Name	ID
Mohamed Abdelrahman Anwar	20011634
Hussein Mohamed Mansour	20010499
Amr Ahmed Abd ElAzim	20011037
Hussien Khaled Hussien Elsaid	20010494

Programming 2

Assignment 4 Report

Problem Statement:

The aim of the project is to implement the basic functionalities of a mail server, including the manipulation of mails, attachments, contacts and folders.

You should provide the following requirements:

- designing folders for emails such as: inbox, draft, sent, trash and custom folders.
- design compose section for sending emails.
- manipulations on emails such as searching and sorting based on different attributes (date, sender, receivers, importance, subject, body and attachments).
- users should be able to add, delete and view attachments.
- Users should be able to add, edit and delete contacts.
- Users should be able to search and sort contacts.
- Apply at least 5 design patterns.

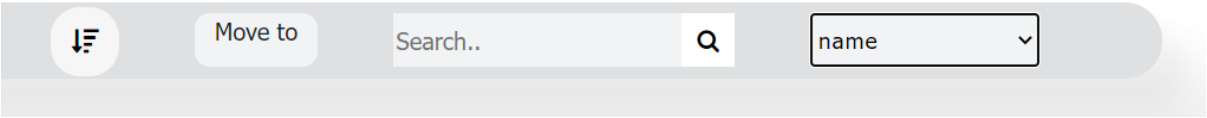
How-To-Document:

- Download the zip file containing the front and back folders.
- Open the back end folder using any ide.
- Note: to run the back end you should download PostgreSQL 15 database and set up the database on your computer.
- If you face an error about the port used you can go to src/main/java/resources/application.resources and then change the server port, then run the project again.
- Open the front-end folder using an ide.

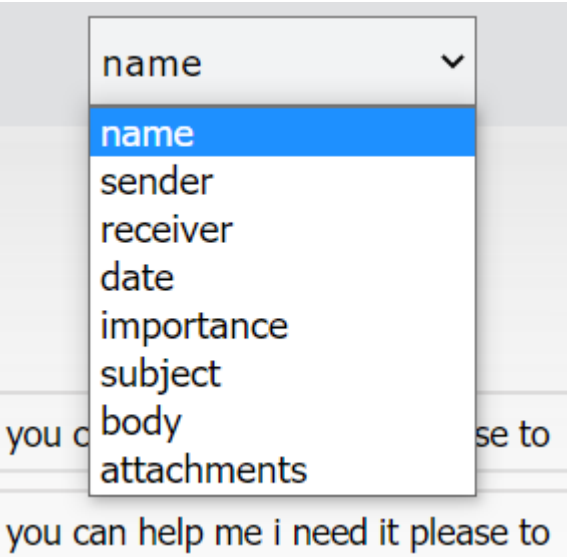
- Make sure you have installed nodeJs on your computer, If not open the terminal and type “npm install”
- After installing you can serve your project on a local host by typing on the cmd (ng serve –open) Make sure you are on the project directory, Note: if you have changed the port in the back end make sure to update it in the emailService file in the front folder.
- The site will be open and the email application is ready to use.

Design Decisions:

- the user could search or sort by column by different attributes, to do that there is a droplist he could search or sort depending on the value of the droplist

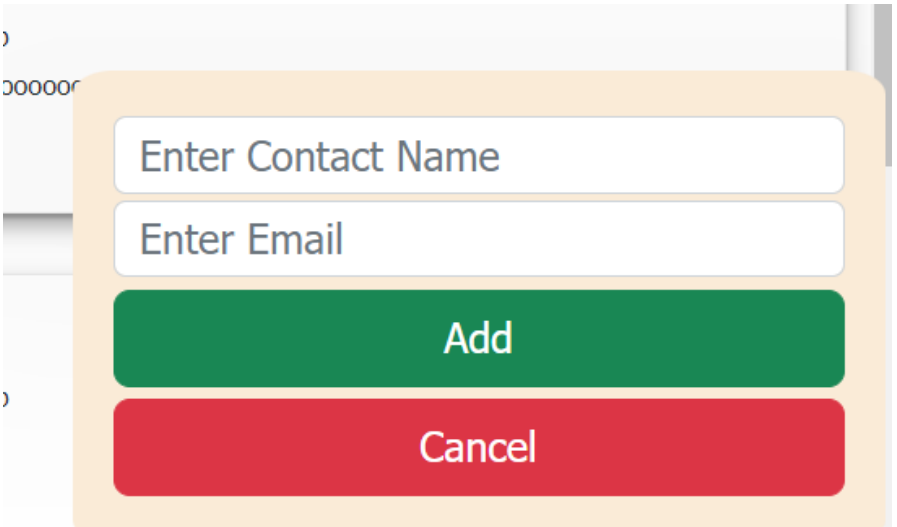


so whatever the value in the droplist the search and the sort are applied on it.

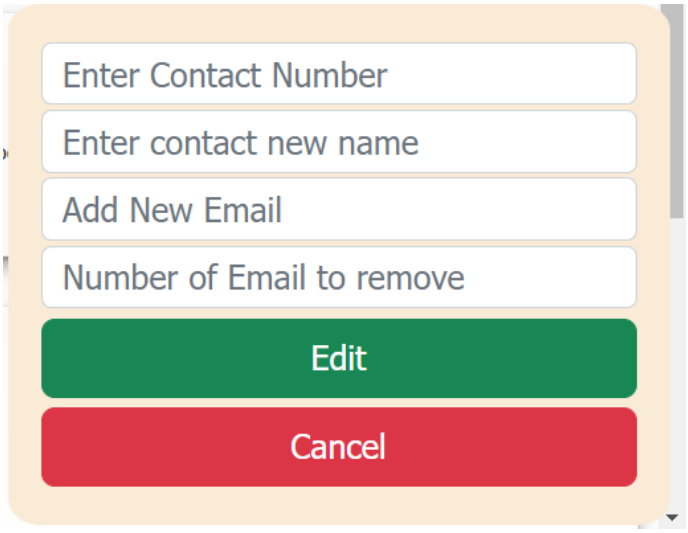


- In contacts, all the manipulations are done by a pop screen which takes the data and does the operation where each icon takes data input from the user.
- The same procedure is applied for the folders section.

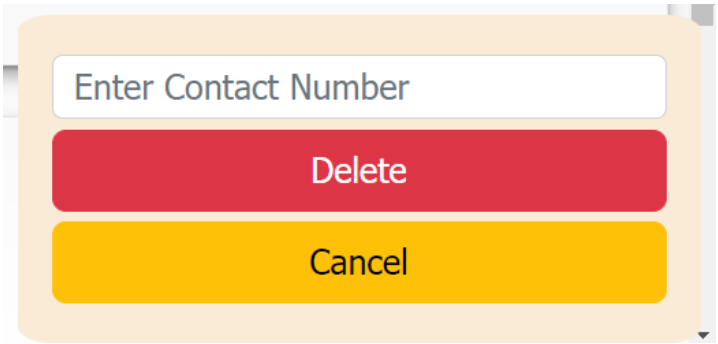
- add contact:



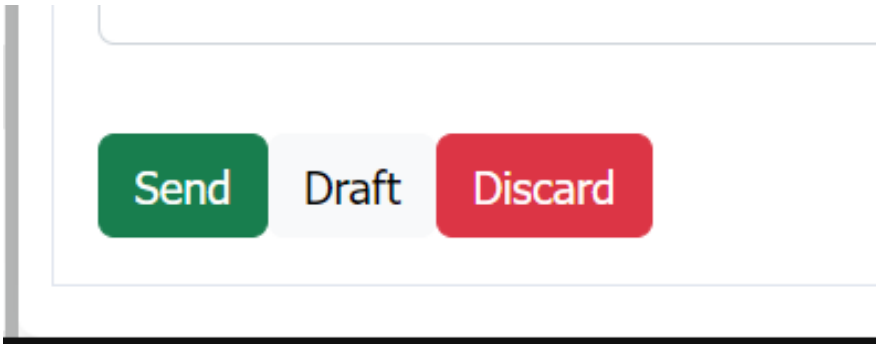
- edit contact:



- delete contact:

