



GoForMail

User Manual

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: May 2nd 2025

Table of Contents

Table of Contents.....	2
1. Introduction.....	3
1.1 Overview.....	3
1.2 Functions.....	4
1.2.1 Mailing List Forwarding:.....	4
1.2.2 Mailing List Management:.....	4
1.2.3 Mailing List Locking:.....	4
1.2.4 User Management:.....	4
1.2.5 User Management Permissions:.....	4
1.2.6 Mailing Archiving:.....	5
2. Installation.....	5
2.1 Docker Deployment (easy).....	5
3. Web UI Usage.....	5
3.1 General Overview.....	5
3.1.1 Logging in.....	6
3.1.2 Dashboards.....	7
3.2 Managing Users.....	7
3.2.1 Creating a user.....	7
3.2.2 Modifying a user.....	8
3.2.3 Deleting a user.....	9
3.3 Managing Lists.....	9
3.3.1 Creating a mailing list.....	10
3.3.2 Viewing A Mailing List's Details.....	10
3.3.3 Editing the list.....	11
3.3.4 Deleting the list.....	12
3.3.5 Editing the list's moderators.....	12
3.3.6 Editing the list's recipients.....	13
3.4 Emails.....	13
3.4.1 Viewing an archived email.....	13
3.4.2 Approving an email.....	14
4. REST API Usage.....	14
4.1 General Overview.....	14
4.2 Auth.....	15
4.2.1 Logging in.....	15
4.2.2 Validate Token.....	16
4.3 Users.....	16

4.3.1 Get all users.....	16
4.3.2 /user/.....	16
4.4 Managing Lists.....	17
4.4.1 Get all mailing lists.....	17
4.4.2 /list/.....	17
4.5 Emails.....	18
4.5.1 Get Emails.....	18
4.5.2 Get Single Email.....	18
4.5.3 Approve an email.....	18
5. CLI Usage.....	19
5.1 General Overview.....	19
5.1.1 Executing commands.....	19
5.1 Managing Users.....	19
5.1.1 Creating Users.....	19
5.1.2 Updating Users.....	20
5.1.3 Getting Users.....	20
5.1.4 Deleting Users.....	21
5.2 Managing Lists.....	21
5.2.1 Creating Lists.....	21
5.2.2 Updating Lists.....	21
5.2.3 Getting Lists.....	22
5.2.4 Deleting Lists.....	22
5.3 Emails.....	22
5.3.1 Getting Lists.....	22
5.3.2 Approving Emails.....	23
6. Email Usage.....	23
6.1 Approved Sender.....	23
6.2 Unapproved Sender.....	23

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 Functions

GoForMail has the following functions:

1.2.1 Mailing List Forwarding:

- Users can send an email to a mailing list in which GoForMail will forward that email to every email address associated with that list

1.2.2 Mailing List Management:

- Mailing lists can be created, deleted, and modified. This is done through the web interface or the CLI. This include being able to add or remove recipients, being able able to add or remove approved senders, and being able to add or remove mailing list moderators
- Mailing lists have a list of moderators. These moderators are users who are allowed to modify the mailing list without requiring the “manage mailing list” permission.
- to add or remove moderators of the mailing list, and being able to add or remove approved senders

1.2.3 Mailing List Locking:

- Mailing list locking can prevent spam from unapproved actors to send emails to the mailing list. This is done by having whitelisted members only be able to send emails to the mailing list or if a user is not whitelisted, their email needs to be approved before the email is sent to the recipients.

1.2.4 User Management:

- Users can be created, deleted, and modified. This is done through the web interface or the CLI.

1.2.5 User Management Permissions:

- The purpose of users for the service is for mailing list management as users require the correct authorizations for list management. The following permissions are:
 - **Create Mailing List:** The permission to be able to create a mailing list. A user can only create a mailing list if they have this permission. Users are automatically added as a moderator for the newly created mailing list which means they are able to modify it as well.
 - **Manage Mailing List:** The permissions to be able to modify mailing lists. A user can modify all existing mailing lists without the user needing to be a moderator for the mailing list.

- **Create User:** The permission to be able to create a new user. As there is no way of signing up to be able to use the mailing list manager, an existing user with the appropriate permission must be the one to create a new user. The purpose for this is for added security as only the intended actors should be allowed to access the service. User creation includes setting a user's email and password, and setting their permissions.
- **Manage User:** The permission to be able to manage users. Users with this permission can remove users from the service and also be able to modify another user's permissions.
- **Admin:** This permission enables the user to do all operations regarding the permissions above

1.2.6 Mailing Archiving:

- Emails sent to mailing lists are logged for viewing

2. Installation

2.1 Docker Deployment (easy)

1. Set up postfix with configs below
2. Clone the Repo
3. Run "docker compose up -d"

Sample postfix configs are provided. Please update them to suit your needs.

