GoForMail

Functional Specification

Student 1: Andre Rafael Cruz da Fonseca **Student ID:** 21460092

Student 2: Sean Albert Dagohoy **Student ID:** 21392656

Project Supervisor: Stephen Blott

CSC1097 - Final Year Project Dublin City University

Last Updated: November 3rd 2024

Table of Contents

1. Introduction

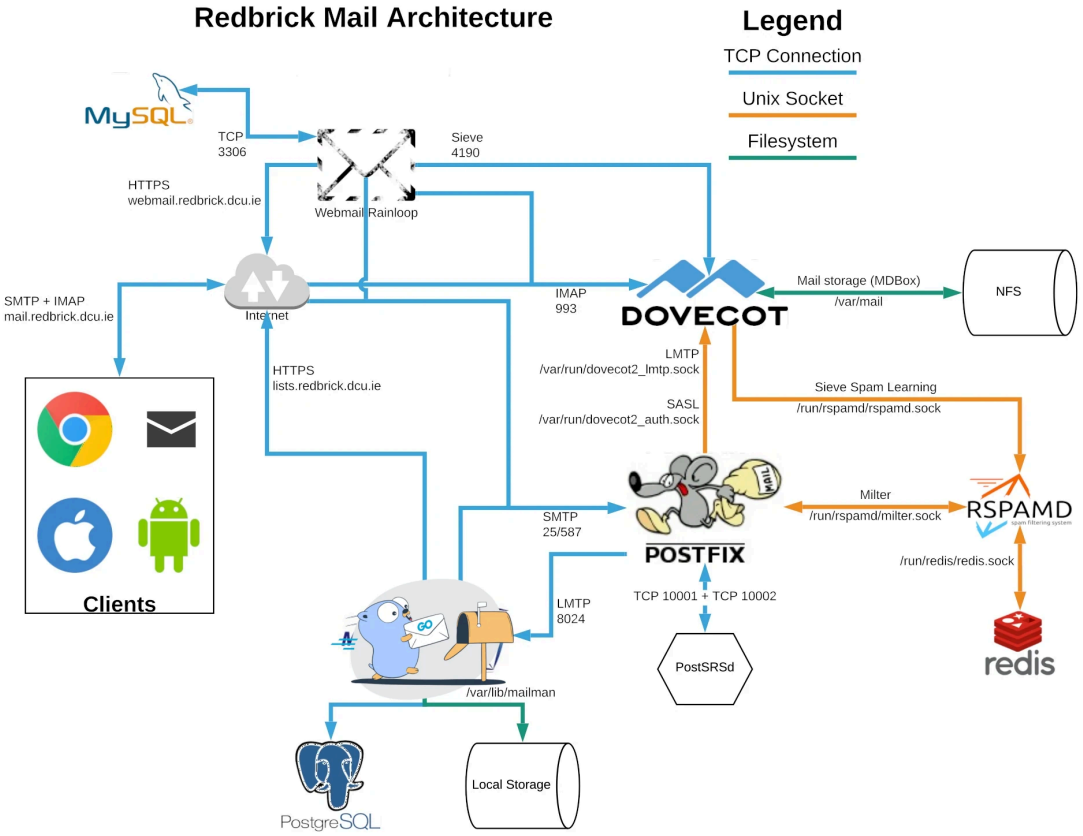
1.1 Overview

GoForMail is a Mailing List Manager, written in Go.

The project is similar to other applications such as GNU/Mailman but features a more modern interface. It was written to serve Redbrick, by connecting to their Postfix instance.

1.2 Business Context

GoForMail is a project aimed at replacing a component in Redbrick's systems and hence the main context will be for them to use it for free. Below is an image of how it fits into their tech stack.



However, there are some ways the system can be adapted in future for an increased customer base.

Pay to Use Software - The project could be released as a binary at a fixed cost or recurring subscription to other organisations who wish to release emails en masse. Companies may have many departments and locations with their own email needs which would benefit from GoForMail. Alternatively, it could be sold as a serverless service which runs on the cloud.

Free Open Source Software - The project could be released as open source software on a publicly accessible git repository. This would allow anyone to set up their own GoForMail server for their own needs. While this is usually the case for non-profit projects, it can also be used for profit. A product subscription could include support with setting up and maintaining the product. This would be similar to the way Redhat Linux operates.

1.3 Glossary

API - Stands for Application Programming Interface. They are the mechanisms that enable two software components to communicate to each other.

Bug - An undesired behaviour in a computer system.

Continuous Integration / Continuous Delivery (CI/CD) - A software development principle which aims to streamline development delivery through automated testing and deployment.

Database - A collection of structured information or data - usually stored electronically via servers.