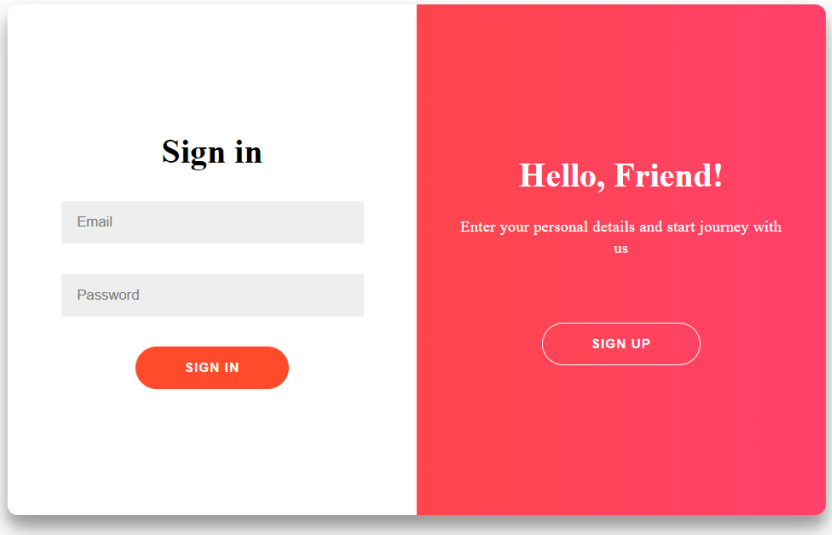


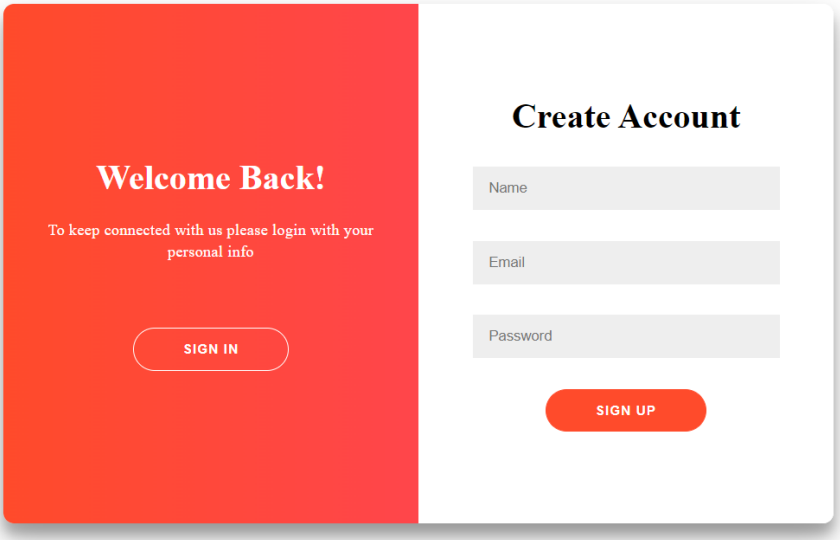
- When the user went to the compose section he had 3 options either send or draft or discard so whatever the option the user wants he clicks on the desired button, so the email to go to draft the user must click on draft.



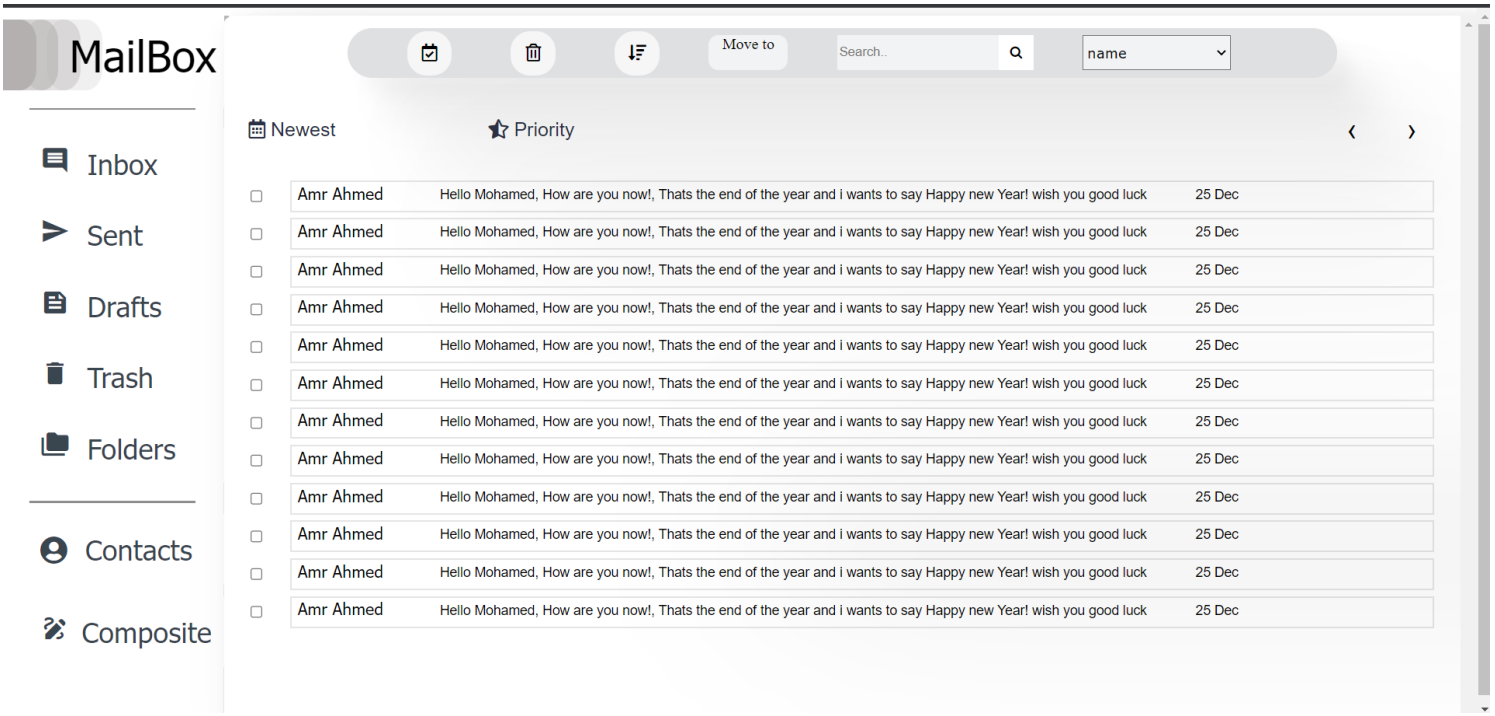
UI Snapshots:

- login - signup screen:

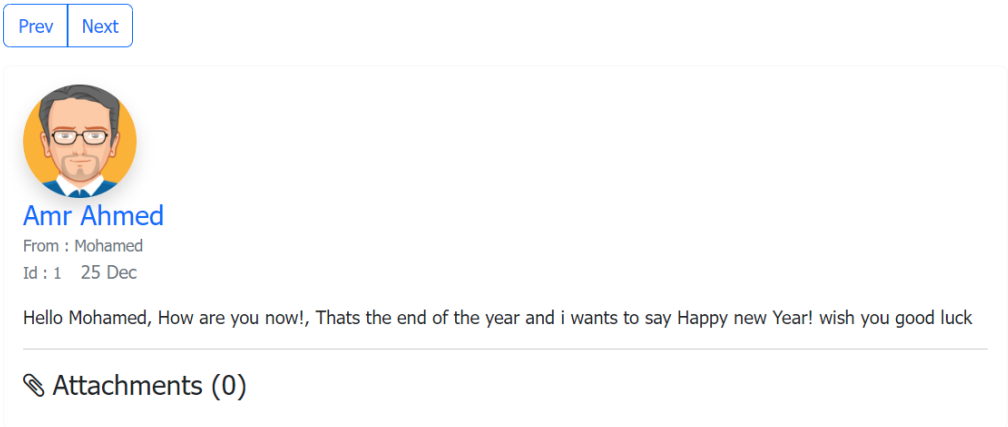




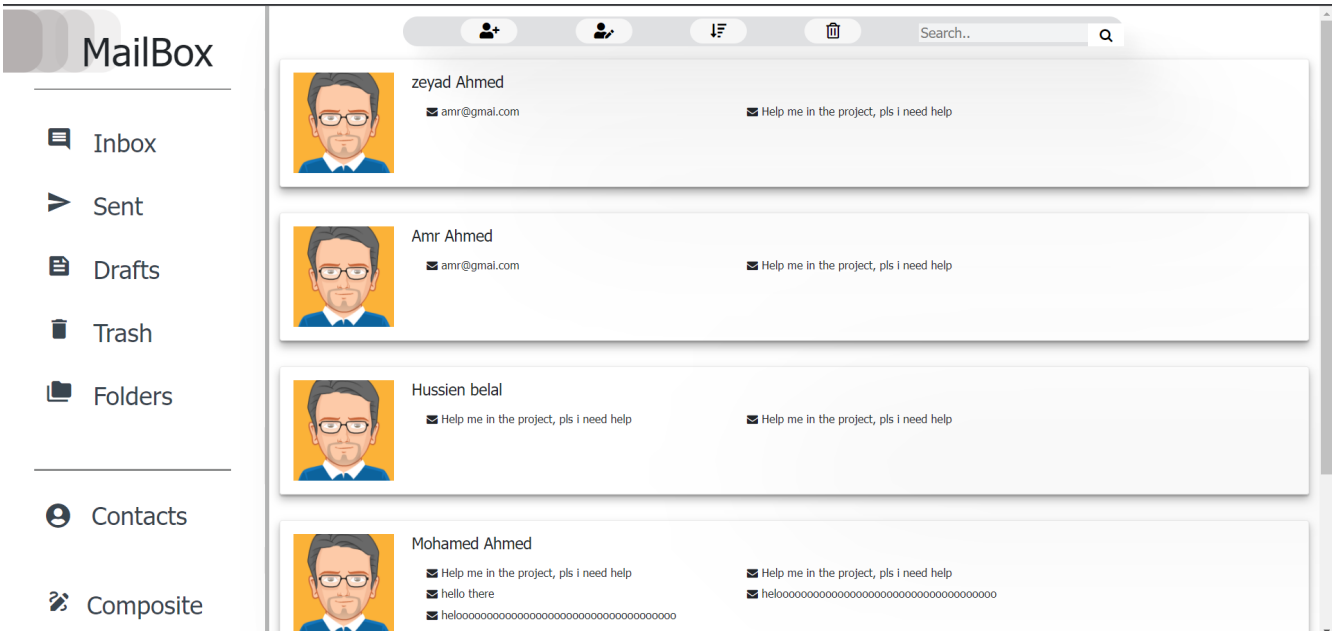
- Inbox, draft, sent and trash folders:



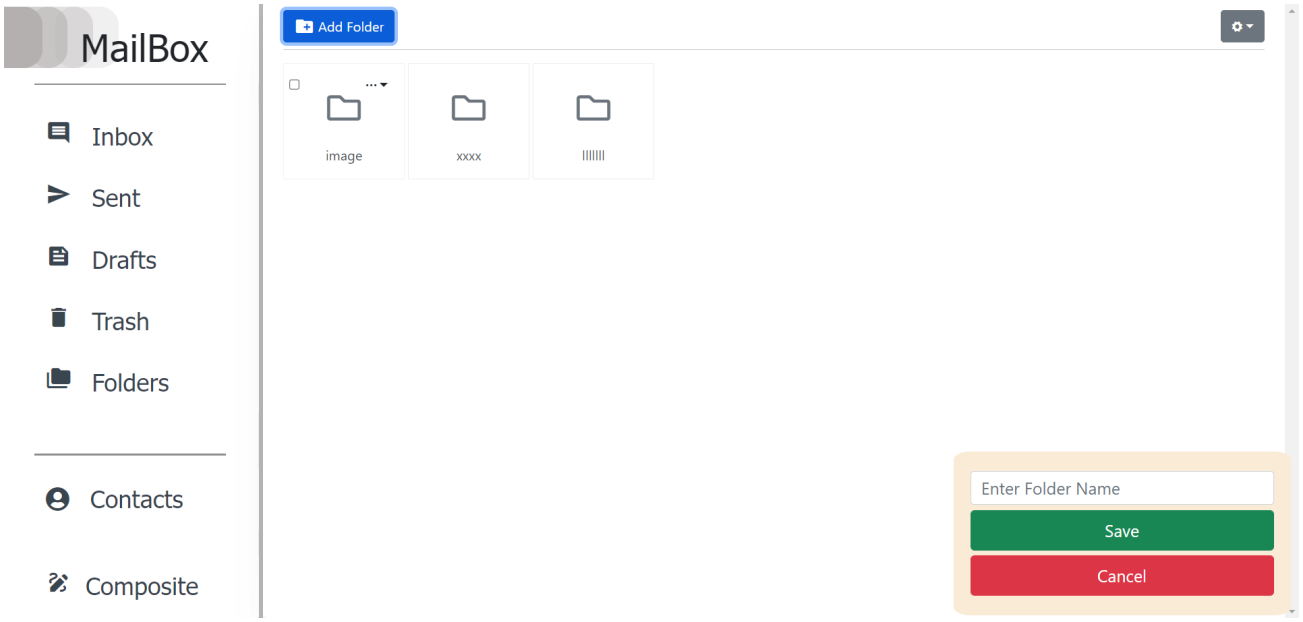
- email view:



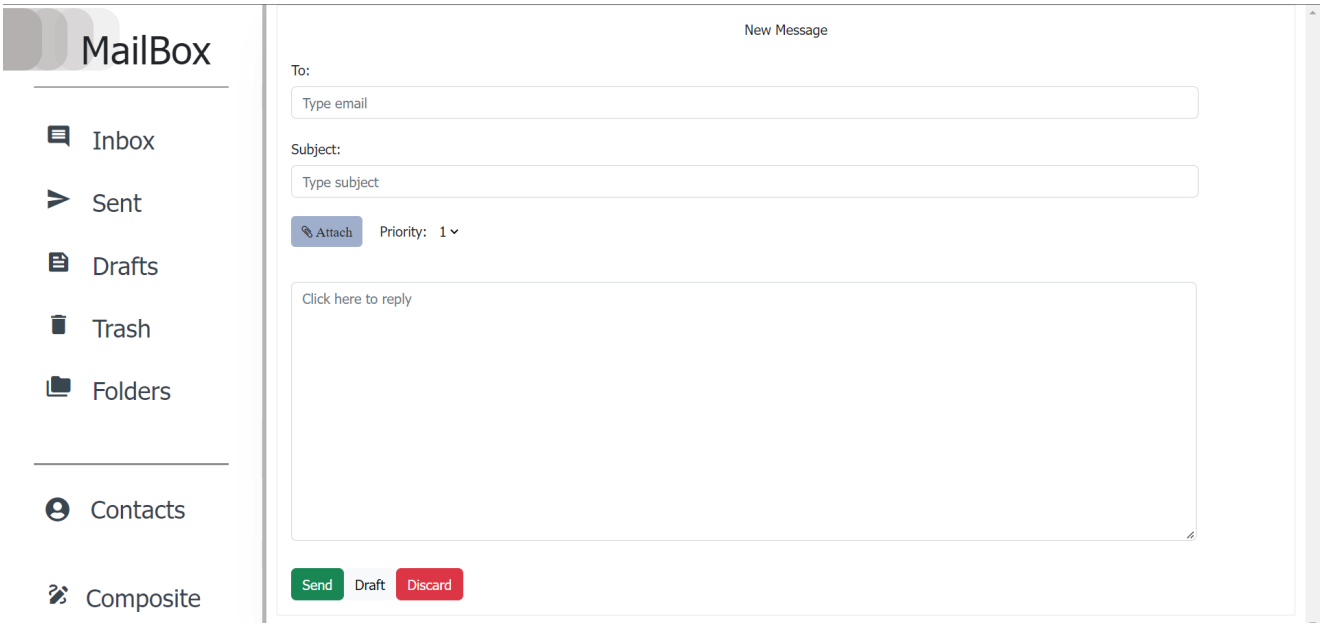
- contacts:



- Folders:

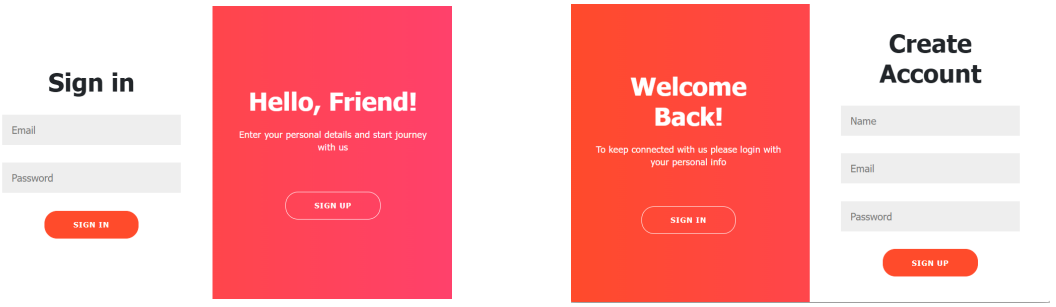


- composite:

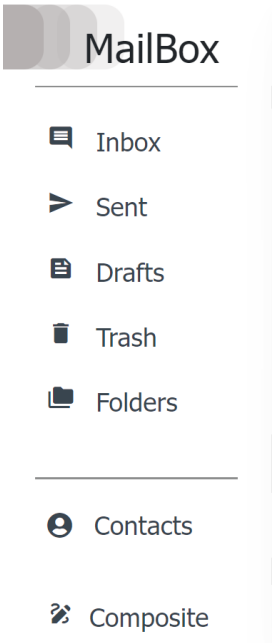


User Guide:

- the user first will have to sign in by entering the username and the password the press sign in if he already signed up before on the site if not, press sign up and create and account then press sign up and the sign in screen will open then he sign in.

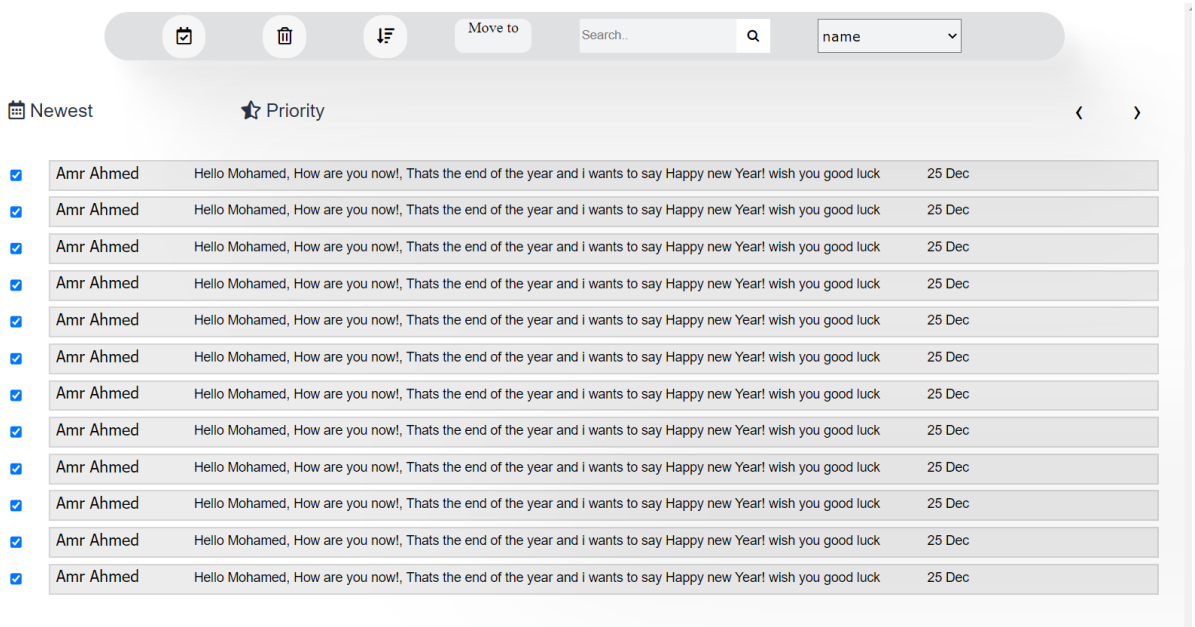


- After signing in the user will be navigated to the inbox section where all the received emails appear.

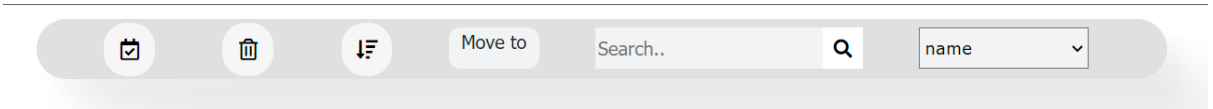


- to navigate between the different folders the user selects the desired folder and the list of emails will appear.

- For email manipulations the user has the options at the top of the screen and he is able to select more than email and perform the operation.



The checkbox at the left of each email allows the user to select it and check the box.

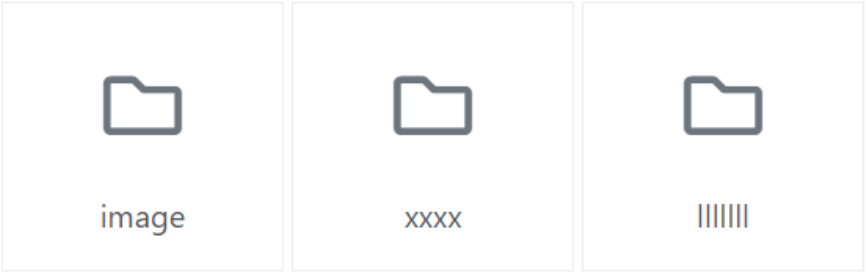


and after selection he could perform one of the operations which are:

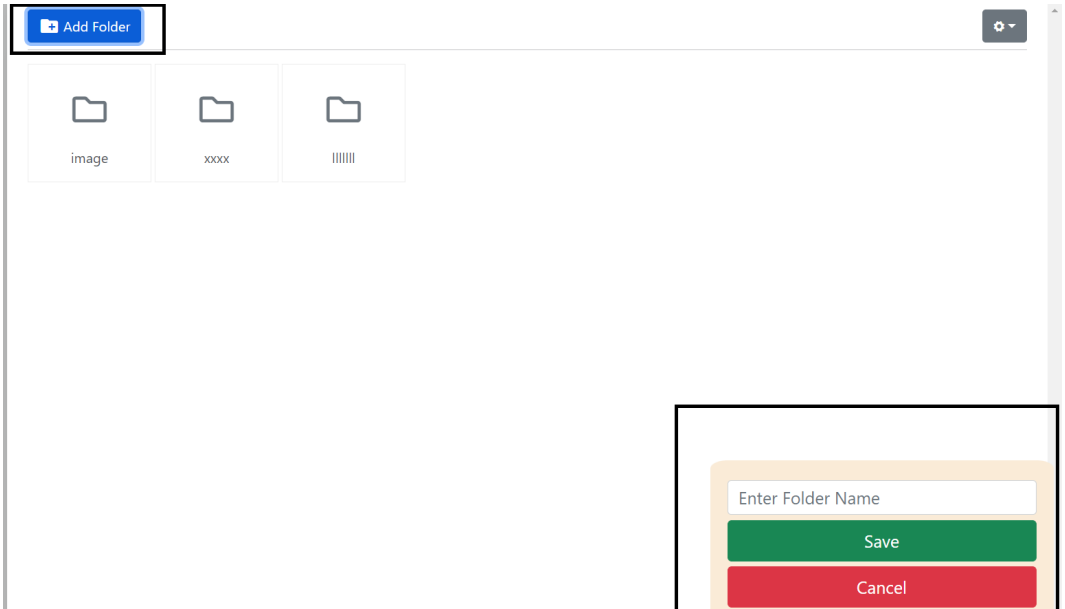
- select all emails or deselect.
- move emails to trash (delete).
- sort the emails upon the label at the drop list.
- move emails to a user folder.
- search for emails by the label at the drop list.

Folders section:

- Here the folders of the user appear and he could choose any of them and the emails appear as above.