REMEMBER TO RUN POSTMAP ON YOUR RECIPIENTS MAP

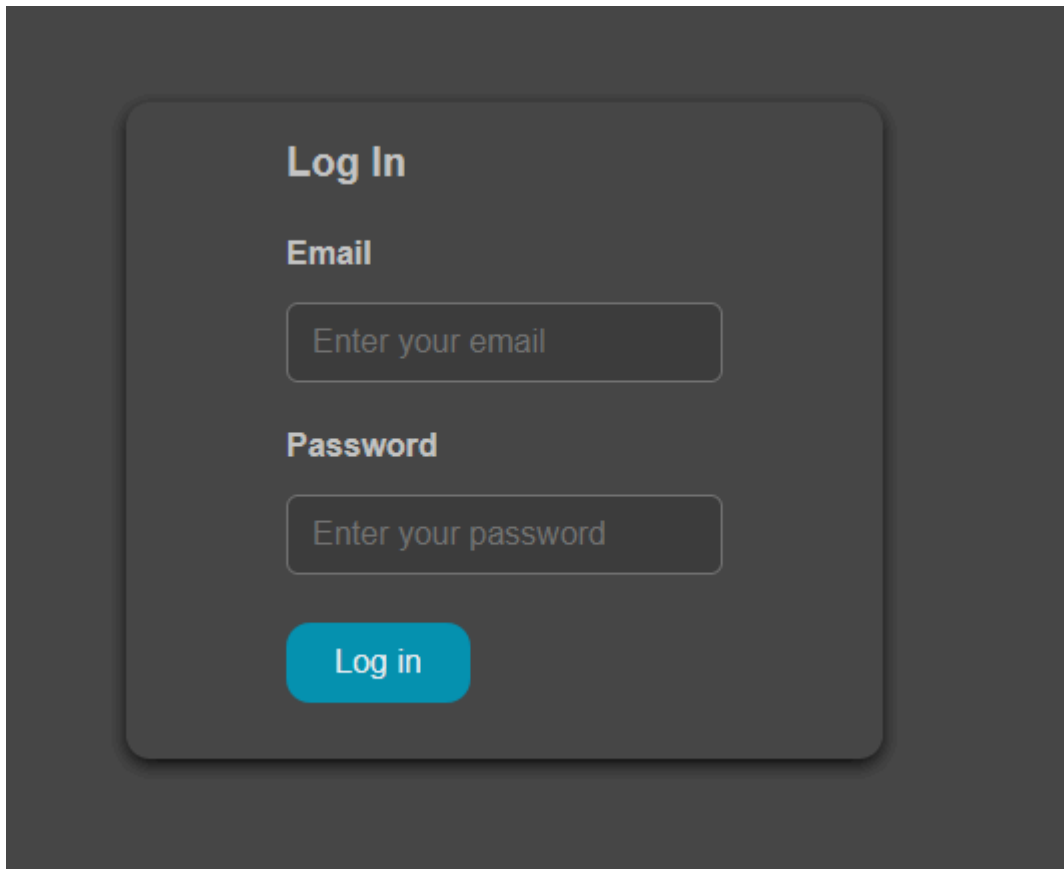
I was super low on time, any questions don't hesitate to email
andre.fonseca3@mail.dcu.ie

3. Web UI Usage

3.1 General Overview

One way to interact with the app is through the web ui. The web ui is accessible through localhost by default and the port for it is the http port within the configs.cf file. For example, assuming we have the http port set to 8000, then the uri for the web ui is localhost:8000/ui/

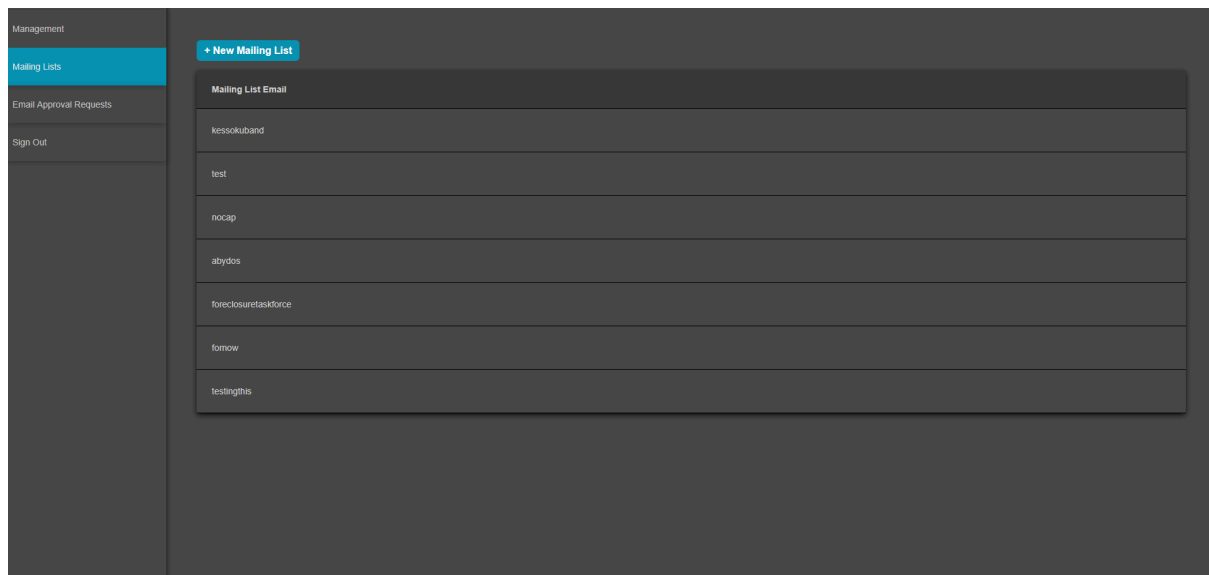
3.1.1 Logging in



The image shows a login form on a dark background. The form is a light gray rounded rectangle. At the top, it says "Log In" in bold. Below that is the "Email" label, followed by a text input field with the placeholder "Enter your email". Then is the "Password" label, followed by a text input field with the placeholder "Enter your password". At the bottom is a blue "Log in" button.