Deployment - The activity which makes a software available to use. This usually involves running software on a server multiple people can access.

Go (GoLang) - A high programming language designed at Google. This is the language used to write GoForMail.

Hard-coded - A value written into the code, making it impossible to change by a user.

Headless - A system without a graphical user interface.

Host - The person or machine responsible for a software's deployment.

Interface - A program which allows a user to interact with the system.

Local Mail Transfer Protocol (LMTP) - A protocol (set of rules) which allows for different programs to transfer email data on a local machine.

Mail Transfer Agent (MTA) - A program which is responsible for receiving and sending emails.

Mailing List - A list of email addresses. Usually used for sending emails en-masse.

Postfix - A mail transfer agent. In this case, the one used by Redbrick.

Redbrick - DCU's Computer Networking Society.

REST - A type of API which provides a simple way for applications to communicate over HTTP (the web).

Salt - Random data fed into a hashing function in order to create a non-identifiable hash.

Server - A computer responsible for deploying and running outward facing programs.

SQL - SQL stands for “Structured Query Language”. It is the language used to store, retrieve and manipulate data in relational databases.

UI - Abbreviation for “User Interface”

Vulnerability - A weakness in a computer system which allows for bad actors to take nefarious action.

2. General Description

2.1 Product / System Functions

GoForMail aims to allow users to send emails en-masse in an easy manner. To do this, we provide a variety of functions in our app.

- **Mailing List Forwarding**: Users send an email to a mailing list. Upon receiving it, GoForMail will forward it to every email address associated with that list.

- **Mailing List Management**: Creation, deletion and editing of mailing lists. This increases flexibility in the app’s functionality as users can adapt their mailing lists to their changing needs.

- **Mailing List Locking**: The locking of mailing lists to only whitelisted members and/or manually approved emails. This is to avoid spam from unintended actors, especially to big mailing lists.

- **User Management**: Creation, deletion and permission updating of user accounts. This increases security in the management of the mailing lists since different users can be assigned different levels of power.

- **Audit Logging**: Logging of every action performed and its corresponding user. Useful in case of admin abuse for accountability purposes.

- **Mail Archiving**: Logging of every email processed by the application.

2.2 User Characteristics and Objectives

Main users of this system will be organisations with custom mail transfer agents seeking to send emails en-masse and hobbyists seeking to enhance the functionality of their self hosted mail transfer agent.

The targeted user for managing mailing lists and sending emails will be those in HR or secretary positions, hence not a lot of technical knowledge should be required. The user hosting the application, however, is expected to be quite knowledgeable in server hosting. This will most likely be the IT department of the organisation.

The system aims to have a user-friendly interface so that managing various mailing lists will be easier to do for the less technical users. This is achieved through a web UI that's clear and easy to use. A command line interface is also available however to aid hosts in cases where debugging is required.

2.3 Operational Scenarios

Below are a series of scenarios that can happen when the service is in use. Any scenarios in which an admin does an action, the action will be logged by the service.

Use Case 1	Sending Emails
Goal	Sender sends an email to the mailing list
Actors	Sender
Preconditions	Sender has an email account
Success End Condition	Email has been sent to the mailing list
Failed End Condition	Email was not sent to the mailing list
Main Flow	1. Sender composes an email through their email provider 2. Email is sent to the service 3. Email gets forwarded to the recipients of the mailing list 4. Email gets archived
Alternative Flows	If the email has been sent to a mailing list that requires admin approval for each email, it will be sent to a list of emails that requires approvals. Once an admin approves the email, it will be forwarded to the recipients.

Use Case 2	Email Approval
Goal	Admin approves or rejects an email
Actors	Admin
Preconditions	Admin has a profile and is signed in Email is sent to a mailing list that requires email approval
Success End Condition	Email is forwarded if approved, email does not get forwarded if rejected
Failed End Condition	Email was not forwarded if it was approved
Main Flow	1. Admin navigates to the list of emails that requires approvals 2. Admin reviews the email awaiting approval 3. Admin approves email 4. Email gets forwarded to the recipients 5. Email gets archived
Alternative Flows	Email gets rejected by an admin and does not get forwarded, however, email is still archived

Use Case 3	Mailing List Creation
Goal	Create a new mailing list
Actors	Admin
Preconditions	Admin has a profile and is signed in
Success End Condition	A new mailing list has been created
Failed End Condition	Mailing list was not created
Main Flow	1. Admin navigates to the interface for creating a new mailing list 2. Enters the email address of the new mailing list 3. Admin can add recipients to the mailing list
Alternative Flows	Admin creates a new mailing list through a command in the command line