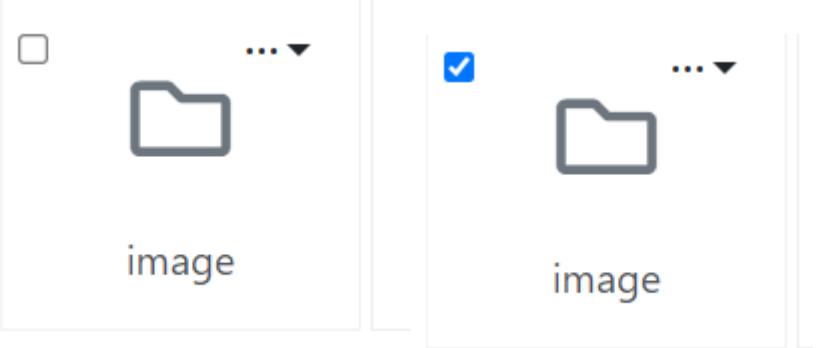


- to add a folder he presses add folder above.

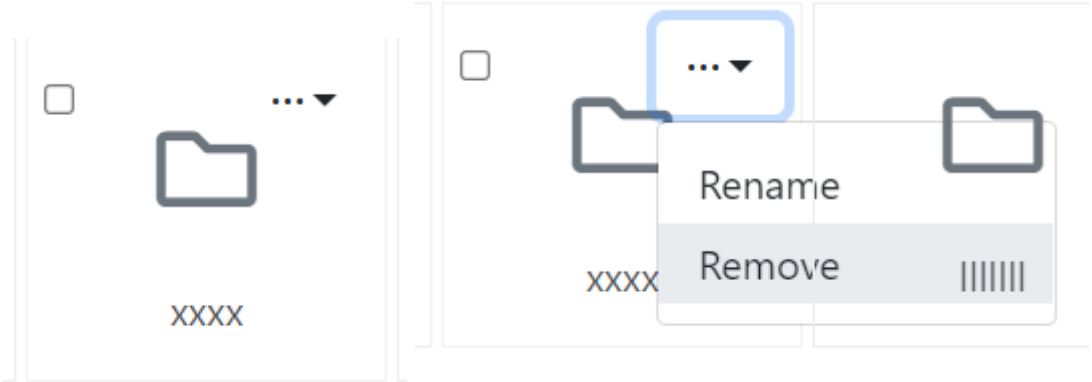


- After pressing the add folder a window appears where he can type the folder name then press save or cancel.

- He could also select several folders and remove them by hovering the mouse on them and a checkbox will appear.

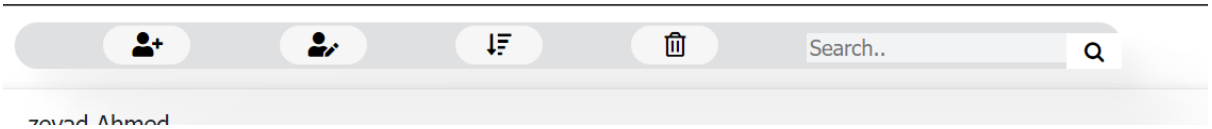


- he could also rename or remove a folder by selecting the drop list on the folder.



Contacts section:

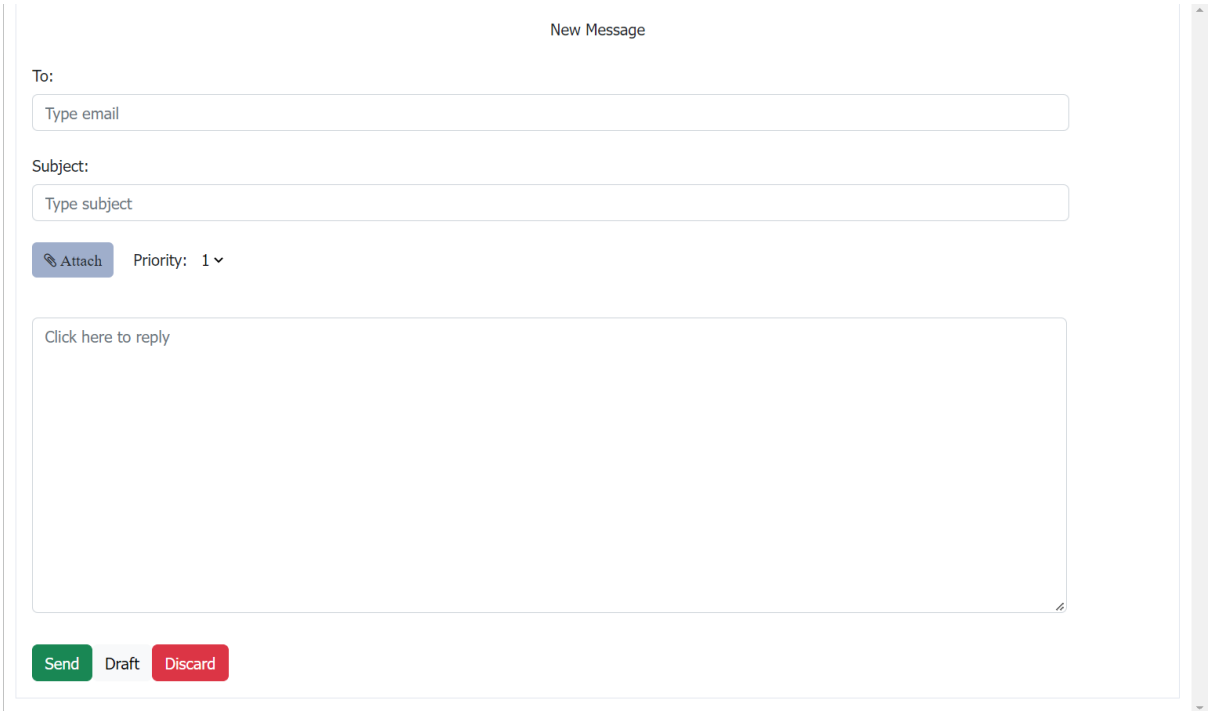
- here all the contacts of the user appear and the emails for each contact.
- the user could do some operations by selecting an option from the top bar.

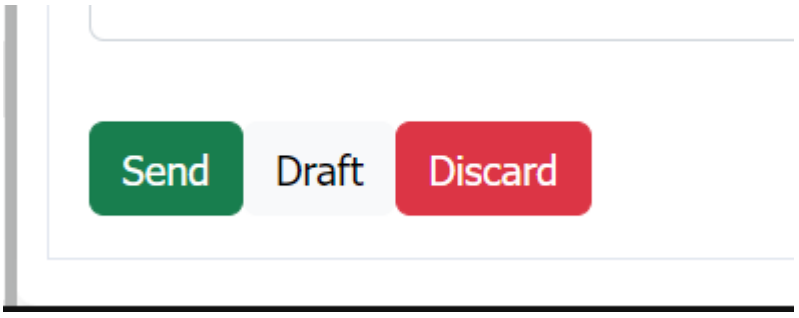


- Each operation will open a window where he enters the data associated with the operation then the operation is done.
- the user could:
  - add a contact.
  - edit a contact.
  - sort the contacts by name.
  - delete a contact.
  - search in the contacts by name.

Composite section:

- here a form appears for the user to fill it where he can send a mail to others and attach one or more attachments.
- the user has the option to send the email or send it to the draft or discard what he filled.