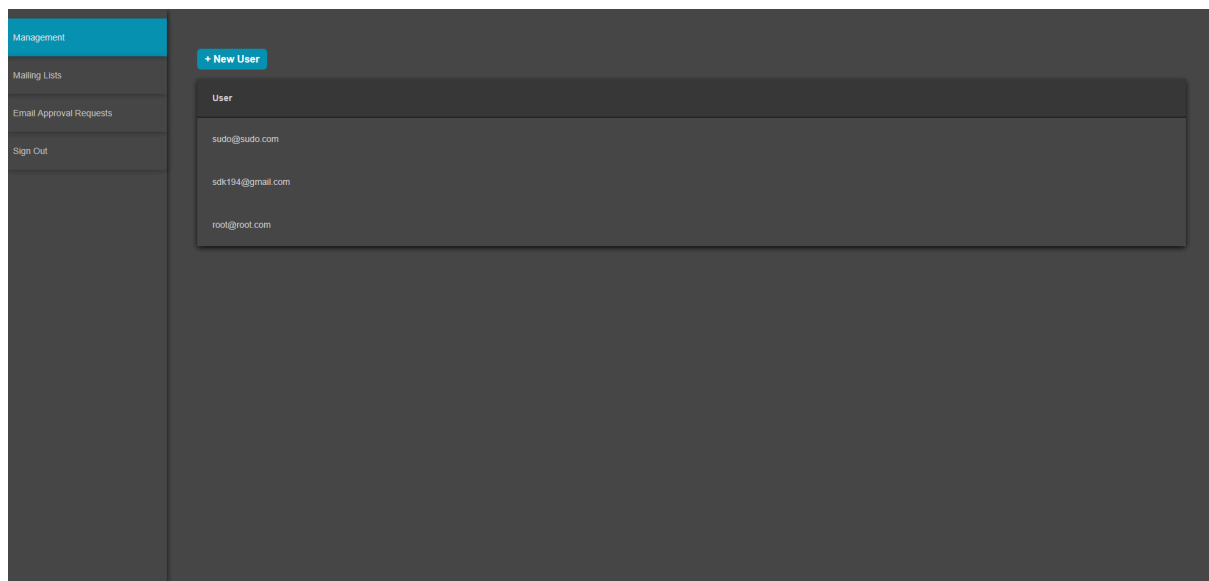
If the user has not logged in or their session token has expired, they are first brought to the login page in which they log in with their credentials. If they do not have an account, they need to contact a known admin with the correct permissions for account creation

3.1.2 Dashboards



Once the user is logged in, they will be sent to the mailing lists page which is where they can view all of the mailing lists

3.2 Managing Users



To view the users within the app, the user clicks on the Management tab on the navbar. Here, a user can edit one of the existing users or create a new user.

3.2.1 Creating a user

To create a user, the user clicks on the “+ new user” button

Create a user

Email Address

Password

Permissions

Admin	<input type="checkbox"/>
Create Mailing Lists	<input type="checkbox"/>
Manage Mailing Lists	<input type="checkbox"/>
Create Users	<input type="checkbox"/>
Manage Users	<input type="checkbox"/>

[Create User](#)

Here, a user needs to fill in the new user's email address and password. A user can have no permissions which means they'd only be able to view lists and manage lists that they are a moderator of. Once done, the user clicks "Create User" and will be taken back to the management page where they can view all users. Note that to be able to create a user, the current user must have the "Create Users" permission.

3.2.2 Modifying a user

To modify an existing user, the user clicks on one of the users from the list of users within the management page.

Edit User

Email Address

Permissions

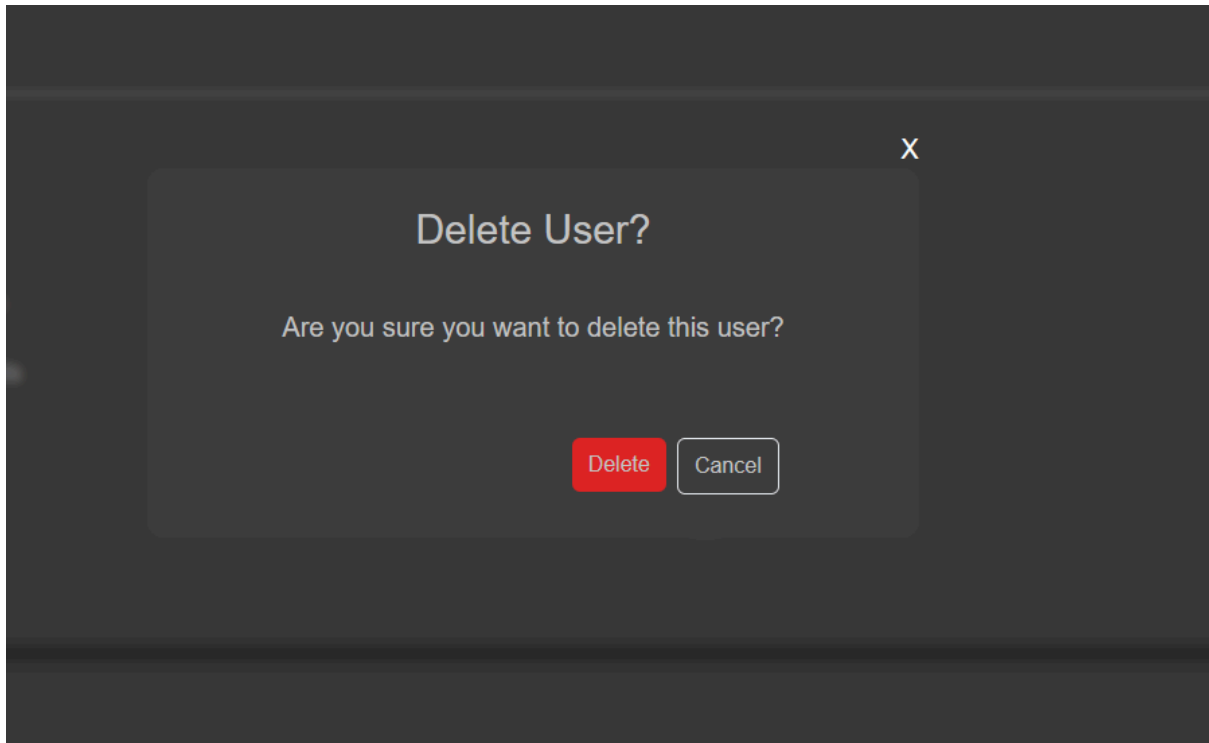
Admin	<input type="checkbox"/>
Create Mailing Lists	<input type="checkbox"/>
Manage Mailing Lists	<input type="checkbox"/>
Create Users	<input type="checkbox"/>
Manage Users	<input type="checkbox"/>

[Edit User](#) [Delete User](#)

Here they can view the user's information regarding their permissions. The only fields that can be modified are the permissions. The user clicks "Edit User" if the user wants to modify the other user's permissions. Note that the current user needs to have the "Manage Users" permission to be able to modify a user's permissions.

