**Web mail project**

**Names and ids:**

Name1: البراء مصطفى محمد

ID1: 22010636

Name2: ايمن ابراهيم محمد قطب

ID2: 22010656

Name3: عبد الرحمن السيد صديق

ID3: 22010870

Name4: اسلام هاني حسين عبدالمنعم

ID4: 19015425

**How to run:**

*database Setup:*

* 1. Install [MySQL :: Download MySQL Installer](https://dev.mysql.com/downloads/installer/) .
  2. Install [MySQL :: Download MySQL Workbench](https://dev.mysql.com/downloads/workbench/) .
  3. Add a new user (webMail with password webMail).
  4. Create a new connection for this user.
  5. Run the SQL script given in the folder to create the database and tables.

*Backend Setup:*

1. install [JDK Builds from Oracle](https://jdk.java.net/) and an IDE to get running with java.
2. Install [Spring Initializr](https://start.spring.io/) and choose (spring web from dependencies)
3. Navigate to the backend folder.
4. Import the project into the IDE and run the spring app or run with Maven on terminal (mvn spring-boot: run)
5. Ensure the server is running on <http://localhost:8080>.

*Frontend Setup*

1. Install [Node.js — Run JavaScript Everywhere](https://nodejs.org/en)
2. Navigate to the frontend folder.
3. Install dependencies: npm install
4. Start the frontend server: npm run dev

*Get started!*

1. Visit the URL that shows up in your terminal after running **npm run dev** to view the paint app in your browser (it should be something like <http://localhost:XXXX/>).
2. Now the paint app should be running on your browser.

**Applied Design Patterns:**

1. **Controller-Service-Repository Pattern (Layered Architecture)**:

* The application follows the layered architecture where controllers (REST controllers) handle HTTP requests, services encapsulate business logic, and repositories interact with the database.

1. **DTO (Data Transfer Object) Pattern** (implicitly used):

* While not explicitly stated, the entities are used as DTOs to transfer data between layers.

1. **Exception Handling Pattern**:

* Custom exceptions like NotFoundException, UserAlreadyExistsException, and WrongPasswordException are used for error handling, which improves readability and modularity.

1. **Dependency Injection**:

* Classes such as services and repositories are injected into controllers using @Autowired, promoting loose coupling.

1. **Singleton Pattern:**

* Spring by default uses Singleton scope for beans.

1. **Façade Pattern:**

* Dealing with a service layer rather than using the JPA repositories directly.
* Dealing with home page to load page elements rather than calling the elements directly

