Software Engineering Software Requirements Specification (SRS) Document

OnlyPets

03/29/23

2.0

By: Erin Argo, Andrea Bonola and Duncan Bruce

Table of Contents

1. Introduction

1.1. Purpose

The goal of the OnlyPets application is to allow animal lovers to share pictures of their beloved pets so that other animal lovers can admire them and show their appreciation.

1.2. Document Conventions

The purpose of this Software Requirements Document (SRD) is to document the client-view and developer-view requirements of the OnlyPets project. In it, we will keep track of the different types of users to our system as well as the requirements

1.3. Definitions, Acronyms, and Abbreviations

Java	A programming language originally developed by James Gosling at Sun Microsystems. We will be using this language to build the Restaurant Manager.
MySQL	Open-source relational database management system.
.HTML	Hypertext Markup Language. This is the code that will be used to structure and design the web application and its content.
SpringBoot	An open-source Java-based framework used to create a micro Service. This will be used to create and run our application.
MVC	Model-View-Controller. This is the architectural pattern that will be used to implement our system.
Spring Web	Will be used to build our web application by using Spring MVC. This is one of the dependencies of our system.
Thymeleaf	A modern server-side Java template engine for our web environment. This is one of the dependencies of our system.
NetBeans	An integrated development environment (IDE) for Java. This is where our system will be created.
API	Application Programming Interface. This will be used to implement a function within the software where the current date and time is displayed on the homepage.

1.4. Intended Audience

The audience is as follows; Admins, Moderators, General users. Developers Andrea Bonola, Duncan Bruce, and Erin Argo have stakes in their grades for this assignment.
For the Admins, Mods, and Users, sections 1.1, 2.1, 2.2, 2.3, 2.4, 4.2.1, 5.3
For the mods and admins: 3.2
For the developers: Everything

1.5. Project Scope

The goal of this web application is to provide a fun online community to connect with and interact with other animal lovers.

The benefits of the project include:

- ★ Users are moderated on this platform, making sure that the content remains purely in the realm of the spirit of the application, which is sharing pictures of pets.
- ★ Real sense of love and happiness as users will have access to see the content posted by other users, which means more cute animals to like!
- ★ A competitive side can be satiated as users will have the ability to rate pictures of other users' pets.

1.6. Technology Challenges

Most of the technology listed in the 1.3 definitions section is new technology for the developers. One of our developers has had to work on getting mySQL to work on their environment.

1.7. References

2. General Description

2.1. Product Perspective

OnlyPets found its origin in a pet owner's desire to share their love for their pet with the world and allow others to do the same without the constant derails and distraction present in other, more general social media platforms. The idea was originated by a pet owner for pet owners.

2.2. Product Features

The product features, starting with users, will allow for users to create accounts, post pictures of their own pets as well as rate the pictures and to rate the pictures of other community members. At the next level, moderators may timeout users they believe are not following community guidelines and the level above that, the administrator will be able to completely remove users or manipulate their content.

2.3. User Class and Characteristics

Our website application only requires users to know how to navigate web browsers. Most of the features available will be very intuitive to use.

2.4. Operating Environment

This application will be available only on the web, but should work on most if not all browsers.

2.5. Constraints

2.6. Assumptions and Dependencies

The software will be dependent on Spring Web and Thymeleaf in order to create and execute the MVC architecture that will be developed within whatever IDE each developer chooses to use.

3. Functional Requirements

3.1. Primary

- ★ FR0: The system will display a feed of pet pics posted by the community.
- ★ FR1: The system will allow the user to enter a new pet picture into the collection of already posted pet pics.
- ★ FR2: The system will allow the user to rate the pet pics in the pet feed.

3.2. Secondary

- ★ Password protection for information only accessible to users, moderators and administrators.
- ★ Authorization scheme so that users can only alter their own content and no other users' orders. Moderators can penalize users not following guidelines and only administrators can remove users completely.

4. Technical Requirements

4.1. Operating System and Compatibility

The application will be compatible with any operating system that is able to view and to interact with traditional web pages.

4.2. Interface Requirements

4.2.1. User Interfaces

The home screen will have the following elements:

- Hamburger menu in the top left; This menu will have the following anchor links (displayed as icons) that will display when the button is clicked (fixed display)
    - Home
    - Settings
    - Logout (only displayed when user is logged in)
- Sign up/Log in anchor links in the top right (fixed display)
- OnlyPets logo at the top in the center
- A wrapper in the center and slightly below the logo for the main content (Images of pets)
    - Below the pet images, buttons for rating the image 1-5

4.2.2. Hardware Interfaces

OnlyPets will run on any hardware device that has access to the internet, the ability to display webpages, and the ability to interact with webpages. This includes, but is not limited to, desktop computers, laptops, smart phones, and tablets.

4.2.3. Communications Interfaces

It must be able to connect with the internet and upload image files.