3.2.3 Deleting a user

To delete a user, it is done through the same page as modifying a user. Right beside the “Edit User” button is the “Delete User” button. Once clicked, a modal will pop up asking if they are sure they want to delete the user.



Pressing “Cancel” will bring the user back to the modifying user screen, however, if “Delete” has been pressed, the user will be sent back to the management page and will also delete the user. Note that for deleting users, the current user must have the “Manage Users” permission.

3.3 Managing Lists

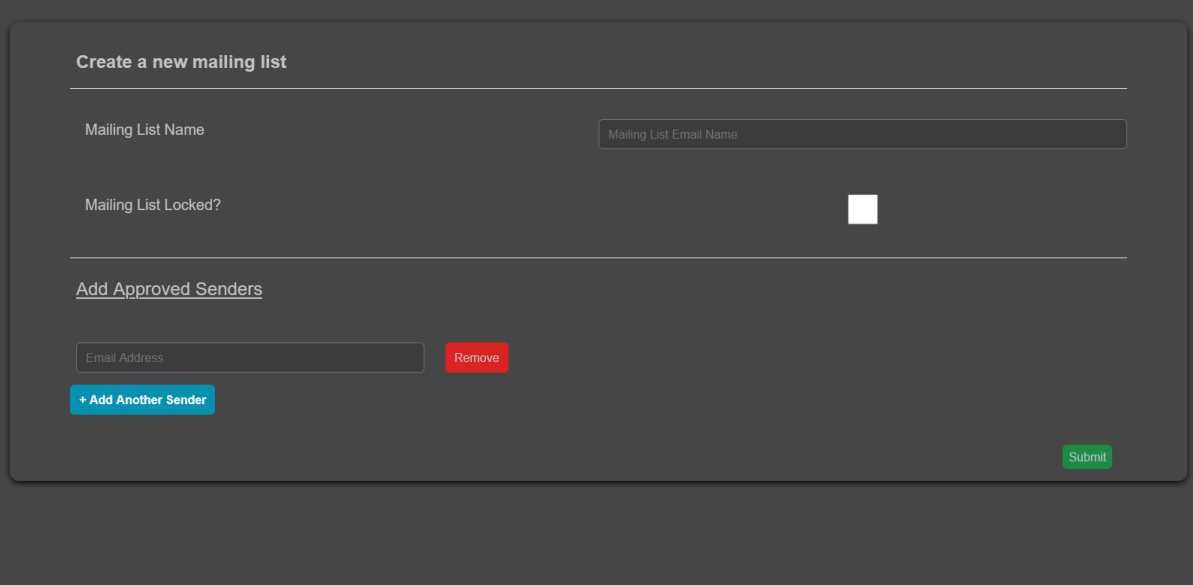
To view and manage lists, the user clicks on the “Mailing Lists” tab from the navbar



Here they can view all of the mailing lists

3.3.1 Creating a mailing list

To create a mailing list the user clicks on the “+ New Mailing List” button from the mailingList page

The screenshot shows a dark-themed form titled "Create a new mailing list". It contains two input fields: "Mailing List Name" and "Mailing List Email Name". Below these is a checkbox labeled "Mailing List Locked?". A horizontal line separates the top section from the "Add Approved Senders" section. This section includes an "Email Address" input field, a red "Remove" button, and a blue "+ Add Another Sender" button. A green "Submit" button is located at the bottom right of the form.

Here they are presented with the form as shown above. The user inputs the mailing list's name (it must have no spaces and must not include the domain of the email eg “[y@domain.tld](#)” would not be accepted, it must just be “y”). The user can also toggle whether the mailing list will be “locked” (see Functions section for what it does). The user can also input approved senders, however, if the user does not want to add any approved senders, they remove the input by clicking “Remove”. The user can also add more senders by clicking the “+ Add Another Sender” button. All sender fields must be in a valid email address format. Once the user is done creating the mailing list, they press the “Submit” button which will send them to the page for viewing the newly created mailing list's details. Note that a user must have the “Create Mailing Lists” permission to be able to create a mailing list.

3.3.2 Viewing A Mailing List's Details

The user can view a mailing list's details by either clicking on one of the mailing lists within the mailing lists page or when they create a new mailing list, they get sent to the page to view the newly created list's details

Management
Mailing Lists
Email Approval Requests
Sign Out

newlist

Manage Moderators
Manage List

From	Date
No Data to show	

Manage Recipients

Recipients
No Data to Show

The above table shows the lists of emails the mailing list has received and sent to its recipients before and the table below it is the table of the recipients of the mailing list

3.3.3 Editing the list

To edit the list, the user clicks on the “Manage List” button from the single mailing list view page.