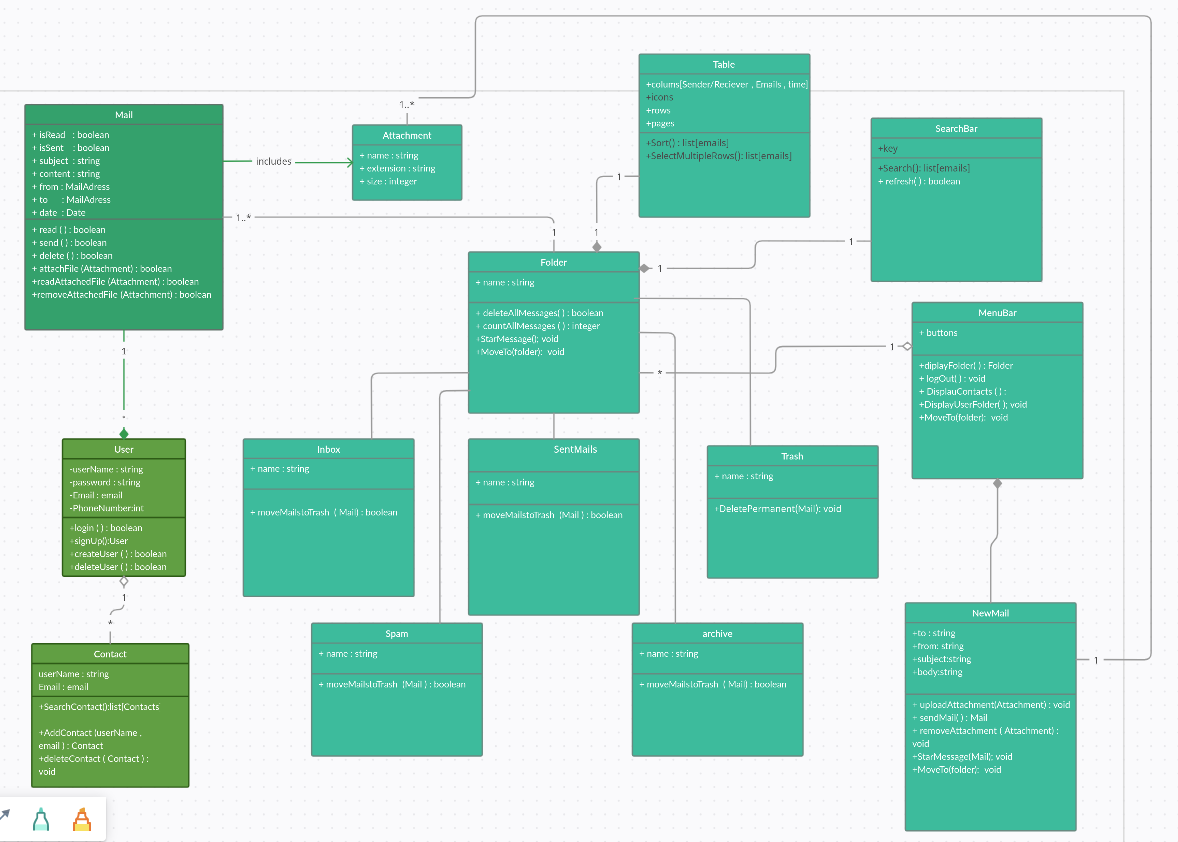
1. **Proxy Pattern:**

* The user doesn’t call the rest controller directly, they call a proxy to handle authentication first then move on.

1. **Factory Pattern:**

