SWE 530 – Final Exam Assignment
Designing a Project Part-3

Mehmet Eyüpoğlu
Instructor: Dr. Hüseyin Birkan Yılmaz

May 24, 2024

TABLE OF CONTENTS

1. Requirements

1.1. Previous Requirements

Previously provided functional requirements:

- **F1:** A workspace has an owner.
- **F2:** There should be channels inside the workspace.
- **F3:** There should be a channel named "main" for all members, but some of the channels should be protected, which means only subscribers of the channel can enter the channel.
- **F4:** In channels, there should be a task management tab for simple task management operations such as assigning tasks to users and viewing the status of the tasks.
- **F5:** People should be able to join a workspace via email and link.
- **F6:** Members of the workspace should be able to post text, media, link to the channels (if they are subscribers of that channel).

- **Login**
  - **F7**: Once logged in to the system, the user shall be able to update their credentials
  - **F8**: On mobile devices, the user shall be logged in for 6 months until a new password is set
- **Registration**
  - **F9**: The user shall be given a pre-defined username and password once they click the invitation link
  - **F10**: On the registration screen, the user shall see for which workspace they are invited
  - **F11**: The user shall be able to customize their login credentials in the registration period

  - **F12**: The user shall be able to enter this information on the registration screen: username, password, firstname, lastname. All required.
  - **F13**: Username must be unique
  - **F14**: The user shall be asked for a password change every 6 months for security reasons
  - **F15**: The user shall be notified via email once they are registered to the system
- **User permissions**
  - **F16**: The system must have the following roles with the attached authority
    - **F16.1**: Owner:
      - They can create the workspace or channels and invite new members with an external invitation link
      - They can remove a person from a channel or workspace completely
      - They can assign roles to other users
    - **F16.2**: Manager
      - They can remove a person from the channel (owner excluded)
      - They can respond to the join channel requests by other users
      - They can transfer the manager role to others
      - They cannot leave a channel that they moderate
    - **F16.3**: User
      - Owner and manager role extend this user role and its authority
      - Users can message, create, assign, close, and comment on tasks
      - Users can create video or voice calls

Non-functional requirements:

- **NF1:** The system must work on both Android and IOS mobile phones.
- **NF2:** The system shall ask for permission if an update is planned.
- **NF3:** The system shall support these languages: English, German, French, Turkish, Chinese, Russian, Arabic.
- **NF4:** The system shall ask for permission if an update is planned.

- **NF5:** Users shall receive notifications for new messages, mentions, or assigned tasks.
- **NF6:** Notification preferences (e.g., email, in-app) shall be customizable by the user.
- **NF7:** Notifications shall include a brief summary of the new activity.
- **NF8:** All communication within the system, including messages and file transfers, shall be encrypted.
- **NF9:** The system shall have a secure password recovery process for users who forget their passwords.
- **NF10:** Access to certain features (e.g., workspace creation) shall require multi-factor authentication for added security.
- **NF11:** Users with administrative roles shall have access to detailed usage reports.
- **NF12:** The system shall log user activities for auditing purposes.
- **NF13:** Users shall be able to report inappropriate content or behavior, triggering a review process by administrators.
- **NF15:** Users shall have the option to synchronize tasks and deadlines with external calendar applications.
- **NF16:** Integration with video conferencing tools for scheduled calls shall be available.
- **NF17:** The user interface shall be designed with accessibility in mind, adhering to WCAG guidelines.
- **NF18:** Users shall be able to navigate and perform essential functions using screen readers.
- **NF19:** Color schemes and font sizes shall be customizable to accommodate various user preferences.

1.2. Additional Requirements

F17. Tasks

- Add comments to tasks with rich text formatting (bold, italics, underline, etc.)

- Assign tasks to other users.

F18. Channels

- Invite others to channels by username or email address.
- Set public or private channel permissions.

F19. Kanban Board

- Create a kanban board for tasks with different stages (e.g., To Do, In Progress, Done).
- Allow dragging and dropping tasks between stages.

F20. Attachments

- Add attachments in the comments in various formats (documents, images, videos).
- Preview attachments within the comments section.

F21. Sprints

- Plan and track sprints with defined start and end dates.
- Assign tasks to specific sprints.
- View burndown charts to visualize sprint progress.