Delete Mailing List

Mailing List Name

Mailing List Locked?
☒

Approved Senders

Remove

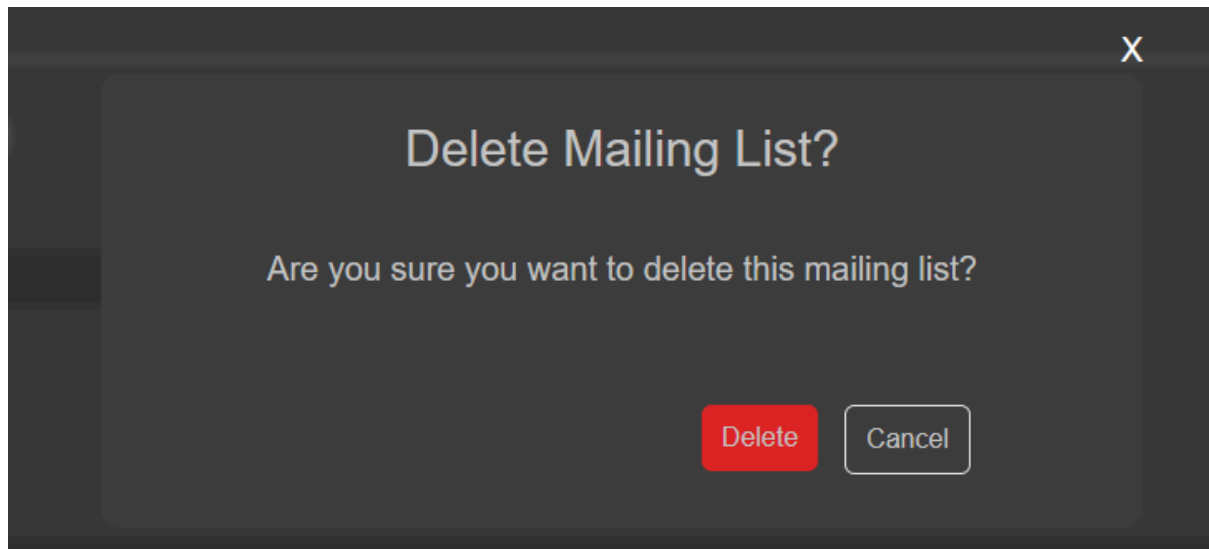
+ Add Another Sender

Submit

Here they can modify whether the list will be locked and also modify the approved senders. Once the user is finished editing, they click on the “Submit” button which will edit the list and send them back to the single list view page. Note that the user needs the “Manage Mailing List” permission or be a moderator of the list to be able to edit the list

3.3.4 Deleting the list

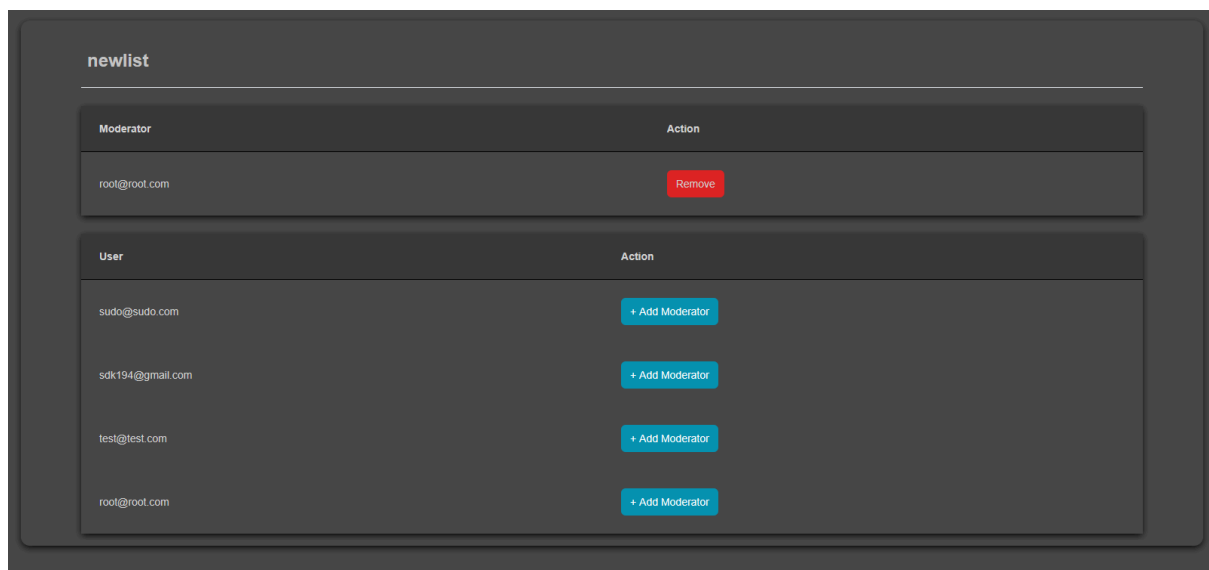
To delete the list, the user clicks on “Delete Mailing List” from the edit list page. A modal will pop up asking if they are sure to delete the list



If the user clicks “Cancel” they’ll be brought back to the edit list page. If the user clicks “Delete”, the user will be sent back to the list of mailing lists page with the mailing list deleted. Note that for a user to be able to delete a list, they must have the “Manage Moderators” permission or be a moderator for the list.

3.3.5 Editing the list’s moderators

To edit the list’s moderators, the user clicks on the “Manage Moderators” button from the single mailing list view page.