Use Case 4	Mailing List Editing
Goal	Admins add or remove recipients from a mailing list
Actors	Admin
Preconditions	Admin is logged in
Success End Condition	Admin adds or removes recipients from an existing mailing list
Failed End Condition	Admin fails to add or remove recipients from the mailing list
Main Flow	1. Admin navigates to an existing mailing list 2. Admin can add a new recipient 3. Admin can remove an existing recipient 4. Admin saves changes
Alternative Flows	

Use Case 5	Mailing List Deletion
Goal	Mailing list gets deleted
Actors	Admin
Preconditions	Admin is logged in
Success End Condition	Mailing list gets deleted
Failed End Condition	Mailing list is not deleted
Main Flow	1. Admin navigates to an existing mailing list 2. Admin navigates to delete the mailing list 3. Prompt shows asking if the admin wants to delete the mailing list 4. Admin deletes mailing list if they are sure
Alternative Flows	Admin decides they do not want to delete the mailing list when the prompt shows up. Action will not be logged and the mailing list does not get deleted.

2.4 Constraints

Below is the list of constraints that the project will face.

- **Time:** Deadlines will be one of the bigger obstacles the project will face. To combat this, appropriate time management and planning will be done throughout the development of this project. Jira will be used to keep track of progress of the project.

- **Mail Transfer Agent:** As Redbrick is our target audience for this project, we will be using postfix as the mail transfer agent to do testing as this is what they use. An investigation will be necessary to know if further work must be done to make it compatible with other mail transfer agents.

- **Storage:** Storage can potentially be a problem for the service. As the service gets used, more storage needed will be necessary to keep logs, email content, mailing list information, etc. For testing, we will be using one of our own servers which should be satisfactory for its purpose.

- **Knowledge:** Various research will need to be done to complete this project. The Go programming language and mail servers are examples of research that will need to be done as we have never worked with either.

3. Functional Requirements

3.1 Mailing List Forwarding

Once an email has been received by GoForMail, it will forward it to every member of its corresponding mailing list. In the event that something goes wrong during this process, it is reported back to the user.

Criticality - This is the core functionality of the project as this is what a mailing list manager does. Therefore, it is extremely high priority that this functionality is implemented.

Issues - A mail transfer agent will be needed to receive and send emails. We will be using “Postfix” as the mail transfer agent for testing as this is what Redbrick uses. An investigation will be needed to understand whether we need to do anything extra for other mail transfer agents.

Soft Dependencies -

- **Mailing List Management:** Without being able to manage mailing lists, emails can only be forwarded to a hard-coded list.

3.2 Mailing List Management

In order to accommodate any users' mailing list needs, GoForMail is able to create, edit and delete mailing lists. Without such a feature, only hard-coded mailing lists would be usable.

Criticality - This is a high priority requirement as it is one of the basic requirements Redbrick needs for a mailing list manager.

Issues - Storage may become an issue if there are many mailing lists created.

Hard Dependencies -

- **REST API / Command Line Interface:** It will be impossible to make any changes to the state of mailing lists unless we have an interface.

3.3 Mailing List Locking

GoForMail allows a mailing list to be 'locked'. When this is the case, any mail going to that mailing list will require a manual approval before it can be forwarded to its members. Alternatively, certain emails can be whitelisted in order to bypass the approval process. This is to prevent spam onto big mailing lists.

Criticality - This is a high priority requirement as it is one of the basic requirements Redbrick needs for a mailing list manager.

Issues - We will need to consider a scenario where a sender would like to schedule an email. Due to the nature of having each email be approved before forwarding, this scheduled email may not reach the recipients at the preferred time frame.

Hard Dependencies -

- **Mailing List Forwarding:** Emails need to be successfully sent to the mailing list service to be able to lock incoming emails.
- **Mailing List Management:** An existing mailing list must exist before an email can be sent to the service for checking.

3.4 REST API

The REST API is one of the 2 main interfaces through which GoForMail can be managed. Ideally a web UI would be used to interact with the REST API, however as a bare minimum this can be used for basic input/output. This feature also allows users to create their own UIs to interact with the app.

Criticality - This is a high priority feature as the system will require some sort of interface to interact with the system.

Issues - Detailed documentation will need to be provided for users to know how to use the API.

3.5 Web UI

GoForMail ships with a default web UI that connects to the REST API to provide a clean, easy to use interface for less technologically savvy users.

Criticality - This is a high priority feature. One of the big selling points of GoForMail is its clean user interface which allows ease of use to any user.

Issues - The UI may have issues displaying on screens of different formats. Extra care will be needed when ensuring compatibility across multiple devices.

Soft Dependencies -

- **REST API:** It will be impossible to connect the web UI to the main app without the API.