4.2.4. Software Interfaces

We will use React and Spring Boot ThymeLeaf to help build the frontend, as well as JPA for the backend database functionality. We will also use Spring Boot with Java to connect the frontend to the backend.

5. Non-Functional Requirements

5.1. Performance Requirements

- NFR0(R): The novice user will be able to create an account in less than 3 minutes
- NFR1(R): The novice user will be able to post a picture in less than 2 minutes
- NFR2(R): The novice user will be able to log in in less than a minute.
- NFR3(R): The Website will update the Database with a new user in less than 10 seconds.
- NFR4(R): The Website will update posts upon refresh and take less than 20 seconds to populate.

5.2. Safety Requirements

To prevent users posting harmful content, Moderators and admins will be able to take their content down. Moderators will be able to time users out and admins will be able to ban the users.

5.3. Security Requirements

- NFR5(R): Only authorized users will be able to use the post and rating features.
- NFR6(R): User accounts will be password protected.
- NFR7(R): Session ID tokens will be used to authenticate the user.

5.4. Software Quality Attributes

5.4.1. Availability

Ideally, as a website, OnlyPets would be available as a website 24/7 with an expected 100% uptime. However, for the project, OnlyPets is available to those who have the source code and can run it.

5.4.2. Correctness

It should be expected that the app behaves as expected in all scenarios. However, you can refer to 5.8 for more information on what can go wrong.

5.4.3. Maintainability

We will probably not maintain this app but seeing as it's using well documented frameworks and we are keeping everything as simple as possible, it should be possible to maintain it well into the future

5.4.4. Reusability

It should be highly reusable - whatever code that's specific to only pets can be scrapped and you should still be left with a template website

5.4.5. Portability

Not very portable with a heavy java backend. Should be able to be run on any modern desktop but you'd have to do some work to get it to run on anything else. The front end should be able to run in any modern browser.

5.5. Process Requirements

5.5.1. Development Process Used

Agile Model.

5.5.2. Time Constraints

A prototype must be created and ready for presentation by 3/21/2023.
The design document for the OnlyPets project must be completed by 3/16/2023.

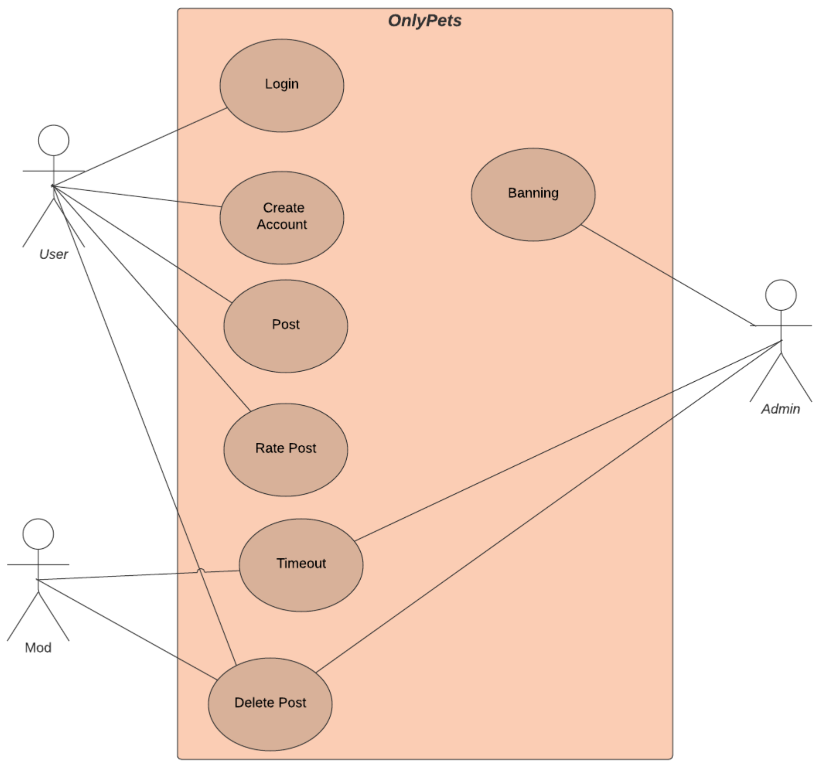
5.5.3. Cost and Delivery Date

The development cost of OnlyPets is $0.00. The delivery date for the OnlyPets web application is 4/18/2023.

5.6. Other Requirements

TBD

5.7. Use-Case Model Diagram



5.8. Use-Case Model Descriptions

5.8.1. Actor: User (Erin Argo)

- **Login**: Authenticates user
- **Post**: Posts picture to website
- **Rate Post:** Rating system for pictures, “Likes”
- **Create Account:** Creates user document (or table) in Database
- **Delete Post:** Can delete their own post
- **Report Post:** User can notify mods of a post not following guidelines
- **Password Reset:** User can reset password.