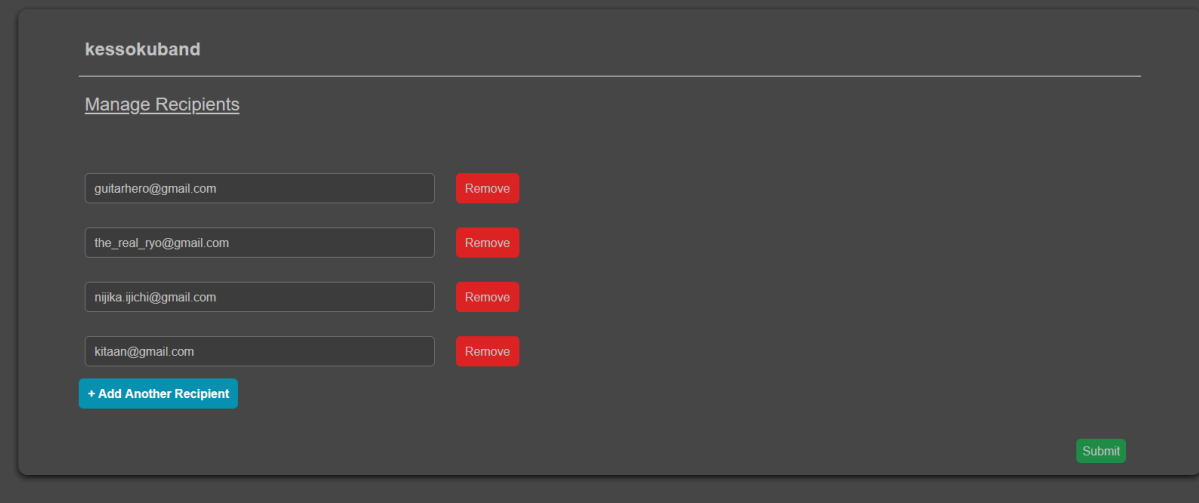


The above table is the current list of moderators and the table below is the list of users within the service. To remove a moderator, the user clicks on the “Remove”

button from the moderators table and to add a moderator, the user clicks on the “+ Add Moderator” from the user table. Note that to be able to add/remove moderators, the current user must either be a moderator for the list or has the “Manage mailing lists” permission.

3.3.6 Editing the list’s recipients

To manage recipients, the user clicks on the “Manage Recipients” button from the single mailing list view page.



The screenshot shows a web interface for managing mailing list recipients. At the top, the mailing list name "kessokuband" is displayed. Below it, a link "Manage Recipients" is visible. The main area contains a list of four email addresses, each in a text input field with a corresponding "Remove" button to its right:

- guitarhero@gmail.com
- the_real_ryo@gmail.com
- nijika ijichi@gmail.com
- kitaan@gmail.com

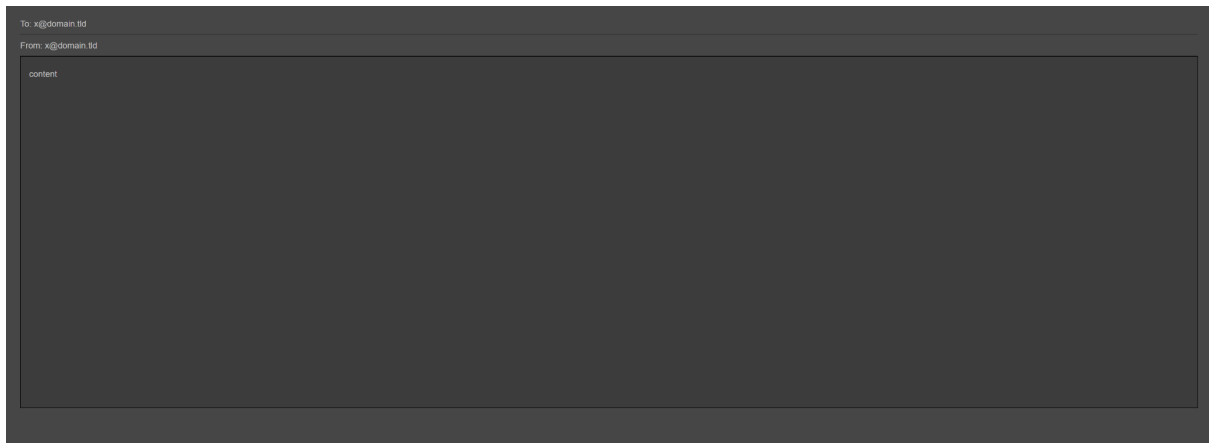
Below the list is a blue button labeled "+ Add Another Recipient". In the bottom right corner, there is a green "Submit" button.

Here the user can edit the email addresses of the recipients, remove recipients or add more recipients. Once done, the user clicks on the “Submit” button which will edit the recipients of the mailing list and then send them back to the single list view page. Note that the user must be a moderator of the mailing list or has the “Manage mailing lists” permission.

3.4 Emails

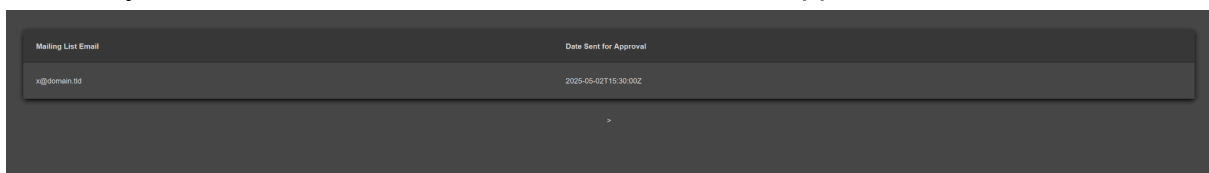
3.4.1 Viewing an archived email

To view an archived email, the user goes to a mailing list and clicks on one of the emails from the emails table.