F22. Messaging Text area for messaging or creating tasks or comments should have a text editor with features like:

- Spell check
- Markdown support
- Ability to @mention other users

2. Use Case Diagram

A use case diagram is a visual representation of how a user interacts with a system to achieve a goal, in this case, logging in. The user, or ”actor”, initiates the login process. The system then validates the entered credentials. If correct, the user gains access. If incorrect, an error message is displayed. If the user forgets their password, a reset option is available. This diagram helps ensure the system meets user needs and aids communication among stakeholders.

In a workspace, users interact with channels for various activities. They can join channels, post messages, and manage tasks. Some channels are open to all, while others are restricted to subscribers. This fosters collaboration and communication. Also, you can see the roles matter in the workspace through this diagram.

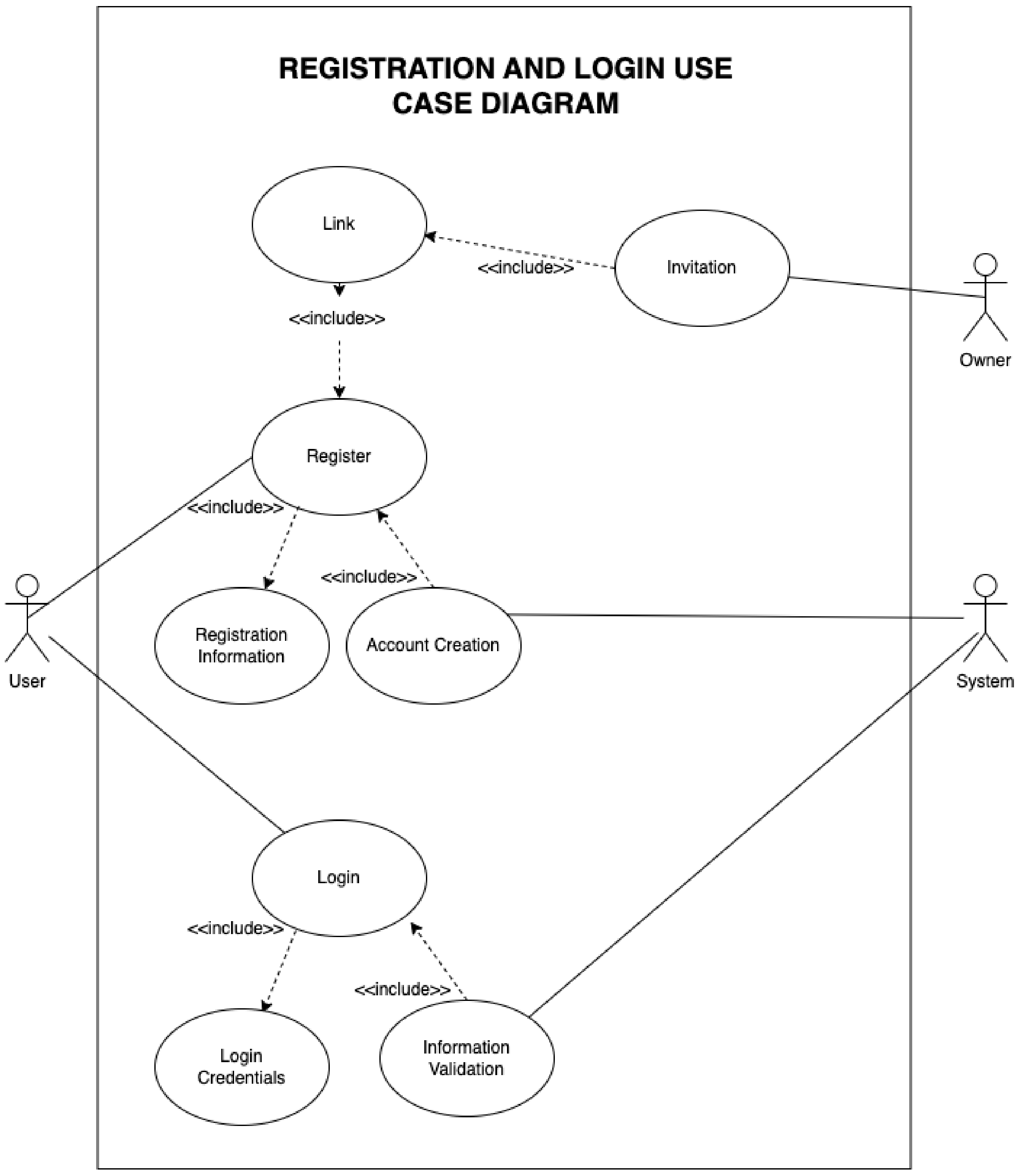

Figure 2.1. Use Case Diagram for Login

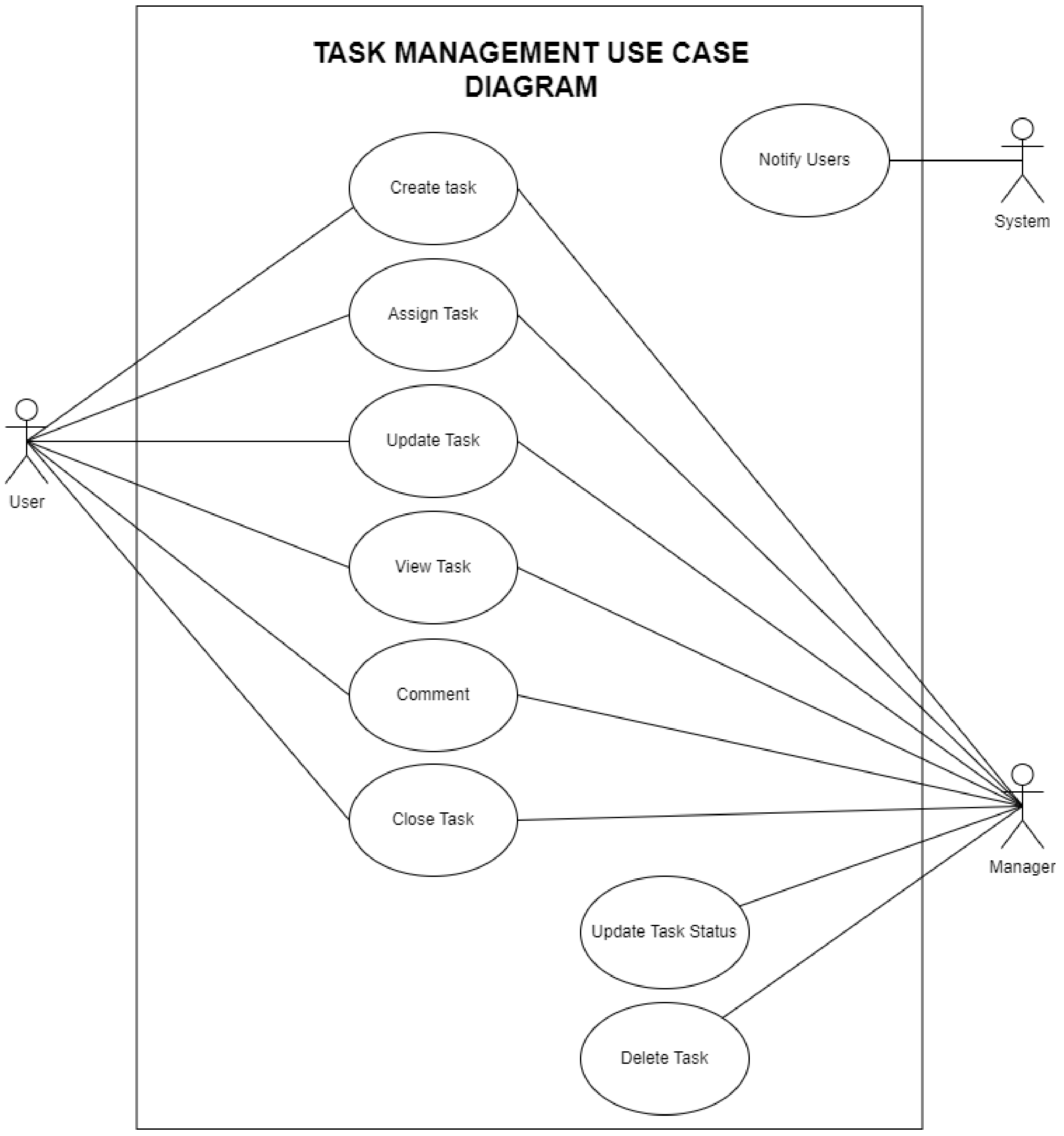

