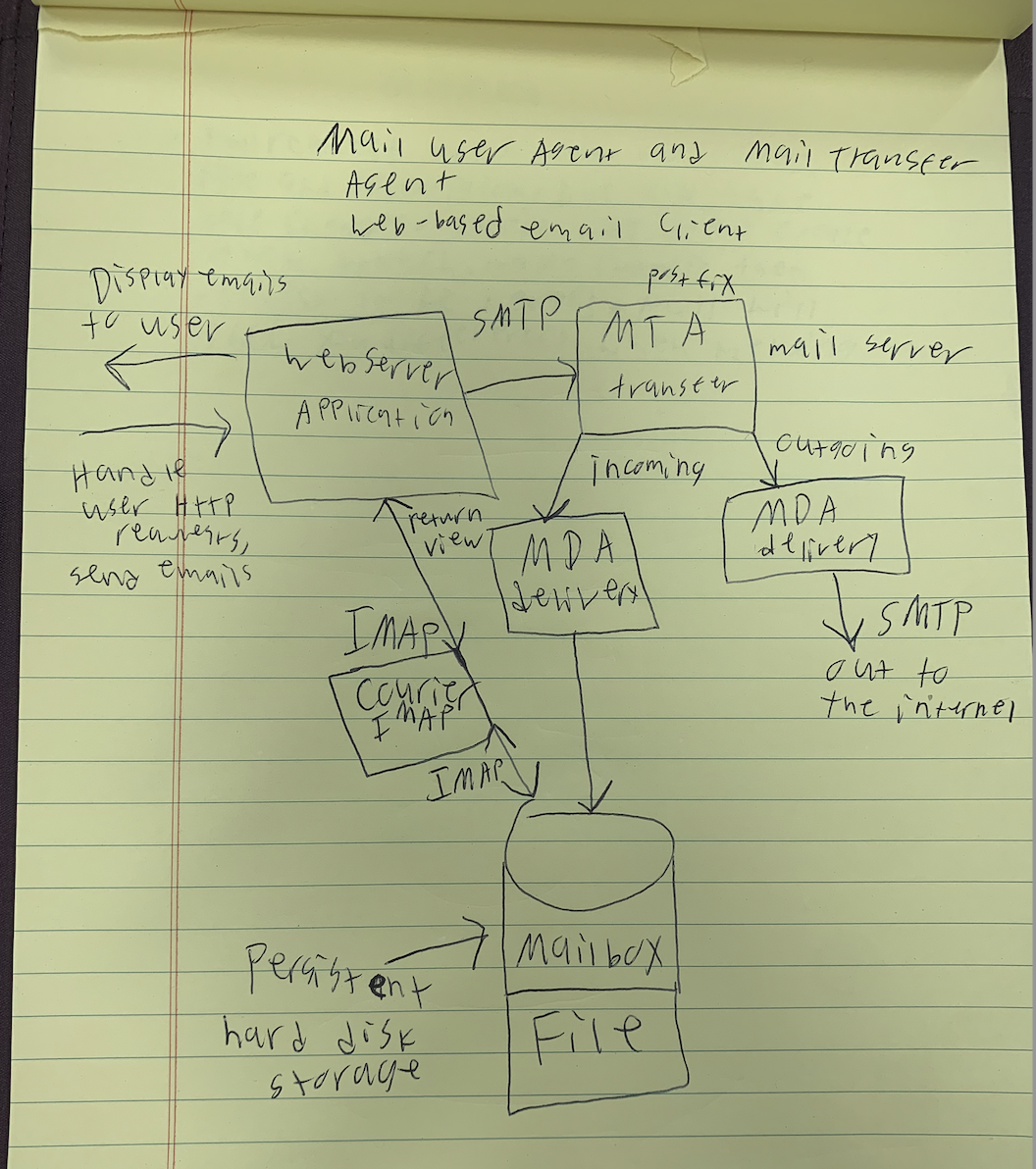
This document is intended for any student working on the web application, as well as the other students who may wish to interact with this lab. It is also for the instructor of Intro to Software Engineering and the lab instructors who administered the project requirements. This project will be useful to any CSE student who wishes to learn about web development as well as host/client communication between servers, remote data communication, and web design. The order in which one may read this document should be from section 01 through section 05.

## Product Scope

The purpose of the email web application project is to facilitate understanding of software development using AGILE techniques and development cycle. Furthermore, it will allow users to login to access emails, compose, edit and send emails to other users, search and forward emails to other users, and send attachments along with the email. We will use Python for data communication and to design the user interface. GitHub will be used in order to maintain prototype versions, share code, and maintain documentation.

## References

## We created an architecture sketch. (Shown Below)



# Overall Description

## Product Perspective

The product that is being developed is a follow-on member of a product family. It is an email client that will act and follow the same functions as an email client that has been developed such as gmail, outlook, hotmail, etc. It will be strictly online using a web framework that will communicate with a database storage to pull and receive data from each email sent and received. It will interact with a service using SMTP.

## Product Functions

List of Functions:

1. Search Bar: search is using a search engine based on keywords to sort through old emails.
2. Number of users: The system is able to support a large number of online users at a time.
3. Compose & Receive: The system should operate basic emailing functions, such as sending, receiving, attaching documents, etc.
4. User account: The system allows the user to create their accounts in the system and provide features of updating and viewing profiles.
5. Contact List: The system should allow the user to create a contact list that adds the name of the person and matches that person with their email address.