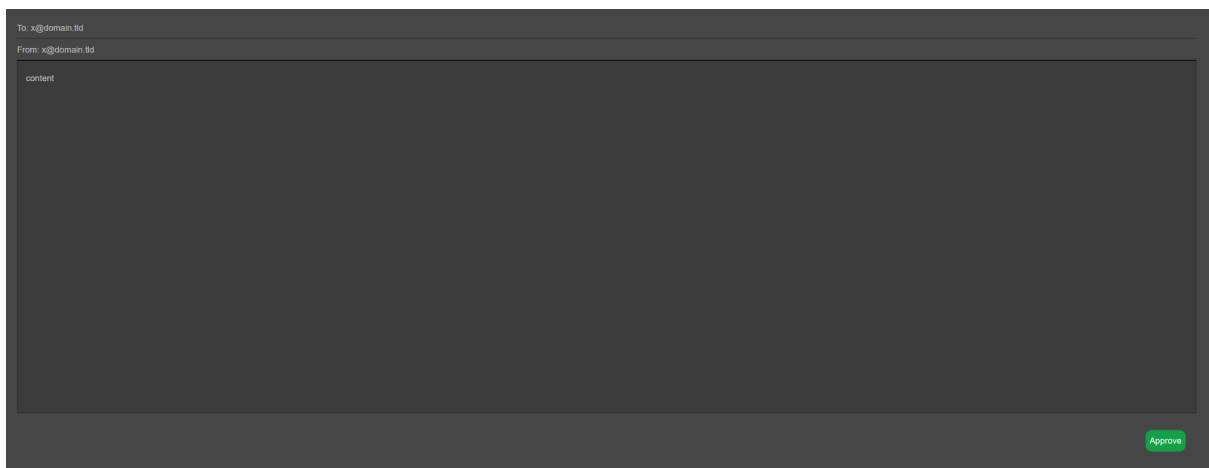


3.4.2 Approving an email

The user must first navigate to the “Email Approval Requests” tab within the navbar. Here they will be shown a list of emails that need to be approved.



Clicking on an email will send them to the following page



Here the user can view the email’s contents and then approve it they deem it fine to send to recipients. Note that a user must be a moderator of the mailing list or has the “Manage mailing lists” permission.

4. REST API Usage

4.1 General Overview

The REST API is used along with the web ui. The REST API uri is similar to the web ui's uri but with the /api rather than /ui eg localhost:8000/api/

Note: All endpoints except the login endpoint requires the Authorization header with a user's session token

Example endpoint: localhost:8000/api/users/

```
const response :Response = await fetch(url, {
  method: "GET",
  headers: {
    "Authorization": `Bearer ${sessionToken}`
  }
})
return await response.json()
}
```

2xx responses will be laid out in a json format like:

```
{
  message: "successful fetch"
  data: {}
}
```

4.2 Auth

4.2.1 Logging in

To get a session token for logging in, the api to be called is the "/login/" endpoint and the request is a "POST".

```
const response :Response = await fetch(url, {
  method: "POST",
  body: JSON.stringify({
    Email: email,
    Password: password,
  }),
  headers: {
    "Content-Type": "application/json"
  }
})
```

The screenshot above shows what the body must have. If the body is valid, a session token is given to use for all other endpoints.

4.2.2 Validate Token

The session token a user gets given will expire within 72 hours. To check if the token is still valid, a “POST” request on the “/validateToken/” endpoint needs to be called.

```
const response :Response = await fetch(url, {  
  method: "POST",  
  headers: {  
    "Authorization": `Bearer ${token}`,  
  }  
})
```

The body is empty, however, the authorization header is added with the user’s current token. The response received from the request will tell if the token is still valid.

4.3 Users

4.3.1 Get all users

To get all users, a “GET” request is called on the “/users/” endpoint. The data within the response will be an array of users and their details.

4.3.2 /user/


```
func (ctrl Controller) addUserHandlers() {
    ctrl.mux.HandleFunc("/api/user/", func(w http.ResponseWriter, r *http.Request) {
        if r.URL.Path != "/api/user/" {
            handleUnknownMethod(w, r)
            return
        }
        switch r.Method {
        case "GET":
            ctrl.getUser(w, r)
        case "POST":
            ctrl.postUser(w, r)
        case "PATCH":
            ctrl.patchUser(w, r)
        case "DELETE":
            ctrl.deleteUser(w, r)
        }
    })
}
```

All /user/ endpoints except for “POST” will also need an id query parameter. This id is associated with the queried user’s id.

“GET”: This will get a single user’s details

“POST”: This will create a new user

“PATCH”: This will edit an existing user’s details

“DELETE”: This will delete an existing user

4.4 Managing Lists

4.4.1 Get all mailing lists

To get all lists, a “GET” request is called on the “/lists/” endpoint. The data within the response will contain an array of mailing lists

4.4.2 /list/

```
func (ctrl Controller) addListHandlers() {
    ctrl.mux.HandleFunc("/api/list/", func(w http.ResponseWriter, r *http.Request) {
        if r.URL.Path != "/api/list/" {
            handleUnknownMethod(w, r)
            return
        }
        switch r.Method {
        case "GET":
            ctrl.getList(w, r)
        case "POST":
            ctrl.postList(w, r)
        case "PATCH":
            ctrl.patchList(w, r)
        case "DELETE":
            ctrl.deleteList(w, r)
        }
    })
}
```

All /list/ endpoints except for “POST” will also need an id query parameter. This id is associated with the queried list’s id.

“GET”: This will get a single list’s details

“POST”: This will create a new list

“PATCH”: This will edit an existing list’s details

“DELETE”: This will delete an existing list

4.5 Emails

4.5.1 Get Emails

To get emails, the “/emails/” endpoint is used with a “POST” request. This post request is necessary as emails can be filtered based on the post request’s body:

List - An integer. Setting this will make it so that only emails from that specific list will be fetched

Offset - An integer symbolising to start fetching at the (offset)th most recent email. For example, if you retrieved the latest 10 emails and want to retrieve the next 10, select an offset of 10.

Archived - A boolean. Setting this to true would mean only emails that have been sent to recipients will be fetched

Exhausted - A boolean. Setting this to true would mean only emails that has failed to be sent to recipients will be fetched

Pending Approval - A boolean. Setting this to true would mean only emails that are awaiting to be approved by a moderator will be fetched

4.5.2 Get Single Email

To get a single email, the “/email/” endpoint is used with an id query parameter and a “GET” request. This id is the email’s id.