5.8.2. Actor: Moderator (Duncan Bruce)

- **Timeout**: Locks user account and prevents them from posting/rating/doing anything
- **Delete Post**: Can delete any user’s post

5.8.3. Actor: Administrator (Andrea Bonola)

- **Ban**: Prevents user from using the website
- **Hire Moderator:** Promote a user to a moderator
- **Fire Moderator:** Demote a moderator to user

5.9. Use-Case Model Scenarios

5.9.1. Actor: User (Erin Argo)

- **Use-Case Name**: Login
    - **Initial Assumption**: User is authenticated
    - **Normal**: User can use the website
    - **What Can Go Wrong: invalid username, pass**
    - **Other Activities**:
    - **System State on Completion**: Session ID is saved
- **Use-Case Name**: Post
    - **Initial Assumption**: picture is saved
    - **Normal**: Once saved, picture is displayed to socket/session
    - **What Can Go Wrong**: Corrupted image or upload doesn’t complete
    - **Other Activities**:
    - **System State on Completion**: picture is saved
- **Use-Case Name**: Rate Post
    - **Initial Assumption**: Rating is incremented in db
    - **Normal**: displayed to socket/session
    - **What Can Go Wrong**: Displayed number is incorrect because of lag or update errors
    - **Other Activities**:
    - **System State on Completion**: rating is incremented
- **Use-Case Name**: Create Account
    - **Initial Assumption**: Account is created in DB
    - **Normal**: User should be able to login/change password
    - **What Can Go Wrong**: Cannot connect to DB
    - **Other Activities**:
    - **System State on Completion**: new user is created
- **Use-Case Name**: Delete Post

- **Initial Assumption**: Post is deleted
    - **Normal**: Post is deleted
    - **What Can Go Wrong**: Request doesn't get sent or is sent multiple times or multiple posts are deleted
    - **Other Activities**:
    - **System State on Completion**: Post is deleted
- **Use-Case Name**: Report Post
    - **Initial Assumption**: Post is flagged
    - **Normal**: Flagged post is shown to moderators
    - **What Can Go Wrong**: Request doesn't get sent or is sent multiple times. User abuse
    - **Other Activities**:
    - **System State on Completion**: Flagged post is removed
- **Use-Case Name**: Password Reset
    - **Initial Assumption**: Account is inaccessible
    - **Normal**: User is able to reset password
    - **What Can Go Wrong**: User doesn't remember what email they used
    - **Other Activities**:
    - **System State on Completion**: User is sent a reset email

5.9.2. Actor: Moderator (Duncan Bruce)

- **Use-Case Name**: Timeout
    - **Initial Assumption**: User is flagged for timeout
    - **Normal**: Moderator sets a time in hours or days and user is deactivated until set time
    - **What Can Go Wrong**: DB Update Error
    - **Other Activities**:
    - **System State on Completion**: User is timed out
- **Use-Case Name**: Delete User Post
    - **Initial Assumption**: Post is deleted
    - **Normal**: Post is deleted
    - **What Can Go Wrong**: Request doesn't get sent or is sent multiple times or multiple posts are deleted
    - **Other Activities**:
    - **System State on Completion**: User Post is deleted

5.9.3. Actor: Administrator (Andrea Bonola)

- **Use-Case Name**: Ban
    - **Initial Assumption**: User is banned
    - **Normal**: User is denied access to the website
    - **What Can Go Wrong**: DB Update Error
    - **Other Activities**:
    - **System State on Completion**: User is banned
- **Use-Case Name**: Hire Moderator
    - **Initial Assumption**: User is promoted to Moderator
    - **Normal**: User is granted moderator privileges
    - **What Can Go Wrong**: DB Update Error
    - **Other Activities**:

        - **System State on Completion**: User is promoted to Moderator
    - **Use-Case Name**: Fire Moderator
        - **Initial Assumption**: Moderator is demoted to user
        - **Normal**: User loses moderator privileges
        - **What Can Go Wrong**: DB Update Error
        - **Other Activities**:
        - **System State on Completion**: User regains user status

6. Design Documents

6.1. Software Architecture

6.1.1. Erin Did this one!!

Have a class for each actor

Admin

Moderator

User

For each class include every use case

- AdminController extends ModeratorController
    - Ban
    - hireModerator
    - fireModerator
- ModeratorController extends UserController
    - Timeout
- User
    - Constructor
- UserController
    - getAllUsers
    - getUserById
    - createUser
    - resetPassword
    - authenticateUser
    - getUserByUsername
- UserRepository
    - Extends jparepo
- UserService
    - getAllUsers
    - getUserById
    - getUserByUsername
    - save