3.6 User Management

User accounts can be created in GoForMail which allows different users to have different permission levels. This means some accounts may have only access to approve and reject emails, some may have access to also edit mailing lists but not create or delete them and some accounts may have access to everything.

Criticality - This is a medium priority feature. While it increases security, the app will work regardless of it and a similar level of security can be achieved by reducing the amount of people with access to the interface.

Issues - Password storage and security will need to be considered to ensure passwords cannot be used to compromise users' accounts on other services. Another issue is the complexity of permissions. If each individual has different permissions for each individual mailing list, the complexity will grow massively. The extent of the permission customisation is subject to how much implementation time we are able to get.

Hard Dependencies -

- **REST API / Command Line Interface:** It will be impossible to make any changes to the state of mailing lists unless we have an interface.

3.7 Command Line Interface

In order to more easily access the management options of GoForMail through a headless host, a command line interface is available. It features all the same functionality of the REST API but can be accessed through a headless environment.

Criticality - This is a medium priority feature. It is quite a useful feature in some cases, however it does not add any functionality exclusive to it.

Issues - Functionality could end up unsynced with the REST API if poorly implemented.

3.8 Audit Logging

For traceability, every significant action done in GoForMail (e.g. creating a mailing list) by a user gets logged.

Criticality - This feature has medium priority. While extremely useful for accountability, it is not required in most circumstances and will only be needed if someone does something they shouldn't.

Issues: Increased storage requirements may become an issue due to logs being created.

Hard Dependencies:

- **User Management:** For logging to be more descriptive, profiles need to exist as the action performed can be traced back to a user.

3.9 Feedback Emails

Whenever an email is sent to a mailing list, an email is sent back to the sender informing them of the status of their email. This may be that it was successfully sent, is pending approval, was successfully approved or was rejected.

Criticality - This feature has a low priority. While a nice quality of life for email senders, it does not provide any real new functionality to the program.

Issues: Feedback emails may be marked as spam due to always following the same layout.

Hard Dependencies:

- **Mailing List Forwarding:** It will be impossible to send an email back if the app is not listening for emails.

3.10 Scheduled Email Approvals

If an email to a locked list needs to be sent at a specific time, this can be used to approve an email ahead of time and have it be sent at the desired time.

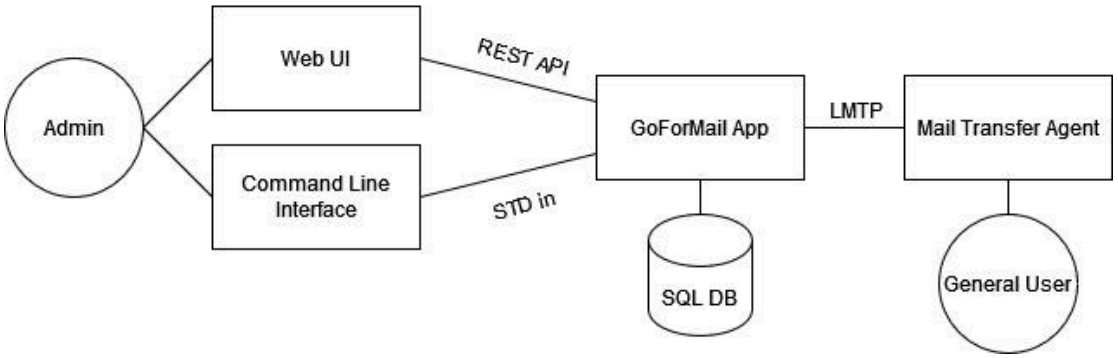
Criticality - This feature has a low priority. Its functionality is quite nice, however it can also be achieved through schedule sending the email using the email provider's scheduler on a whitelisted email address.

Issues: Scheduled times will be dependent on a timezone. If someone is unsure what timezone it will be sent they might schedule it for the wrong time. The timezone being used should be clear on UIs.

Hard Dependencies:

- **Mailing List Blocking:** The premise of this is that approvals can be scheduled. If emails were not blocked, they could just be scheduled with their email provider.

4. System Architecture



The above diagram shows the relationship between the different components of the system. All system components are shown as rectangles, users as circles and the database as a cylinder.

4.1 GoForMail App

The GoForMail application is the main component of the system. It links all other components and runs all the business logic. It is written in Go and consists of 3 layers:

- **Controller Layer**: A section of the code responsible for interacting with user inputs. This includes the REST API controller as well as the standard input controller.

- **Repository Layer**: A section of the code responsible for interacting with the database. This will ensure users have no direct access to the database.

- **Service Layer**: A section of the code responsible for the business logic of any operation received. This will keep the functions of the supported interfaces aligned and reduce what would otherwise be code duplication.