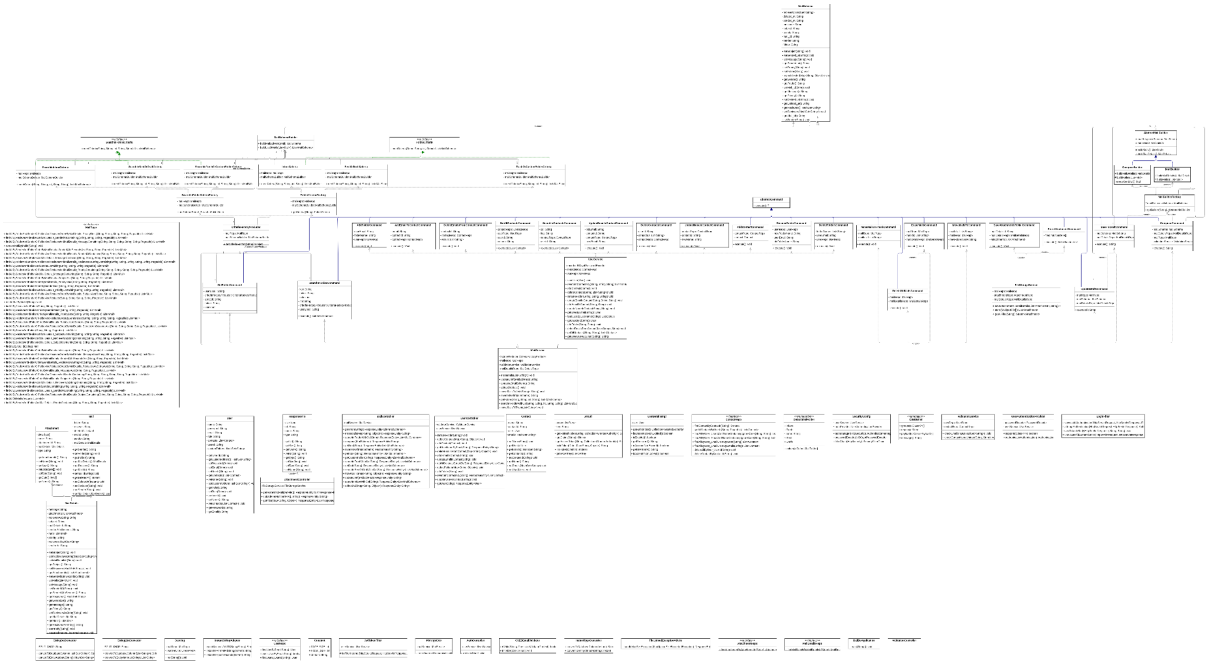


Github Repo links:

FrontEnd: https://github.com/AmrAhmed119/Email-Service.git

BackEnd: https://github.com/MohamedAnwar121/MailBE.git

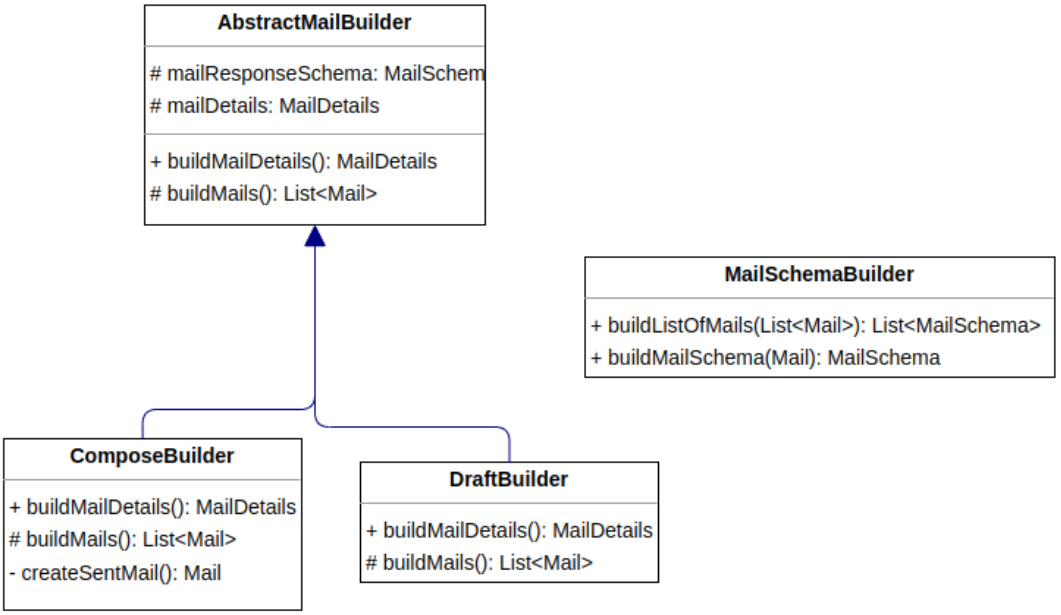
UML Class Diagram:



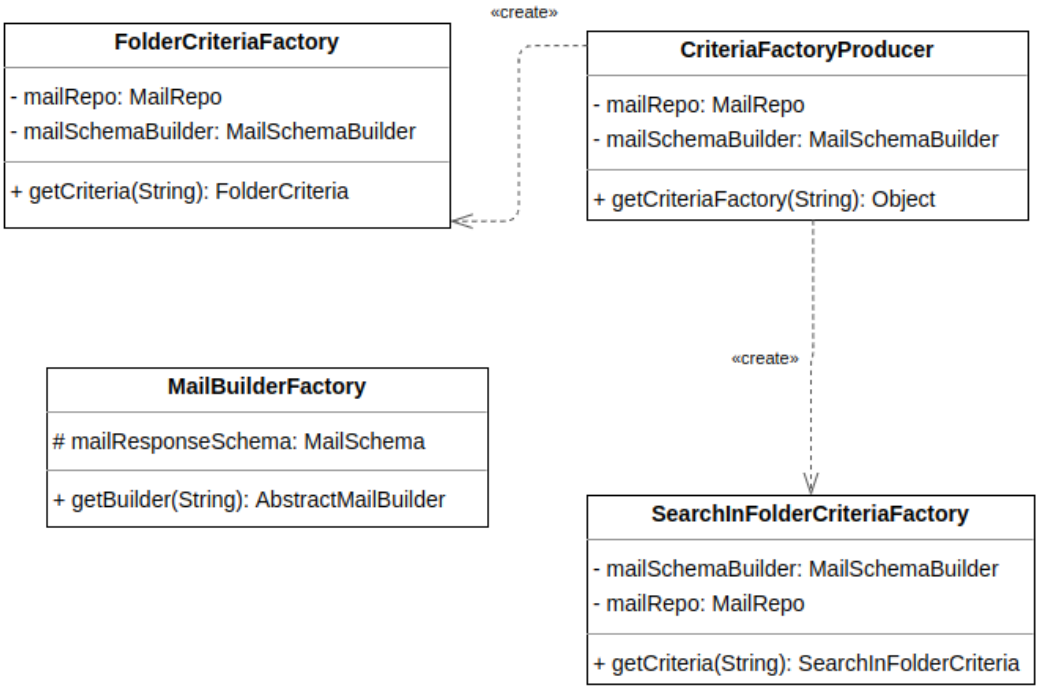
Packages :

Model:

Builder:



Factory:



File command:

SaveAttachmentsToMailCommand

- attachments: List<Attachment>
- mailDetailsId: String
- mailDetailsRepo: MailDetailsRepo

+ execute(): Void

StoreAttachmentCommand

- files: MultipartFile[]

+ execute(): List<Attachment>

Controller:

UserController

- userService: UserService
- validatorService: ValidationService

+ addEmailToContact(String): void
+ deleteContacts(Map<String, Object>): void
+ getUserNameByEmail(String): ResponseEntity<String>
+ addContact(Contact): void
+ searchInContactsBy(String): ResponseEntity<List<Contact>>
+ renameFolder(String): void
+ updateEmailInContact(String): void
+ updateNameInContact(String): void
+ getAllContactsSortedBy(String): ResponseEntity<List<Contact>>
+ addFolder(String): void
+ getUser(String): ResponseEntity<User>
+ deleteEmailsFromContact(Map<String, Object>): void
+ deleteFolders(Map<String, Object>): void

MailController

- mailService: MailService

+ getCustomFolder(String): ResponseEntity<List<MailSchema>>
+ searchInFolderWithCol(String): ResponseEntity<List<MailSchema>>
+ deleteMail(Map<String, Object>): ResponseEntity<String>
+ getInbox(String): ResponseEntity<List<MailSchema>>
+ searchInTrashWithCol(String): ResponseEntity<List<MailSchema>>
+ updateDraft(MailSchema): ResponseEntity<String>
+ getSent(String): ResponseEntity<List<MailSchema>>
+ saveToDraft(MailSchema): ResponseEntity<String>
+ getDraft(String): ResponseEntity<List<MailSchema>>
+ compose(MailSchema): ResponseEntity<String>
+ getTrash(String): ResponseEntity<List<MailSchema>>
+ moveMailTo(Map<String, Object>): ResponseEntity<String>
+ searchInSentOrDraftWithCol(String): ResponseEntity<List<MailSchema>>
+ eraseMails(Map<String, Object>): ResponseEntity<String>
+ searchInInboxWithCol(String): ResponseEntity<List<MailSchema>>