Then we need a class for posts

- Post
    - Constructor
- PostController
    - createPost

    - deletePostById
    - ratePostById
    - reportPostById
- PostRepository
    - Extends jparepo
- PostService
    - getAllPosts
    - getPostById
    - getPostByAuthorId
    - getPostByAuthorUsername
    - deletePostById
    - save

And we also need a class for our controller and we'll just call it RequestController

- RequestController
    - getHome
    - getRegister
    - getLogin
    - getDashboard
    - getPasswordReset
    - Register
    - Post
    - Like
    - Dislike
    - Report
    - Ban
    - Hire
    - Fire
    - Timeout
    - handleFileUpload

For every tier above user we already wrote the logic for the previous use cases, there's no need to write it again, simply import the class below it. For deletepost on user and anyone above user we just need to write an if statement along the lines of if(!(user.is_moderator || user.is_admin) || post.author !== user.id) return;
And similar for writing the moderator use cases if(!user.moderator || !user.admin) return;
requestController handles everything and makes calls to the appropriate services when necessary

Assume that we're using getters and setters and the variables are all protected. Getters should just be getAllUsers and getAllPosts as well as getUserById and getPostById

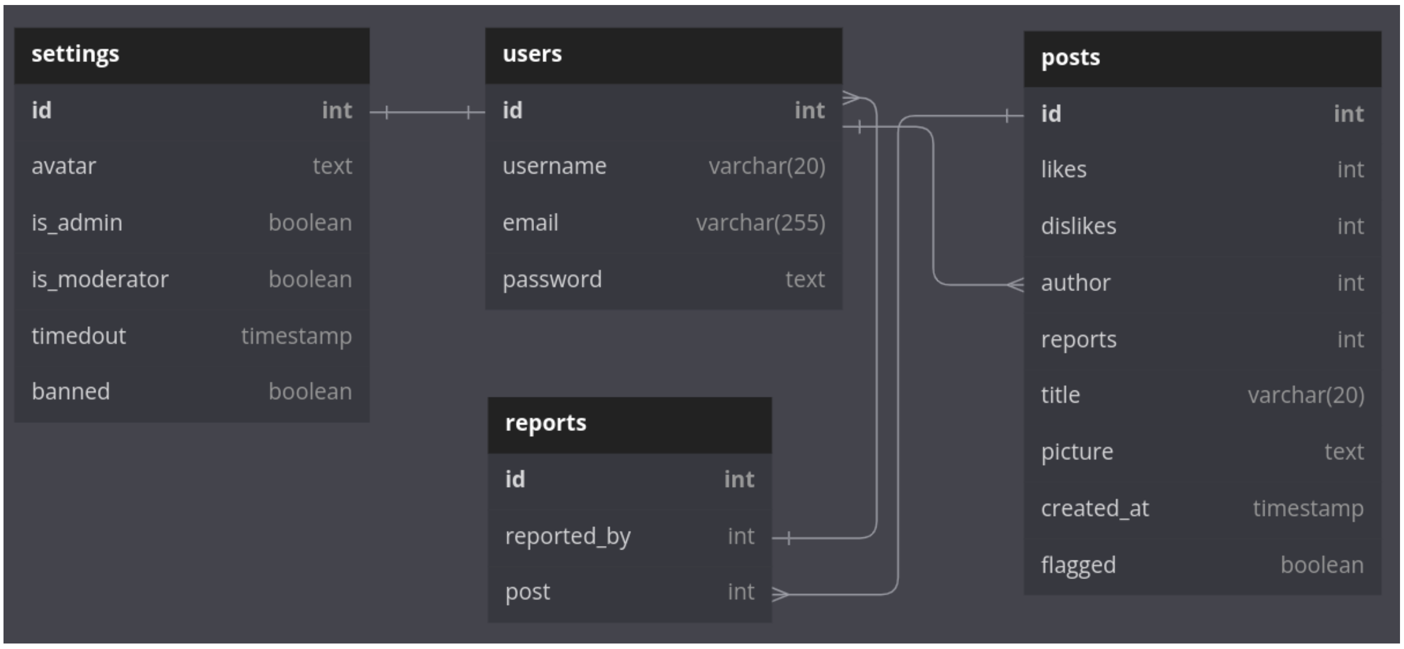
As for models, the current architecture we have defined in our use cases allows for user creation and if we want them to be an admin or moderator later we would have to manually change that with an admin dashboard. Admins have to be added manually into sql but that's okay as there will only be the three of us

The models are covered in 6.2 with our DB structure

6.2. High-Level Database Schema

6.2.1. Erin Also Did This One!!

We currently have two ways of storing data. One is in a Database and the other is by file system (file DB).