Figure 2.2. Task Management Use Case

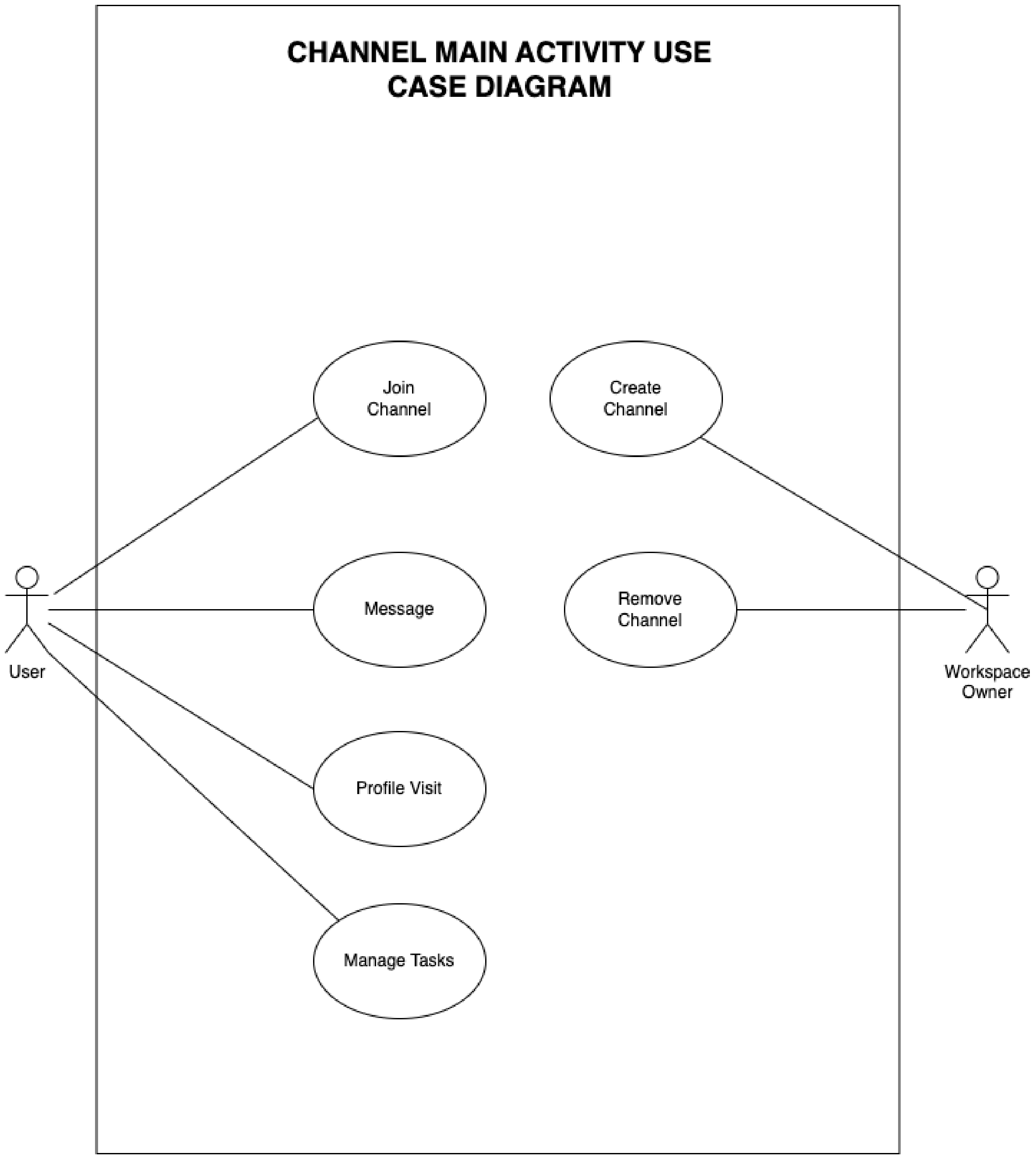

Figure 2.3. Use Case Diagram for Channel Main Activity

3. Class Diagram W/O Functions

A class diagram is a visual tool in UML that depicts the structure of a system. It shows classes, their attributes (data), and methods (behaviors). For example, in a workspace system, a "User" class might have "username" and "password" attributes and "login" and "logout" methods. Relationships between classes, like a User being part of multiple Channels, are also shown.