4.2 Mail Transfer Agent

The MTA is an external application not developed by us. GoForMail will communicate to it through the LMTP protocol in order to successfully send and receive mail. Our aim is to support any MTA, however we will be focusing on supporting Postfix as it is the one currently in use by Redbrick.

Any emails must go through the MTA before reaching GoForMail and return through the MTA before reaching any inboxes.

4.3 Web UI

The web UI will be used for user and mailing list management purposes. Our provided UI will interact with the main application through a REST API. However, due to the nature of this controller, hosts will be able to create their own UIs, so long as they avail of the same REST API.

4.3 Command Line Interface

The Command Line will be used for user and mailing list management purposes. This will be an out-of-the-box option for interacting with the management as the application will become interactive upon execution.

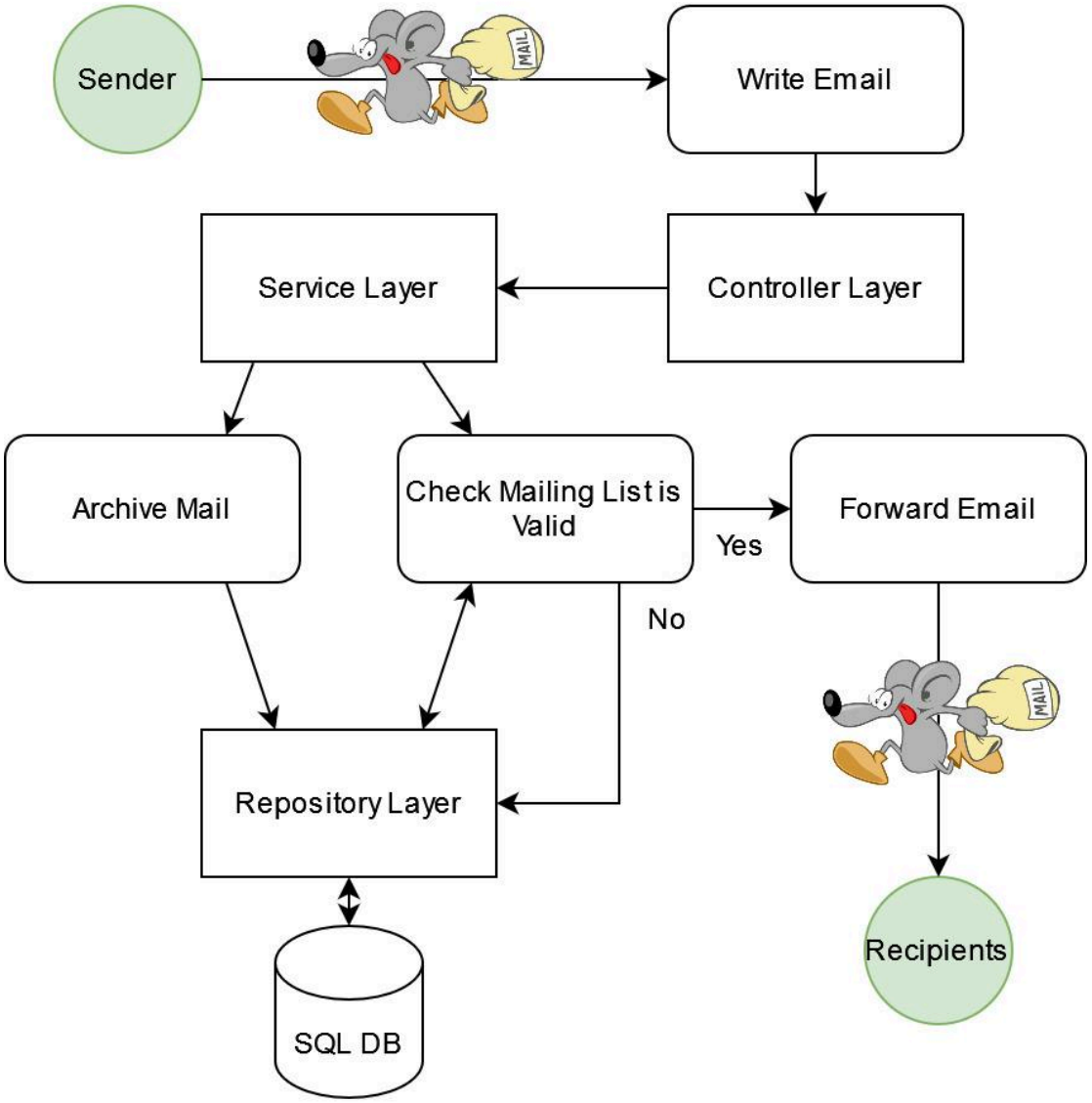
4.3 Database

The app's storage will be managed through a SQL database. This is to allow for scalability of the app and compatibility with Redbrick's existing database. The content in the database will include account information, mailing list details, audit logs and mail archives.