Database Storage

- Users
    - id int [pk, increment] || Autogenerated
    - username varchar(20)
    - email varchar(255)
    - password text
- Posts
    - id int [pk, increment]
    - likes int
    - dislikes int
    - author int
    - reports int [default: 0]
    - title varchar(20)
    - picture text [note: 'href to files assets/pets']
    - created_at timestamp
        - The relationship between posts.author and user.id is many posts to one user which should allow for multiple posts for user but each post can only have one author
    - flagged boolean [default: false]
- Settings
    - id int [pk, note: 'same as user id']
    - avatar text [note: 'href to files assets/avatars']
    - is_admin boolean [default: false]
    - is_moderator boolean [default: false]
    - timedout timestamp [default: 0]
    - banned boolean [default: false]
- Reports
    - id int [pk, increment]
    - reported_by int

    - post int

File System DB

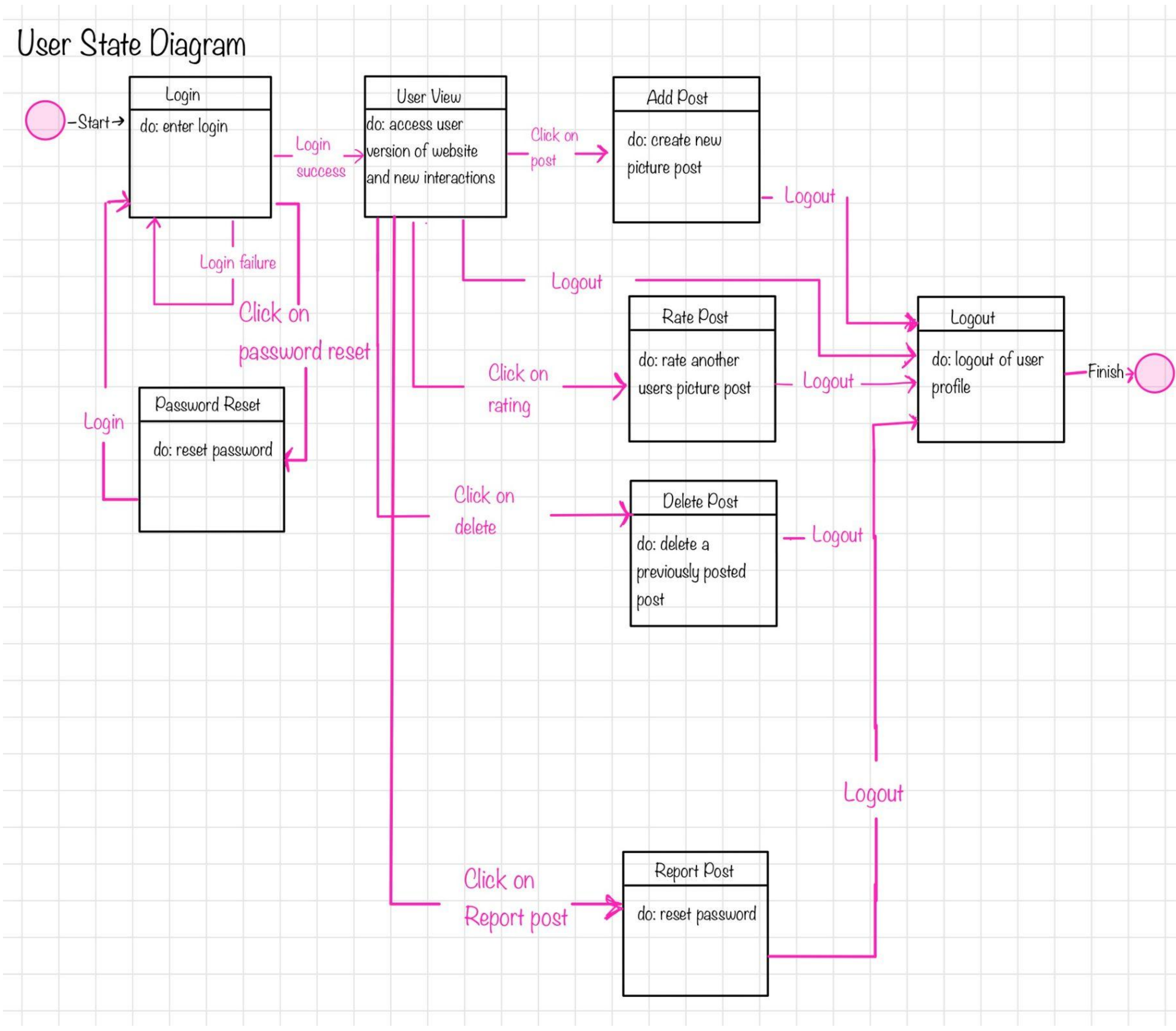
- Avatars
- Icons
- Pets (pictures for posts)

```
._src
|_main
|___Resources
|____Templates
|_____…
|____Static
|_____…
|_____assets
|______avatars
|_______…
|______pets
|_______…
|______icons
|_______…
```