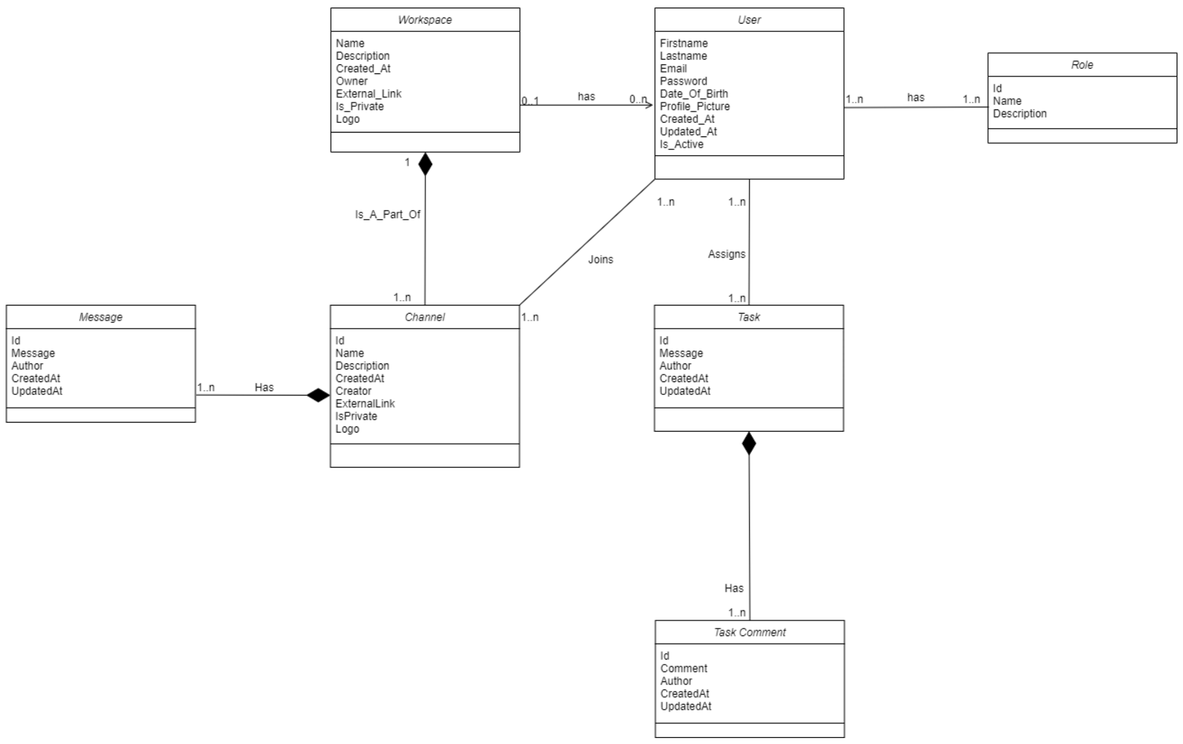

Figure 3.1. Class Diagram Without Functions

4. Class Diagram W/ Functions

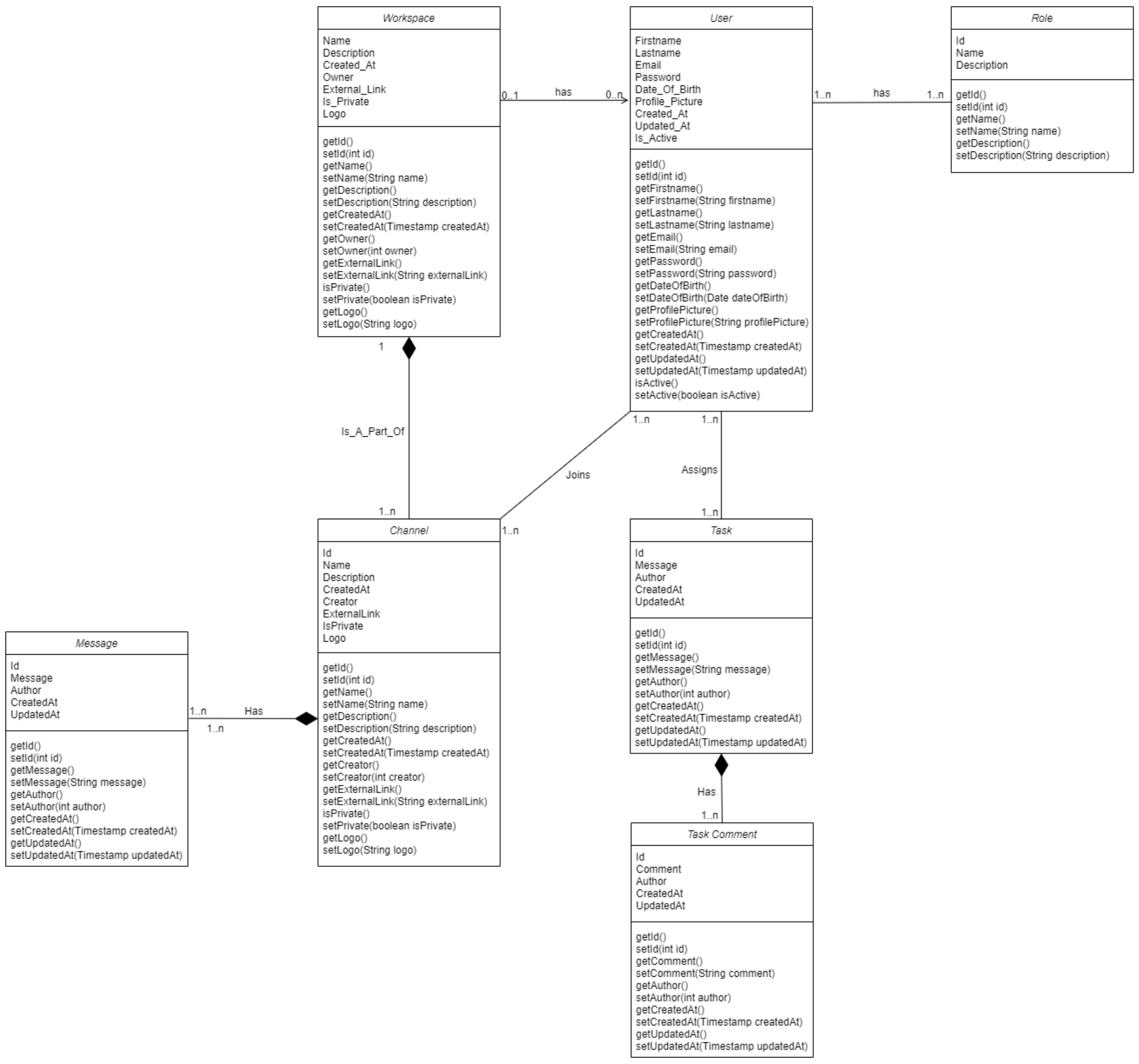

Figure 4.1. Class Diagram w/functions

5. Sequence Diagrams:

This sequence diagram outlines a user login process. It starts with the user attempting to login. The system validates the credentials with the database. If valid, the user sees their dashboard. If invalid, an error is displayed. If the user forgets their password, a reset link is generated and sent via email.