* There exists a component that is called with the required folder name and it handles the creation of the folder.

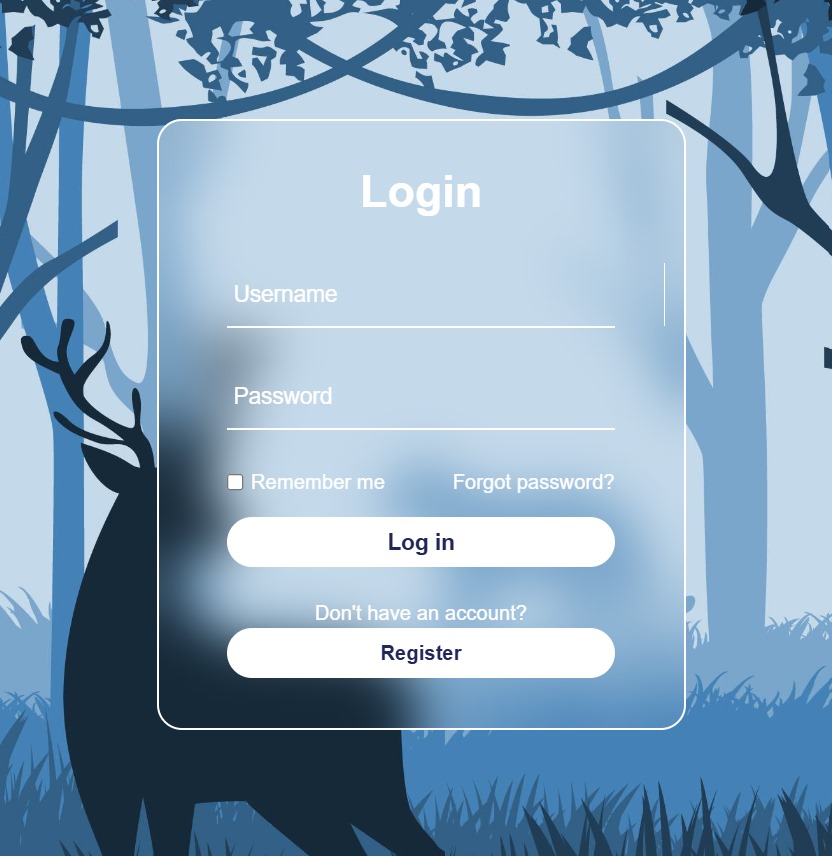
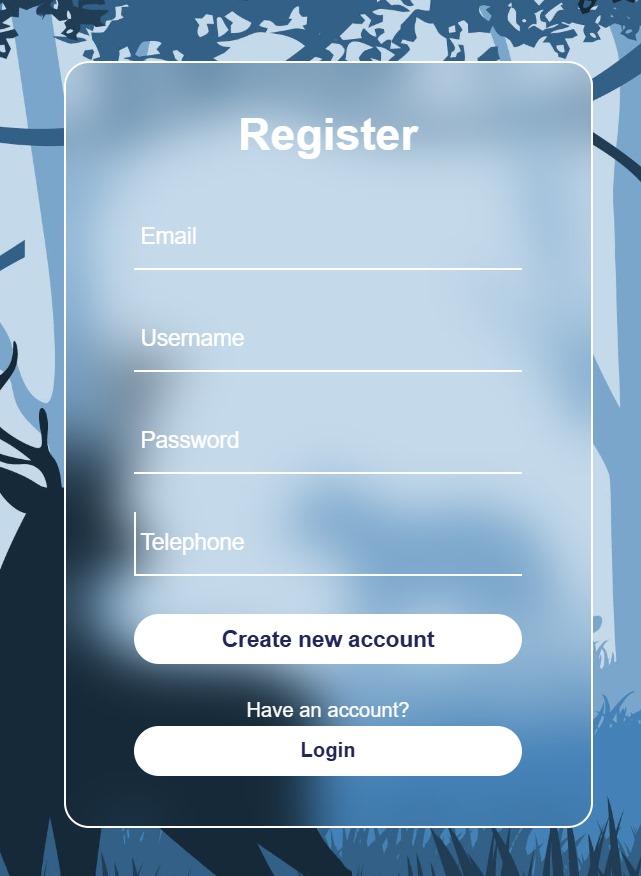
**UML**