5. High-Level Design

This section will contain a variety of diagrams followed by some additional context. These diagrams describe the different processes occurring in GoForMail and how they are linked to each layer of the program. In addition to the process shown, an audit log is created for every action taken.

5.1 Mail Forwarding



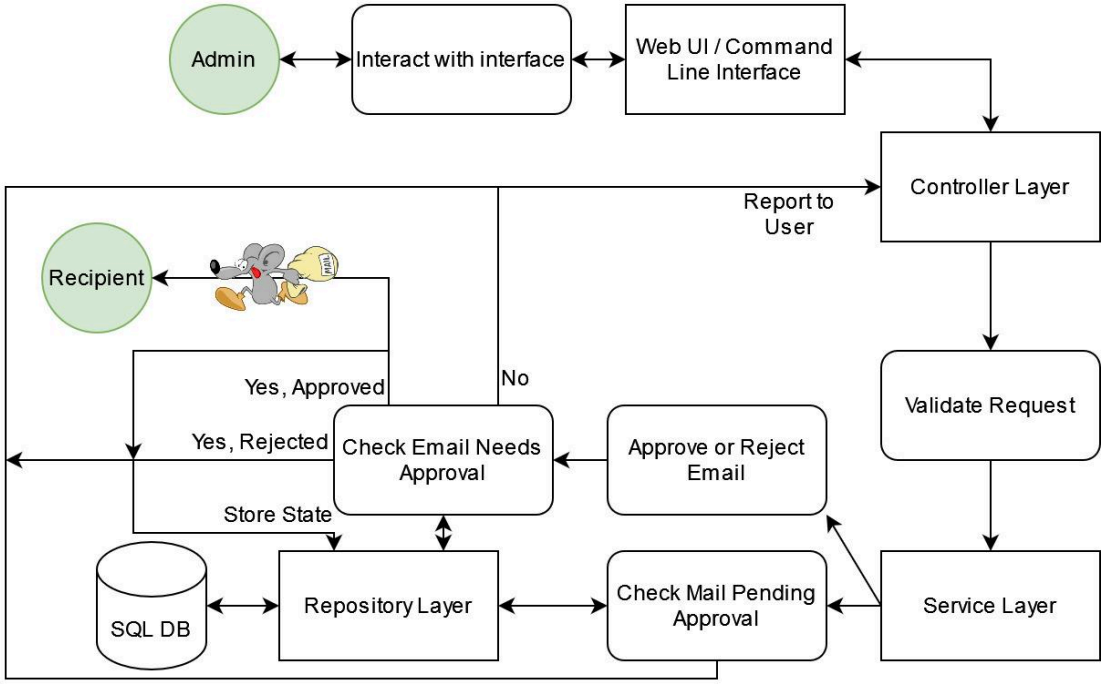
Trigger - User sends an email to a mailing list address. This is picked up by the MTA and routed to GoForMail, entering through the controller layer.

Controller - Here, the controller verifies that the LMTP request is valid and passes on any required information to the service layer.

Service - The service layer archives the email. This will always happen regardless of the validity of the mailing list. The request's desired mailing list is then checked to see if it exists and is unlocked/whitelisted. If both of these are true, the email gets routed back to the MTA to the list members. If one is not true, it will be saved in the database as such.

Repository - The repository layer only acts as a way to talk with the database. It is used to store the new email, as well as fetch information about its mailing list.