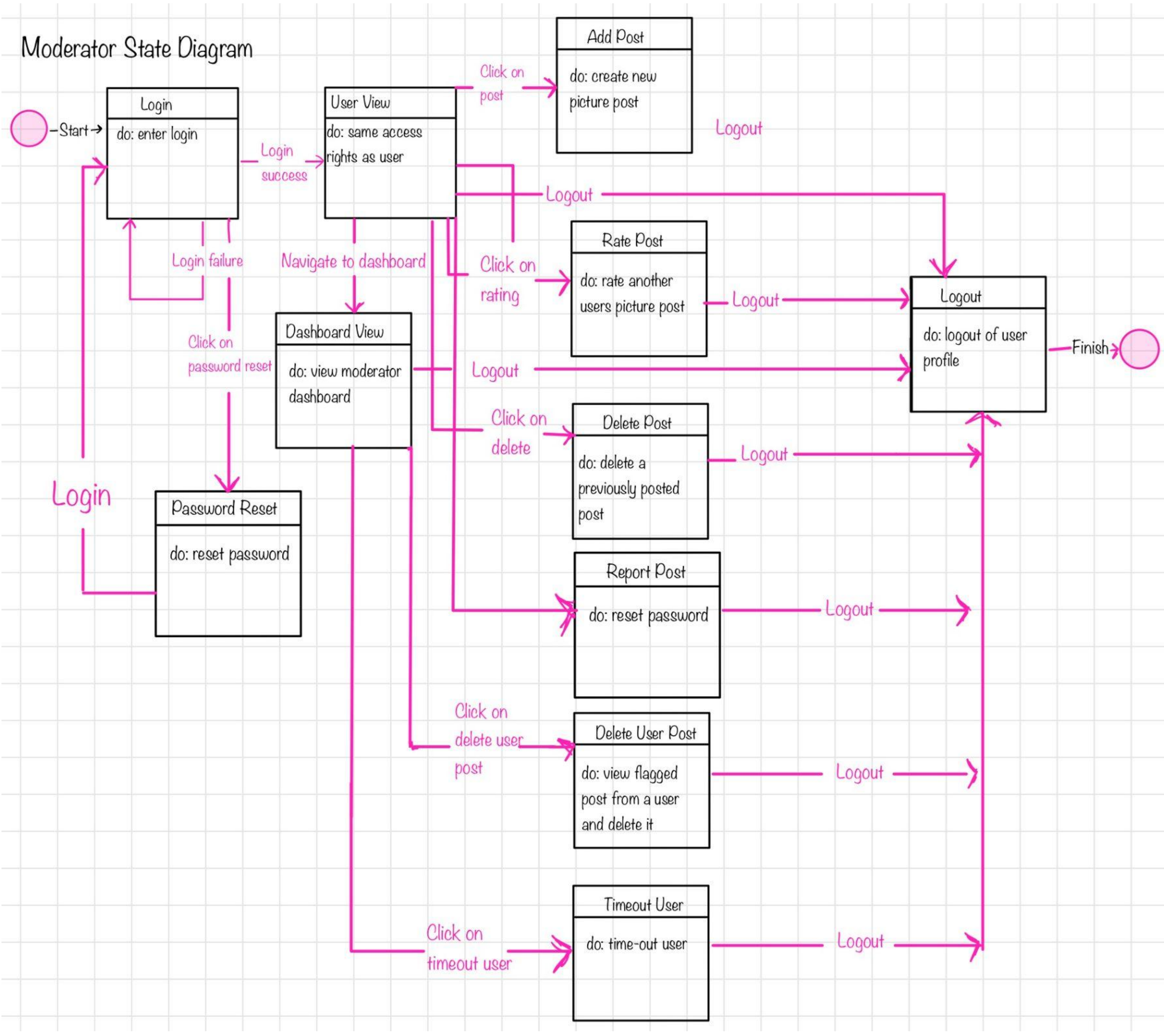
We have an intermediary reports field in the database because we want to limit the number of reports per user per post, this is a simple and effective way to keep track of that.

