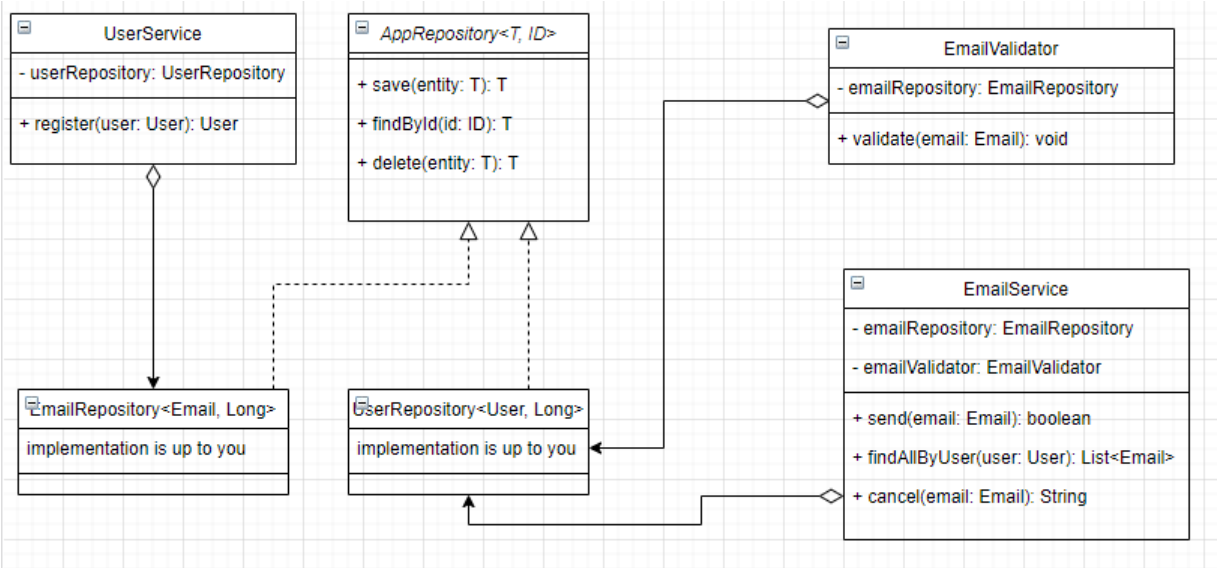
A. Emails

The application you will have to build today represents the backend of an emailing system. The code you write should manage all the required information so that you could perform actions like registering a new user into your system, sending an email to an already registered email address or get a list of new emails for a specific user.

Figure 1: Incomplete UML diagram (you must come up with missing components)



Requirements

1. Read and understand all the requirements. Start by creating the structure presented in the UML diagram. Come up with missing components based on the tasks from the next steps.

2. For users your system should save at least their name and email address. Create a new model for a user and implement the `register` method from `UserService` class and `save` method from `UserRepository` as follows:

- Choose a suitable collection for storing the registered users inside your `UserRepository`
- Save the new user inside the collection chosen at the previous step only if there isn't a user with the same email address registered. Give your new user a new `Long` id which represents the last saved id + 1. Start with id `1L`
- If the email was already taken throw an exception and handle it wherever you're calling the `register` method
- Provide some code for registering two new users and an already existing one

3. For emails its pretty straight forward what information your system should save (from user, to user, subject, message and timestamp). Create a new model for an email and implement the required methods such that your system will cover the following tasks:

- Validate an email such that the 'from' and 'to' users exist and the email has a subject and message which are not blank. If any if the above information is wrong you should throw an exception.
- On sending a new email you should first check that the email is valid before saving it. Set the current timestamp on the email object and return true only if your system managed to save the email.
- Provide two examples of sending a valid and an invalid email

4. Implement the `cancel` method from `EmailService` such that an email could be canceled only if it was sent in the last 30 seconds. Return one of the three messages depending on the context: "Email successfully canceled", "Email cannot be canceled anymore" or "Email does not exist" and provide some code to demonstrate all of the above scenarios.

5. Implement the `findAllByUser` method from `EmailService` such that you could get a list of emails for a specific user. Tweak the existing method such that you could filter the returned list of emails by a specific destination user and provide a way in which you could order the returned list of emails by their timestamp either ascending or descending.