5.2 Mail Approval



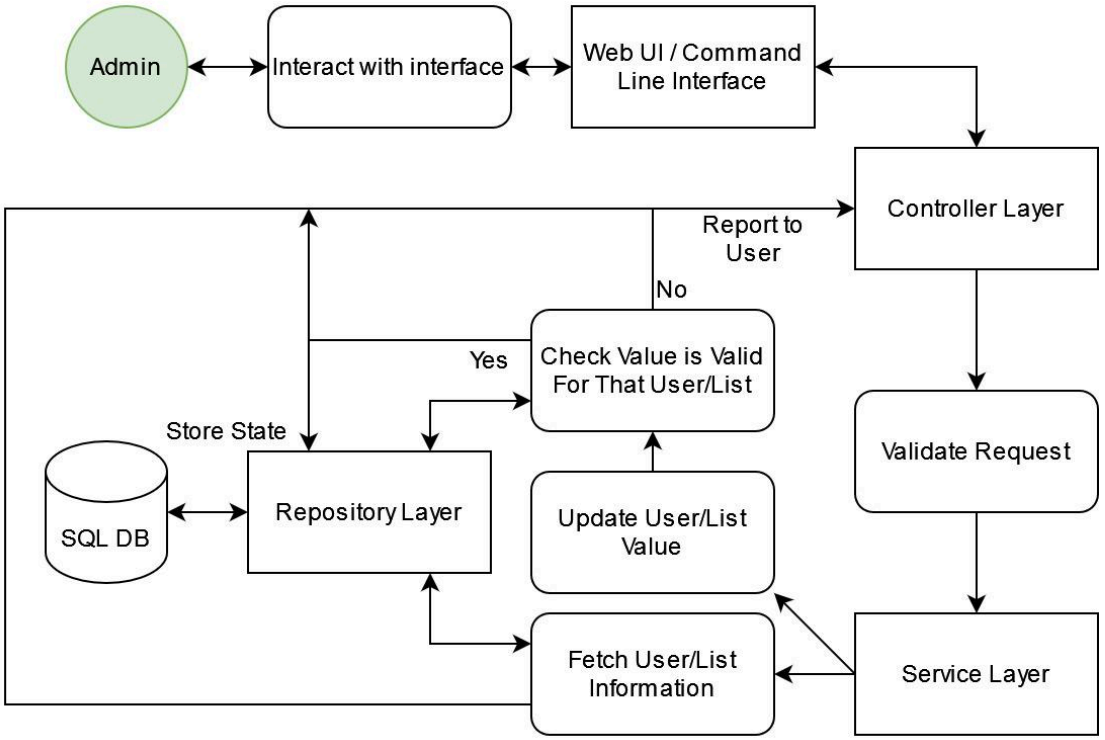
Trigger - Admin interacts with the app through either the web UI or command line interface. This reaches the main application through the controller layer.

Controller - Here, the controller verifies that the user's request is valid and passes on any required information to the service layer. It also reports any feedback for the actions performed back to the user once the action has been performed.

Service - The service layer acts as a middleman between the controller and repository when fetching information about the mail pending approval. It also updates the state of the mail in the repository layer after a decision is made and forwards any approved mail to the MTA.

Repository - The repository layer still only acts as a way to talk with the database. It is used to fetch information on emails' current approval status.