6.3. Software Design (Andy did this)

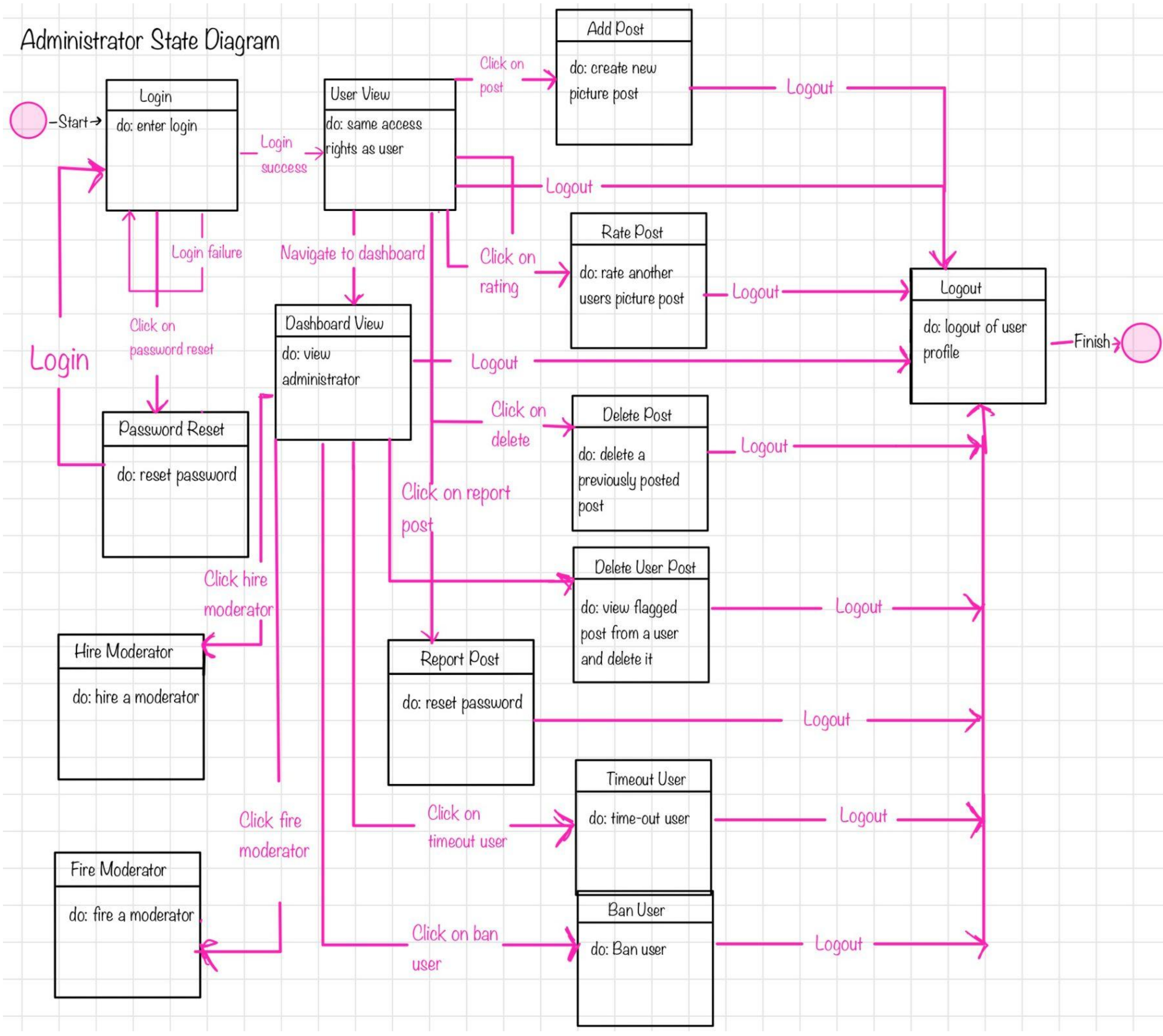
6.3.1. State Machine Diagram: User (Andrea Bonola)