4.5.3 Approve an email

To approve an email, the “/emails/approve” endpoint is used with an id query parameter and a “POST” request. This id is the email’s id

5. CLI Usage

5.1 General Overview

5.1.1 Executing commands

A way to interact with the application through the terminal is through its CLI. The CLI follows the following pattern:

```
goformail <action> <entity> <flags>
```

The below sections go more in depth about all of these variables for different commands.

5.1.2 Flags

Flags can be 3 types- **value** flags, **list** flags and **boolean** flags.

Value flags can be called with a singular value, as shown below:

```
goformail <action> <entity> --email=<email>
```

List flags can be called with multiple values. There's 2 ways to do this as shown below:

```
goformail <action> <entity> --permission=ADMIN,CRT_LIST
```

or

```
goformail <action> <entity> --permission=ADMIN  
--permission=CRT_LIST
```

Boolean flags are never called with a value. Their presence signals everything needed. This is shown below:

```
goformail <action> <entity> --permission=ADMIN  
--permission=CRT_LIST
```

5.1 Managing Users

5.1.1 Creating Users

Users can be created through the following command:

```
goformail create user --email=<email> --password=<password>
```

The command will not work without the **email** and **password** flags. However, there is an additional **permission** flag. This flag can be stacked for multiple permissions like so:

```
goformail create user --email=<email> --password=<password>
--permission=ADMIN,CRT_LIST
```

The following values are valid for each field:

Email- Any email (must be the full email)

Password- Any non empty value

Permission- Any of the following: *ADMIN, CRT_LIST, MOD_LIST, CRT_USER, MOD_USER*

5.1.2 Updating Users

Users can be updated through the following command:

```
goformail update user <id> --email=<email>
--permission=ADMIN,CRT_LIST
```

The command will also work without one or both of the **email** and **password** flags. Only whichever flags are set, update.

The following values are valid for each field:

Id- Any number corresponding to a user's id

Email- Any email (must be the full email)

Permission- Any of the following: *ADMIN, CRT_LIST, MOD_LIST, CRT_USER, MOD_USER*

5.1.3 Getting Users

The following command allows for retrieving a specific user, where id represents a valid user's id:

```
goformail get user <id>
```

However, it is also possible to retrieve all users, without need for ids, through the following command:

```
goformail get users
```

5.1.4 Deleting Users

The following command allows for deleting a specific user, where id represents the corresponding user's id:

```
goformail delete user <id>
```

5.2 Managing Lists

5.2.1 Creating Lists

Lists can be created through the following command:

```
goformail create user --name=<name>
```

The command will not work without the **name** flag. However, there are a few additional flags. These can be seen below:

```
goformail create list --name=<name> --recipient=<rcpts...>  
--mod=<mod_ids...> --approved=<senders...> --locked=true
```

The following values are valid for each field:

Name- Any email without including the domain (*ie. no-reply of no-reply@dcu.ie*)

Recipient- A list of email addresses (including the domain)

Mod- A list of ids for mods

Approved- A list of email addresses for approved sender (including the domain)

Locked- Either of the following: *true, false*

5.2.2 Updating Lists

Lists can be updated through the following command:

```
goformail update list <id> --name=<name> --recipient=<rcpts...>  
--mod=<mod_ids...> --approved=<senders...> --locked=true
```

The command will also work without any of these flags. Only whichever flags are set, update.

The following values are valid for each field:

Name- Any email without including the domain (*ie. no-reply of no-reply@dcu.ie*)

Recipient- A list of email addresses (including the domain)

Mod- A list of ids for mods

Approved- A list of email addresses for approved sender (including the domain)

Locked- Either of the following: *true*, *false*

5.2.3 Getting Lists

The following command allows for retrieving a specific list, where id represents a valid list's id:

```
goformail get list <id>
```

However, it is also possible to retrieve all lists, without need for ids, through the following command:

```
goformail get lists
```

5.2.4 Deleting Lists

The following command allows for deleting a specific list, where id represents the corresponding user's id:

```
goformail delete list <id>
```

5.3 Emails

5.3.1 Getting Lists

The following command allows for retrieving all emails in batches:

```
goformail get emails
```

In order to get the next batch, **set the offset correspondingly**. By default, you get the latest emails, however setting an offset means you start at the (n)th most recent email.

For example, if you retrieved the latest 10 emails and want to retrieve the next 10, select an offset of 10.

```
goformail get emails --offset 10
```

Additionally, the command offers filters to remove any unwanted types of emails.

These can be used through the following flags, which can be stacked:

```
goformail get emails --list 1 --exhausted --pending -archived
```

The following are the effects for each flag:

Offset- Fetch emails starting at the (offset)th most recent email

List- Only retrieve emails for the corresponding list id

Exhausted- Only retrieve failed emails

Pending- Only retrieve emails which need to be approved

Archived- Only retrieve emails which have already been sent

5.3.2 Approving Emails

Emails can be approved, given their id, by the following command:

```
goformail approve email <id>
```

6. Email Usage

6.1 Approved Sender

An approved sender would like to send an email to the recipients of a mailing list. They create an email using an email service. The recipient of the sender's email will be the mailing list. Once they have finished writing their email, the email gets sent to the mailing list which will then be picked up by GoForMail's LMTP service for rerouting to the mailing list's recipients while also archiving the email within GoForMail's database.

6.2 Unapproved Sender

An unapproved sender would like to send an email to the recipients of a mailing list. They create an email using an email service. The recipient of the sender's email will be the mailing list. Once they have finished writing their email, the email gets sent to the mailing list which will then be picked up by GoForMail's LMTP service.

GoForMail will recognise that the sender is not one of the approved senders of the mailing list and so will hold off on sending the email to the mailing list's recipients until it gets approved by a moderator of the mailing list. If the email gets approved, GoForMail will then queue the email to postfix with the mailing list's recipients.