5.3 User/List Management

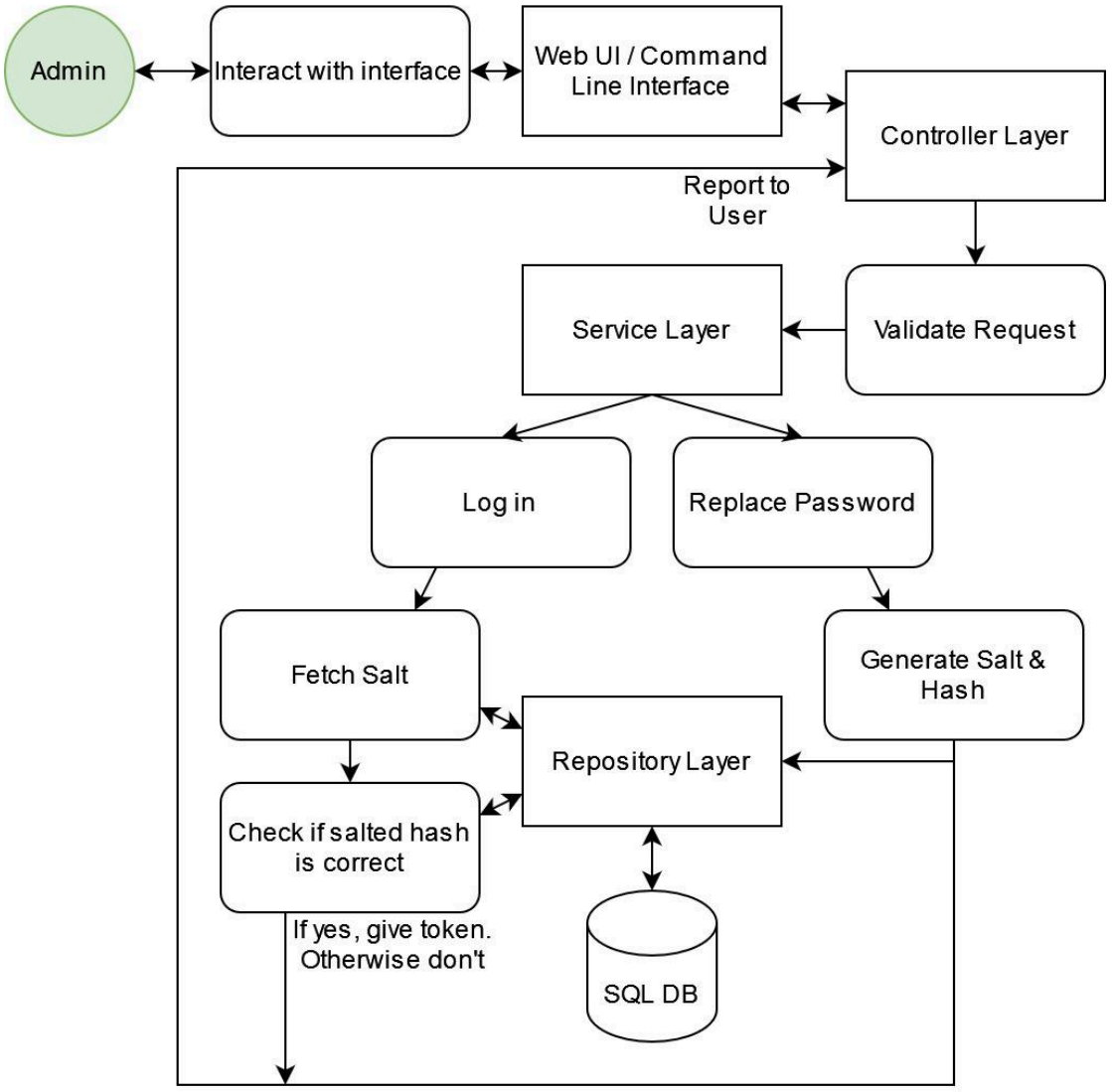


Trigger & Controller - Same as section 5.2

Service - The service layer once again acts as a middleman between the controller and repository when fetching information about the users and mailing lists. It also updates the state of the user or mailing list in question in the repository layer.

Repository - The repository layer still only acts as a way to talk with the database. It is used to fetch and update information on users and lists.

5.4 User Authentication

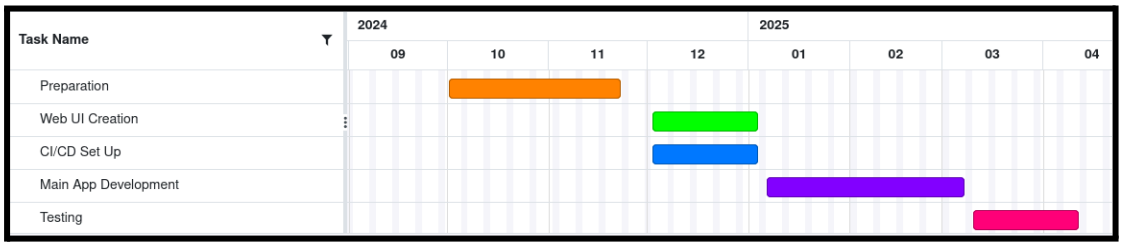


Trigger & Controller - Same as section 5.2

Service - The service layer is responsible for generating salts and password hashes when a password is to be created and storing those in the repository layer. It is also responsible for ensuring that a password generates the correct hash using the salt when a user attempts to log in.

Repository - The repository layer still only acts as a way to talk with the database. It is used to fetch and update information on passwords and salts.

6. Preliminary Schedule



The gantt chart above displays a rough outline for the time allocated for each stage of development. The stages can be found on the left of the image, while the time can be found at the top.

The whole timeline spans from the start of October when the proposal document was initially drafted to the start of April when all compatibility testing with Redbrick should be concluded. A short description of each stage is present below.

Preparation - Writing the required documents for the project's approval, researching the technologies used and familiarising ourselves with the Go Language.

Web UI Creation - Designing and writing the base of the web UI. The UI will be tweaked throughout the rest of the project to fix bugs and implement new functionality, however the design should not change much.

CI/CD Set Up - Setting up gitlab pipelines for code linting, unit test running, vulnerability checking and deployment. An automatic deployment to Andre's server will be attempted, however if this cannot be achieved then a "ready to use" executable will be provided for manual deployment.

Main App Development - Coding the core functionality of the application. This will also include writing unit and integration tests as new features are implemented.

Testing - After all features have been successfully implemented, some time will be used for compatibility testing with different MTAs and databases, as well as Redbrick's existing infrastructure.

7. Appendices

7.1 External Resources

Images used in the document:

- https://mailtrap.io/blog/golang-send-email/

Tools used for diagrams:
- https://www.onlinegantt.com/#/gantt
- https://app.diagrams.net/

7.2 Research Materials

Tool research:
- https://go.dev/doc/
- https://nextjs.org/docs
- https://docs.gitlab.com/ee/ci/

Similar applications:
- https://list.org/
- https://sendy.co/
- http://mlmmj.org/