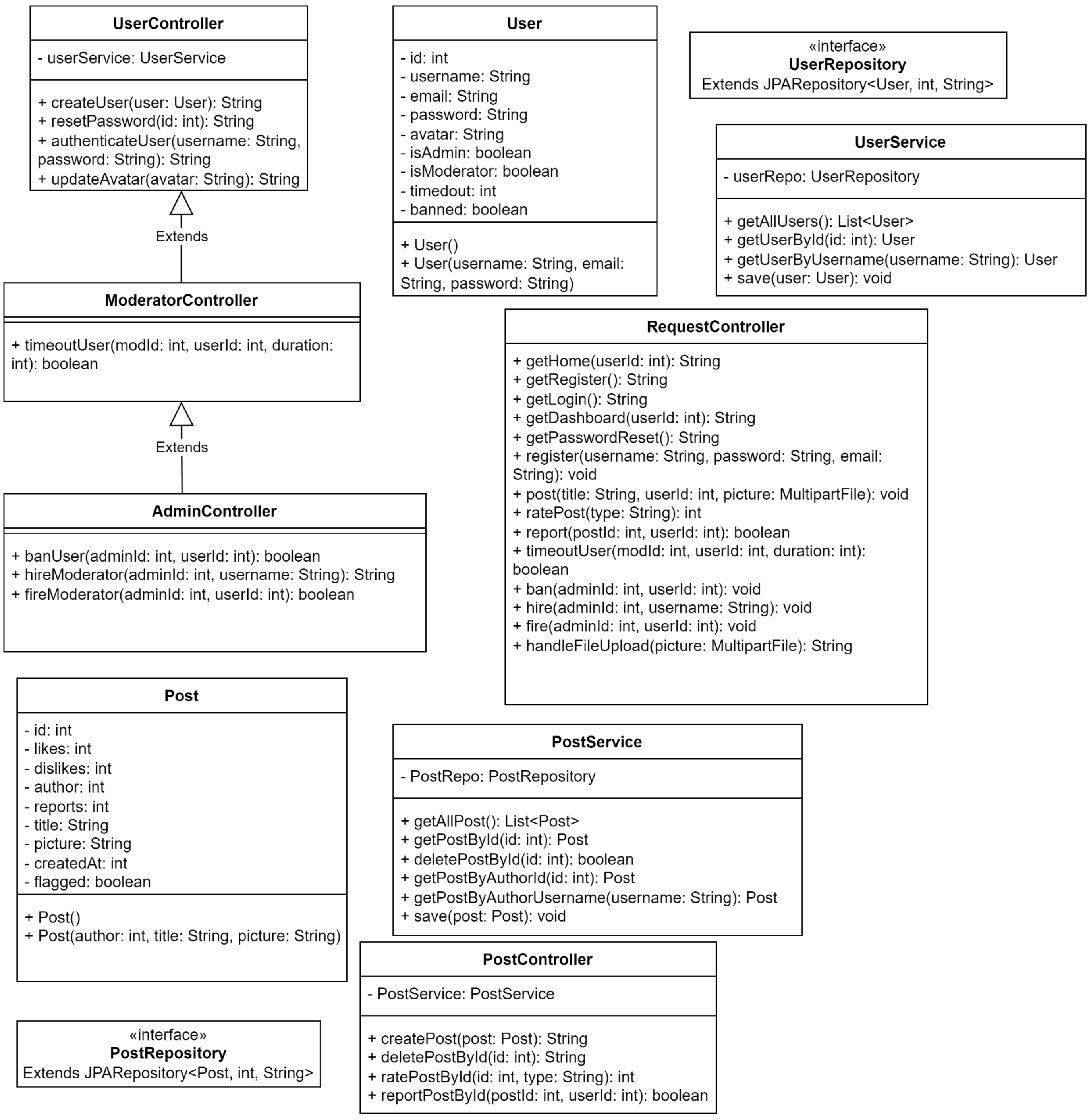
State Machine Diagram: Moderator (Duncan Bruce)



6.3.3. State Machine Diagram: Administrator (Erin Argo)



6.4. UML Class Diagram (Duncan did this)


UserController
- userService: UserService
+ createUser(user: User): String
+ resetPassword(id: int): String
+ authenticateUser(username: String, password: String): String
+ updateAvatar(avatar: String): String
Extends
ModeratorController
+ timeoutUser(modId: int, userId: int, duration: int): boolean
Extends
AdminController
+ banUser(adminId: int, userId: int): boolean
+ hireModerator(adminId: int, username: String): String
+ fireModerator(adminId: int, userId: int): boolean
User
- id: int
- username: String
- email: String
- password: String
- avatar: String
- isAdmin: boolean
- isModerator: boolean
- timedout: int
- banned: boolean
+ User()
+ User(username: String, email: String, password: String)
«interface»
UserRepository
Extends JPARepository<User, int, String>
UserService
- userRepo: UserRepository
+ getAllUsers(): List<User>
+ getUserById(id: int): User
+ getUserByUsername(username: String): User
+ save(user: User): void
RequestController
+ getHome(userId: int): String
+ getRegister(): String
+ getLogin(): String
+ getDashboard(userId: int): String
+ getPasswordReset(): String
+ register(username: String, password: String, email: String): void
+ post(title: String, userId: int, picture: MultipartFile): void
+ ratePost(type: String): int
+ report(postId: int, userId: int): boolean
+ timeoutUser(modId: int, userId: int, duration: int): boolean
+ ban(adminId: int, userId: int): void
+ hire(adminId: int, username: String): void
+ fire(adminId: int, userId: int): void
+ handleFileUpload(picture: MultipartFile): String
Post
- id: int
- likes: int
- dislikes: int
- author: int
- reports: int
- title: String
- picture: String
- createdAt: int
- flagged: boolean
+ Post()
+ Post(author: int, title: String, picture: String)
PostService
- PostRepo: PostRepository
+ getAllPost(): List<Post>
+ getPostById(id: int): Post
+ deletePostById(id: int): boolean
+ getPostByAuthorId(id: int): Post
+ getPostByAuthorUsername(username: String): Post
+ save(post: Post): void
PostController
- PostService: PostService
+ createPost(post: Post): String
+ deletePostById(id: int): String
+ ratePostById(id: int, type: String): int
+ reportPostById(postId: int, userId: int): boolean
«interface»
PostRepository
Extends JPARepository<Post, int, String>

7. Scenario

7.1. Brief Written Scenario with Screenshots