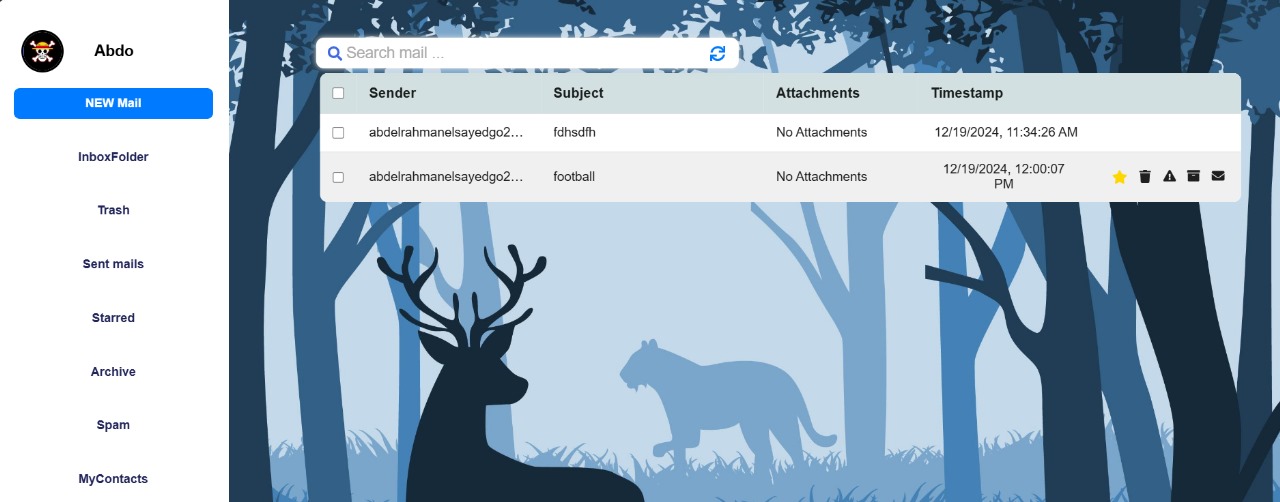
****

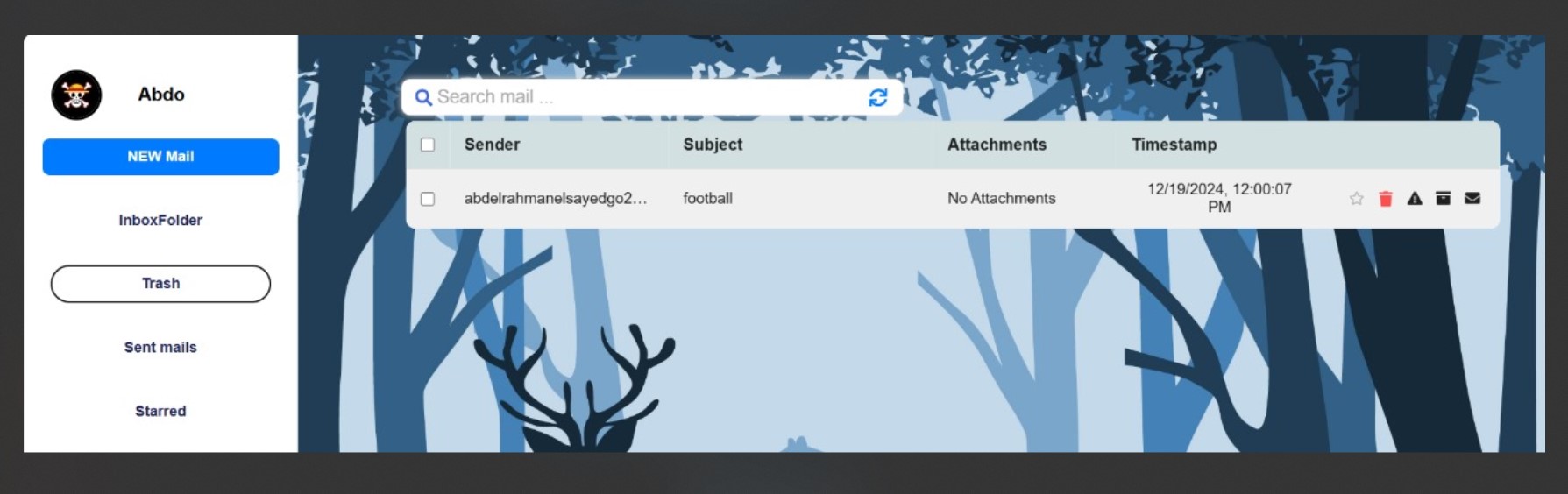
**design decisions:**

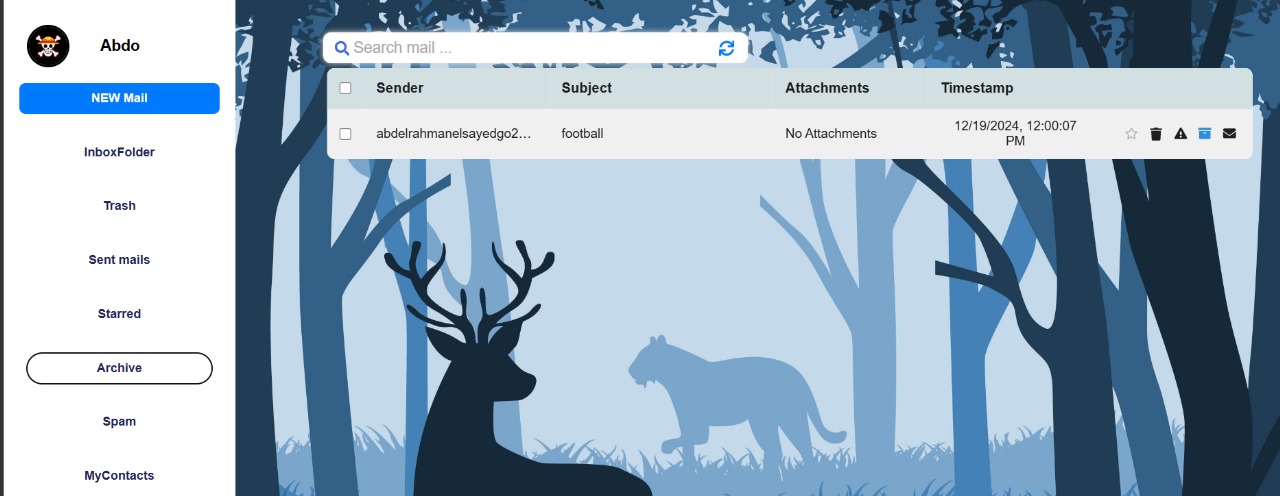
1. For every tab, a session is created with the tab id, which handles multiple user logins at the same time (using session storage).
2. We used tables for users, emails, user Emails (which connects the email to its sender and receiver), attachments and contacts.
3. Tables rows in general are identified using an id as a primary key and may have some connected rows (e.g. attachments have a column that stores the email of its sender).
4. Every email when sent 2 copies are added to the emails table, one with attribute SENT and another with attribute RECEIVED so that the sender has a copy and the receiver has a copy to manipulate it as wished.
5. When an email is marked as starred/read it stays at the same folder with attribute changed (starred is also moved to the starred folder) but when (trashed, archived, spammed) it’s moved to the corresponding folder.
6. We encrypted the passwords using bcrypt.