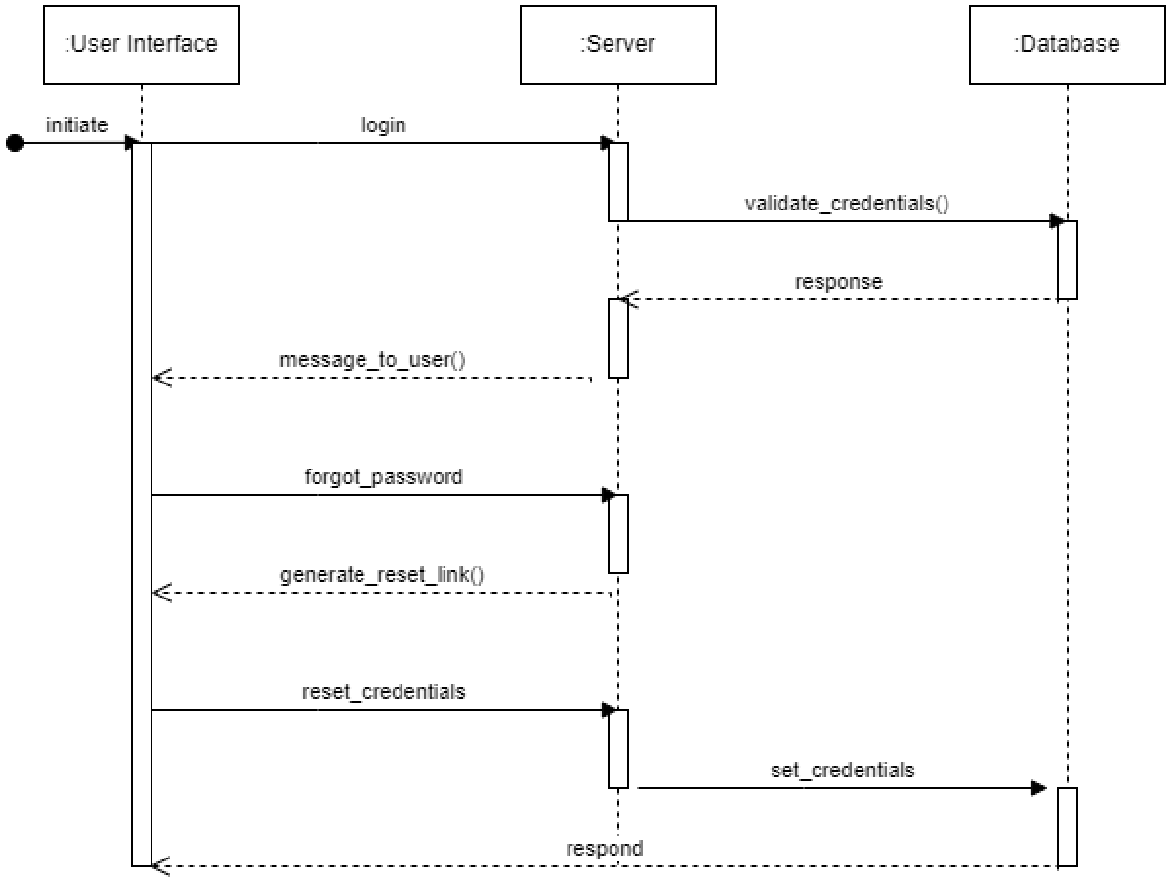

Figure 5.1. Sequence Diagram

6. Jackson Structure Diagrams

The Jackson Structure Diagram (JSD) provides a visual representation of the data structure and the corresponding processing actions of a program. In the context of a task management system, the JSD can be used to illustrate the structure of a task and its related components, such as assignment, general info, and comments.