AuthController

- userService: UserService

+ register(User): void

ValidationController

- validationService : ValidationService
+ checkEmailInDB(map : Map<String,Object>)

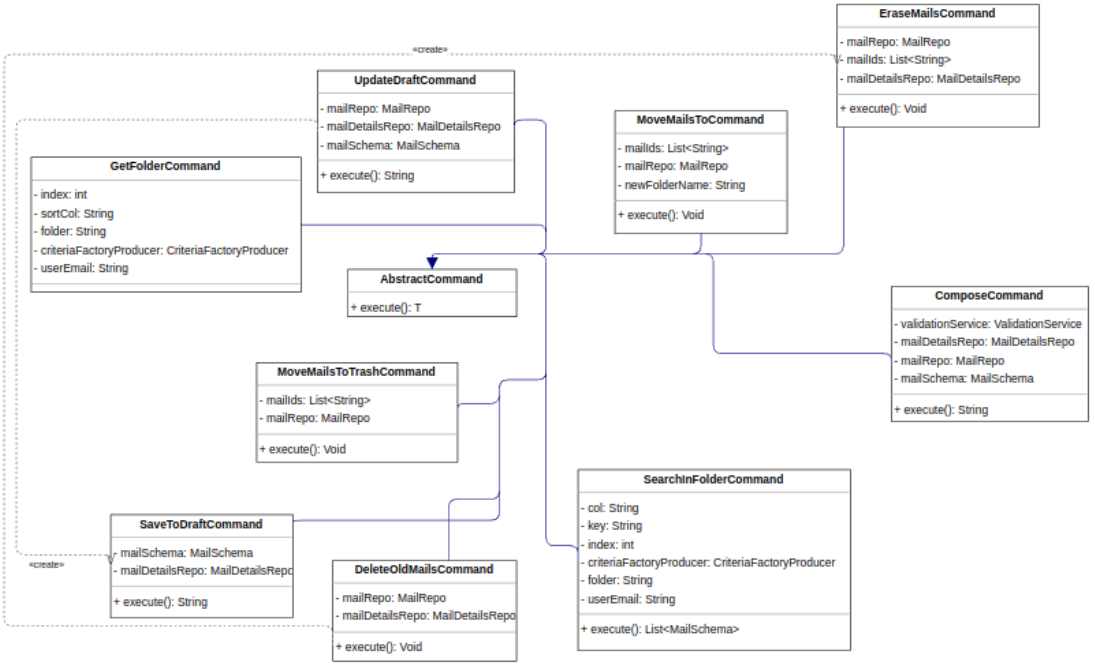
AttachmentController

- fileStorageService: FileStorageService

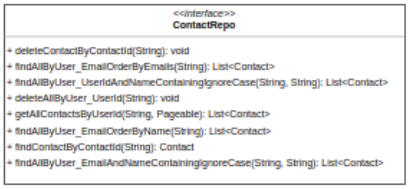
+ uploadFile(MultipartFile[], String): ResponseEntity<String>
+ generateUrls(MultipartFile[]): ResponseEntity<List<ResponseFile>>
+ getFiles(Map<String, Object>): ResponseEntity<List<ResponseFile>>

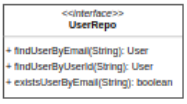
Filter:

MailCommand:

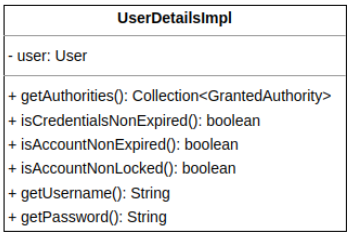


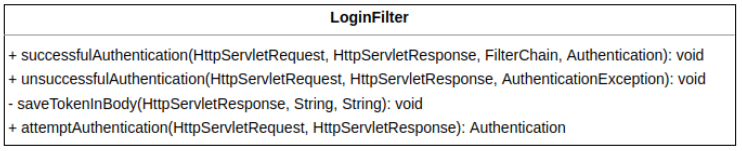
Repository:

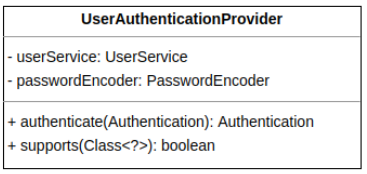


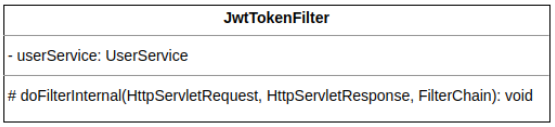


Security:

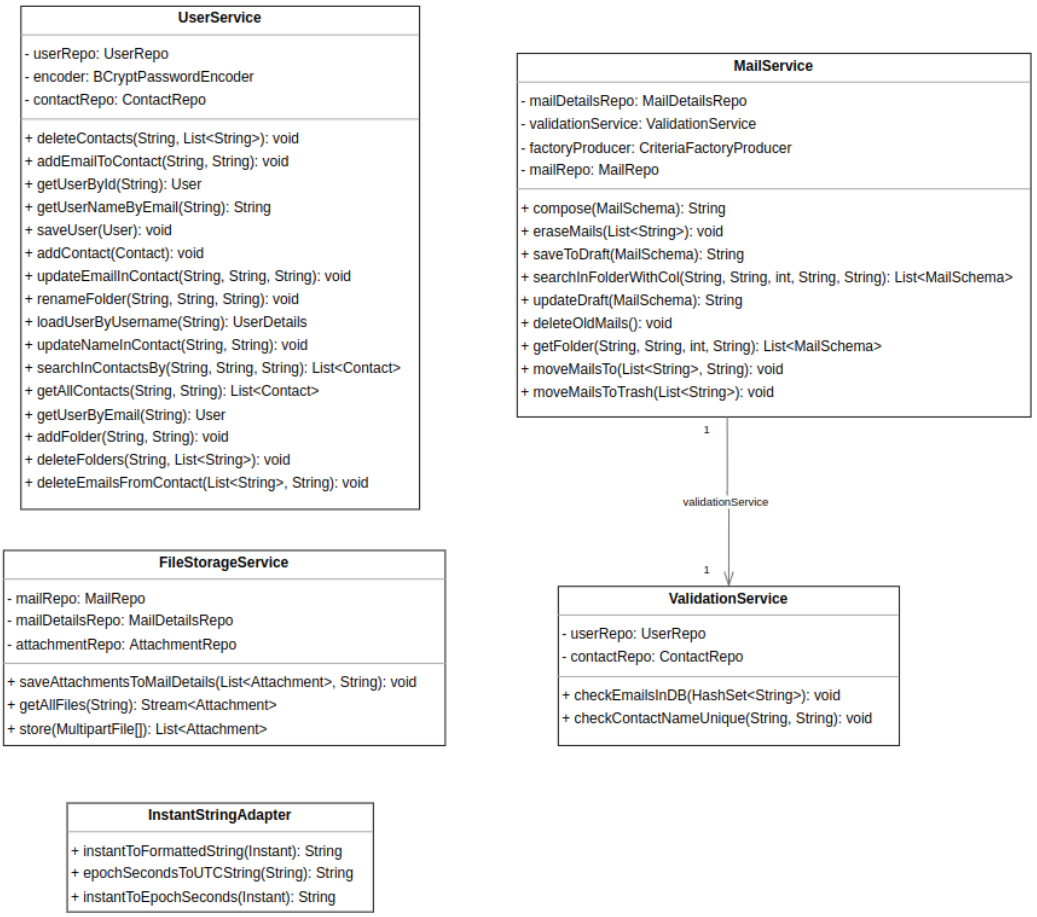




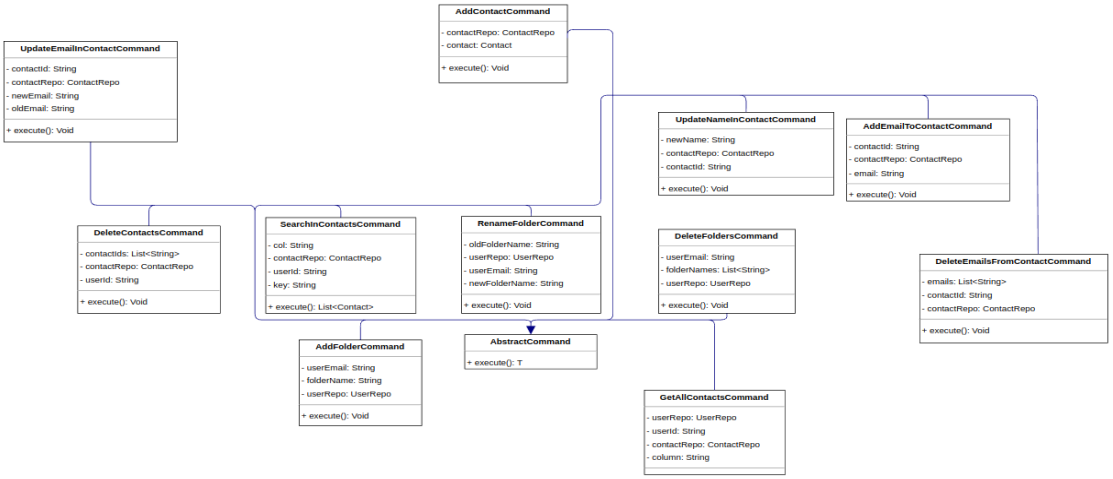




Service:



UserCommand:

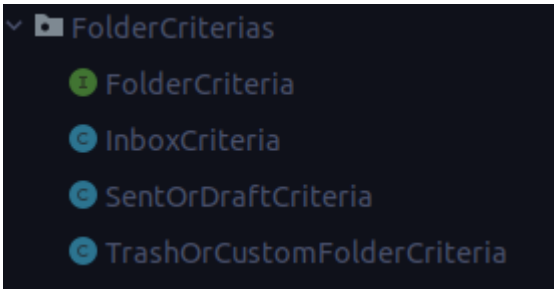


Design Patterns Used:

Filter:- Implemented two filters for getting mails from folder with a sorting criteria (column in the database) and the filtering was implemented using spring data jpa and queryDsl to access and manipulate mails and users in the database

CriteriaInterface For Sorting (Based On Folders):

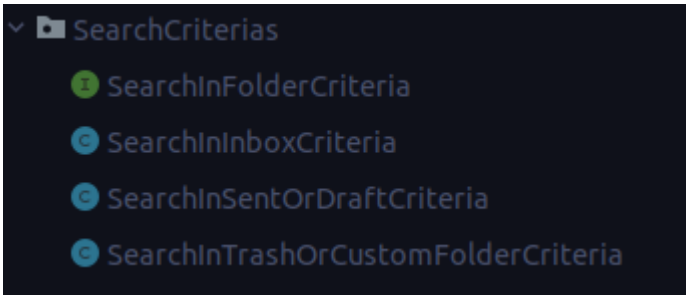
```
3 implementations
public interface FolderCriteria {
    3 implementations
    List<MailSchema> meetCriteria(String userEmail,String folder, int index,String sortCol);
}
```



Sorting and getting mails based on folder

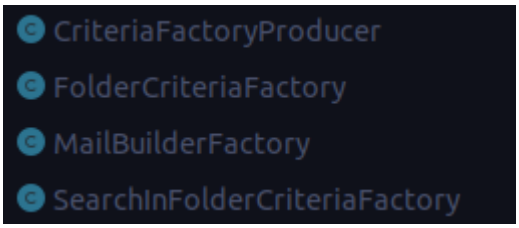
CriteriaInterface For Searching:

```
3 implementations
public interface SearchInFolderCriteria {
    3 implementations
    List<MailSchema> meetCriteria(String userEmail, String folder, int index, String col, String key);
}
```



Searching with some column on a folder

Factory:- Since we had different criteria classes for the filter pattern we implemented two factories to create the criteria objects that implements **FolderCriteriaInterface** and **SearchInFolderInterface** and we have MailBuilderFactory which is responsible for creating Builder Objects which uses MailSchema to build Mail And MailDetails Objects and MailBuilderFactory uses **AbstractMailBuilder** as a parent to create 2 builders **ComposeBuilder** And **DraftBuilder.**



Abstract factory:-

Since we had 2 factories for each set of filter criteria we used the abstract factory pattern (Factory of Factories) , **CriteriaFactoryProducer** which is responsible for creating FolderCriteriaFactory and SearchInFolderCriteriaFactory.

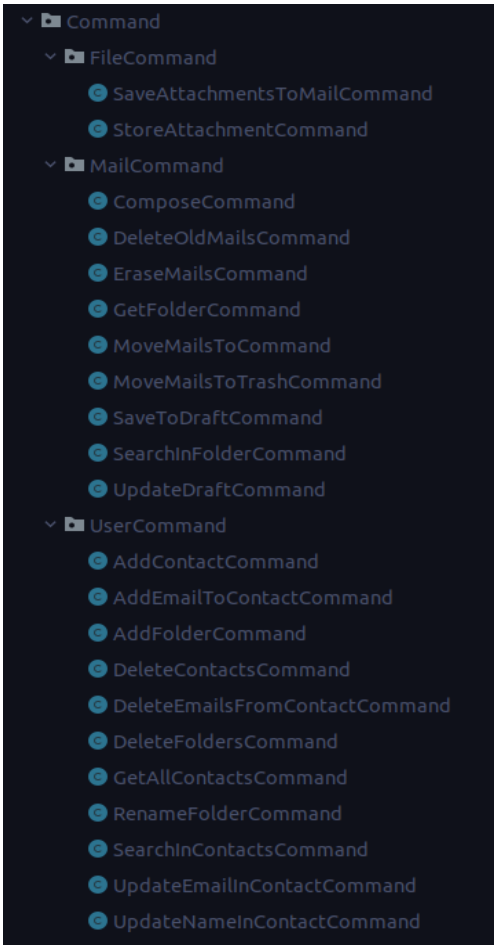
Command:-

we used command design pattern in three phases one for

user Command →manage most functions in the user contact such as add, delete, search and rename contact.

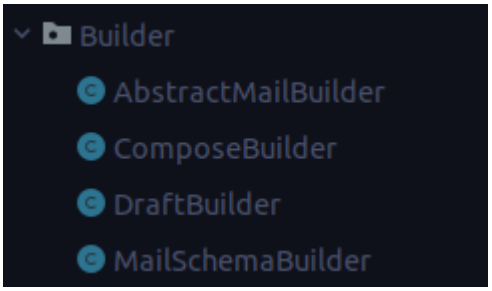
mail Command → manage most functions in the user mail like deleting or getting specific folder update draft search in a folder.

file Command →manages attachments in which it converts multipart files into attachments and takes those attachments and puts them in the database.



Builder:-

We used the builder design pattern to build the mail schema, the draft schema, the compose operation schema and the mail details .



Singleton:-

Used singleton to create one instance of our abstractFactoryProducer since we only need one instant and also used it for the builders

```
@Component
@Scope("singleton")
public class CriteriaFactoryProducer
```

```
@Scope("singleton")
public class AbstractMailBuilder
```

Note: @scope(“singleton) is used to create one instance of the object so that spring can inject that same instant every time it’s needed

Design Decisions:-

We used PostgreSQL database as our relational database,

We have five tables in the database.

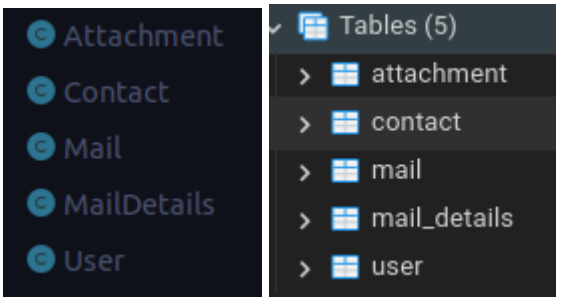
1- Mail

2- MailDetails

3- User

4- Contact

5- Attachment



Our DB Manager for accessing data was SpringJpaRepository which manages sqlQueries without actually typing them.

We used spring security to provide authentication service for users in the system , providing features like access token and also used validation for emails.

Mail table holds the sender , receiver and folder in which the mail is in,it is repeated for senders in cases in which we have many receivers and to solve the issue we used another table **MailDetails** to hold the actual data of the mail like(subject , message and attachments) and it has a one to many relationship with the **Mail** table so that each mail has one instance in the database which reduces the size of the mails in the dataBase.