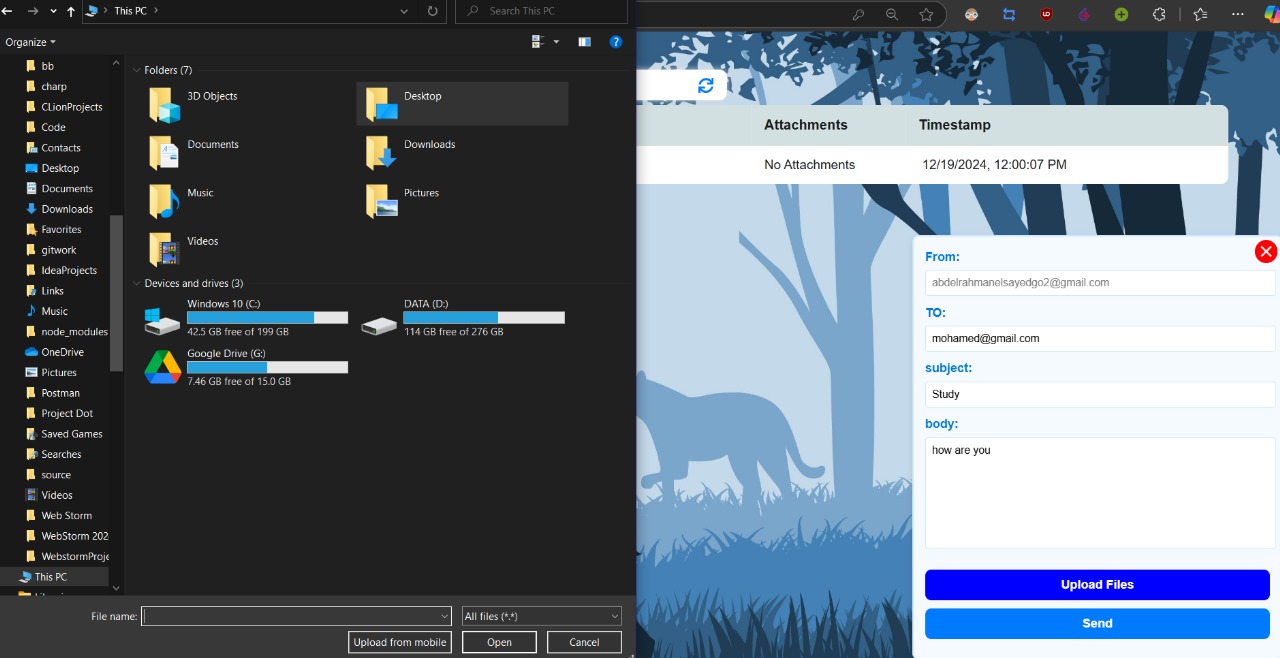
**user guide:**

**login: Register:**

**inbox (notice email icons on hover):**

**trash:**

**archive:**

**new mail:**