## User Classes and Characteristics

Users will create their own account with their own login credentials that will include security questions that will be answered to the specified user. Any kind of user will use this product, as everyone uses email in this era. Specific users that will use this product more than others would be large businesses for their employees. Large business owners that have an IT department should be able to monitor every employee's email that they receive and send. Normal users that use email for personal purposes will be able to send and receive emails and follow basic operations of the product without an IT department watching their activity. The most important users to satisfy are the large business users who need a sort of communication within their office.

## Operating Environment

The web server and email server would run on some linux based operating system on the cloud. Our clients can interface with it on any operating system. The client-side components of the software system must operate within common webbrowser environments.

## Design and Implementation Constraints

Any items or issues that will limit the options of the development team would be bandwidth because it will not be a powerful machine. There will be some memory requirements that will affect how much data we can store in our framework as well as the framework we chose and their policies they have for running our code on it. There will be some interfaces with other applications through a mailing server and database storage. There will be some communication protocols through REST API.

# System Features

## Web-Based User Interface

3.1.1 Description and Priority

This is of high priority. This part of the software architecture that allows for the user to create and send emails along with viewing existing emails. This is the view part of the application.

3.1.2 Stimulus/Response Sequences

This piece if the application should respond to http requests from the user. The display should be updated to reflect the state of the user's mail inbox.

3.1.3 Functional Requirements

REQ1: Allow users to log in (sessions). Use user authentication and reject wrong passwords.

REQ2: Users can compose and send emails to chosen recipients. Return a did not send error if mail was not sent.

REQ3: Search and forward emails to other users.

REQ4: Send Files and attachments along with emails. Return error if file is too large for a certain filetype.

## Mail Transfer Agent using SMTP

3.2.1 Description and Priority

This is of high priority. This part of the software architecture that handles all networking and routing of emails for the application.

3.2.2 Stimulus/Response Sequences

When a user requests an email be sent, the mail server must handle transmitting the email to the correct MTA or writing the email to the local mailbox if it belongs to a local user. The server must listen for incoming SMTP transmissions from the outside Internet. Once Received, the server should save these emails in a mailbox file.

3.2.3 Functional Requirements

REQ1: Listen for incoming SMTP transmissions. Error message if offline.

REQ2: Handle Routing of local SMTP transmissions. Error message if transmission fails.

REQ3: Write received messages to a Mailbox file on the hard disk. Error for corrupted message.

* 1. **IMAP Interface To Persistent Mailbox File**

3.1.1 Description and Priority

Use an IMAP interface to read emails from storage. This is the model part of the application.

3.1.2 Stimulus/Response Sequences

This piece if the application should use an IMAP interface to read emails from a persistent mailbox file. It should then provide this data to the web user interface for the user to see.

3.1.3 Functional Requirements

REQ1: Use IMAP interface to read users emails. Error for corrupted message.

REQ2: Pass along emails to be displayed to the user. Error rendering email in UI.

# Other Nonfunctional Requirements

## Performance Requirements

The main performance requirements should revolve around making a simple and efficient user interface for the consumer. It is crucial to succeed in this due to the nature of the goal at hand. With an emailing platform you want a simple and direct set of options that allow you to send or receive something from point A to B. The server part of the application must be able to handle the amount of traffic coming through the network. The application must also have adequate storage capabilities since it will be dealing with multiple users and all the people they choose to communicate with. The storage collectively encompasses several functions such as logins and passwords, contact lists, and storing emails.

## Safety Requirements

A user agreement would have to be in place in order to protect the product from things related to human error. The firewall should keep unwanted traffic off of the system. We should encrypt our users data when transmitting it over the network. A user agreement is needed to make the user aware and liable for any situation. In addition to securing privacy, malware and virus scans could be done. This could all be sent through the platform and addressing it could be difficult, but there could be potential with warning users if something is triggered through a detection process.

## Security Requirements

Confidentiality and privacy security are of utmost importance. To ensure this could be maintained all users would be required to have a login and password to access any of their profiles. Behind the scenes there would also be a firewall set up in order to prevent malicious traffic from reaching. One’s personal data would also only be linked to that account unless something is shared. That is why the strength of a user's login and password detection system would have to be so sturdy. The website should prevent a malicious user from abusing the application and leaking sensitive data.

## Software Quality Attributes

By keeping a user-friendly interface in mind, the ideal characteristics would be efficiency, flexibility, simplicity, availability, usablitiy, portability, reliability, reusability, and maintainability. All of these play into making it a comfortable experience for the user. Factors such as maintainability and reliability are more for the developers since any issues experienced would have to be solved and with a simplistic and modular model they can maintain more easily . The customer must be satisfied, and by not accomplishing everything above it could make the system too difficult or unattractive to the user’s liking. Efficiency and flexibility should be easily accessed and easy to use for the user. Portability should be used in our system for a user to access our system through different platforms. Users want to access their information and send it from anywhere. It is crucial to develop everything around that.

# Other Requirements

We will need computing infrastructure for our email server and web interface client. All of the dependencies need to be documented so the application can be ported and hosted on many servers.

**Appendix A: Glossary**

*<Define all the terms necessary to properly interpret the SRS, including acronyms and abbreviations. You may wish to build a separate glossary that spans multiple projects or the entire organization, and just include terms specific to a single project in each SRS.>*

**Appendix B: Analysis Models**

*<Optionally, include any pertinent analysis models, such as data flow diagrams, class diagrams, state-transition diagrams, or entity-relationship diagrams*.>

**Appendix C: To Be Determined List**

*<Collect a numbered list of the TBD (to be determined) references that remain in the SRS so they can be tracked to closure.>*