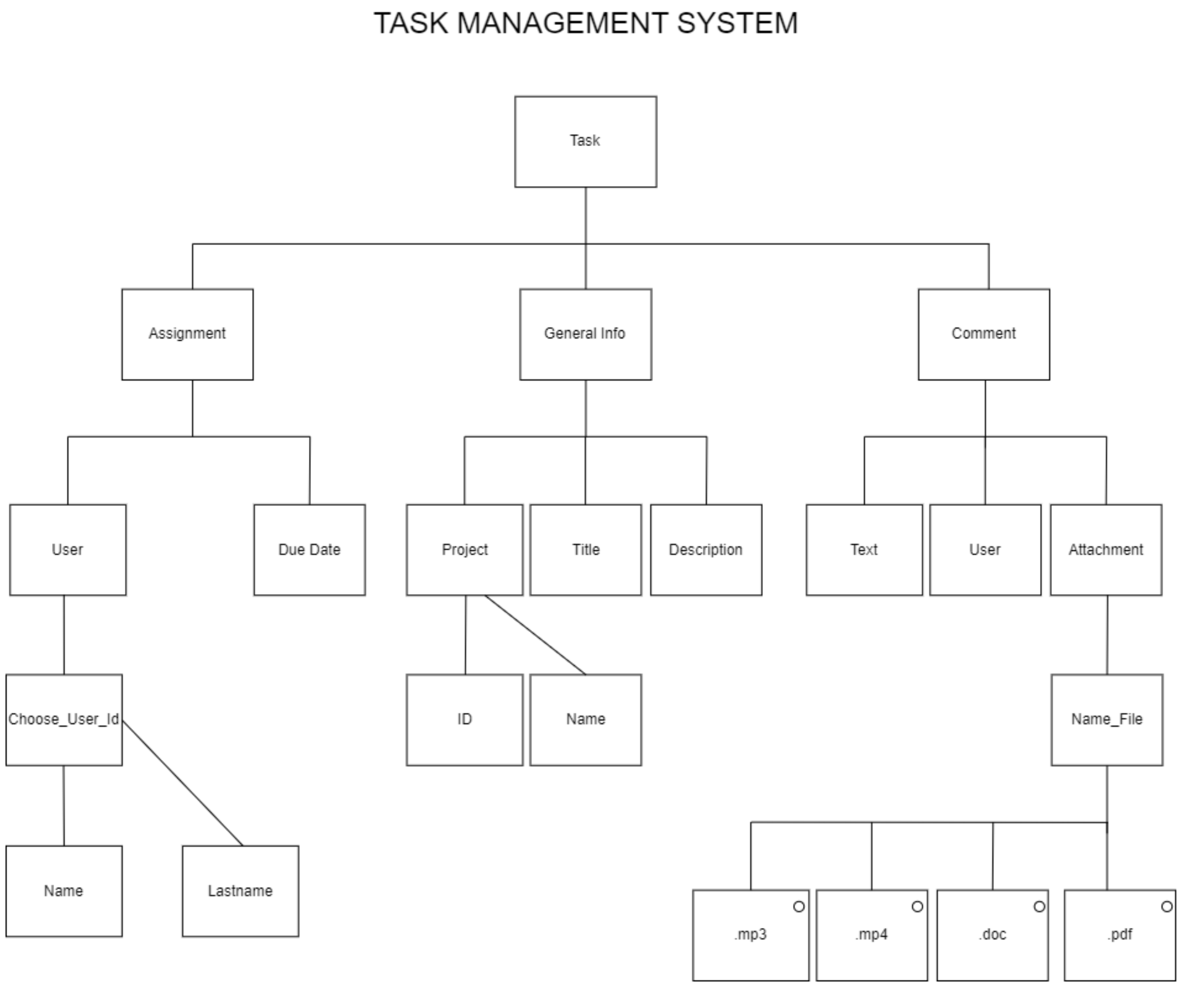

Figure 6.1. Jackson Structure Diagram

7. Bonus: Design Pattern

I choose the factory pattern for the user role scenario. For different roles, we can create different users, attaching the *showPermissions* method from the initially-defined interface. Here is the full code with the explanations as comments:

```
// Step 1: Create an interface.
public interface User {
    void showPermissions();
}

// Step 2: Create concrete classes implementing the same interface.
public class AdminUser implements User {
    @Override
    public void showPermissions() {
        System.out.println("Admin: Full Access");
    }
}

public class RegularUser implements User {
    @Override
    public void showPermissions() {
        System.out.println("Regular User: Limited Access");
    }
}

// Step 3: Create a Factory to generate object of concrete class based on given i
public class UserFactory {
    public User getUser(String userType){
        if(userType == null){
            return null;
        }
        if(userType.equalsIgnoreCase("ADMIN")){
            return new AdminUser();
        } else if(userType.equalsIgnoreCase("REGULAR")){
            return new RegularUser();
        }
        return null;
    }
}

// Step 4: Use the Factory to get object of concrete class by passing an informat
public class FactoryPatternDemo {
    public static void main(String[] args) {
        UserFactory userFactory = new UserFactory();

        User user1 = userFactory.getUser("ADMIN");
        user1.showPermissions();

        User user2 = userFactory.getUser("REGULAR");
        user2.showPermissions();
    }
}
```