Project 5

Team members: Srinivas Akhil Mallela, Likhitha Katakam**,** Vishal Prabhachandar
Section: CSCI 5448-001

Project Summary

Title: StackUnderFlow - A discussion board
Team members: Srinivas Akhil Mallela, Likhitha Katakam, Vishal Prabhachandar
Overview:
We intend to build a discussion board where one can post, answer and search for questions. The application will contain a registration and sign-in option to create a profile and store the details securely. Once the profile is created and logged in, the user can view a dashboard of stats along with actions performed till date and have a provision to add new questions or discussion threads. The application will also contain a view of all the active threads where the user can access them, answer, upvote or delete details. The core idea is to build a boiled down version of a discussion board similar to that of StackOverflow.

Techstack:

Language - Java

Frontend -
HTML5 - For discussion board skeletal components
Materialize - Responsive CSS framework for beautification
Thymeleaf - Modern server-side Java template engine to render components

Backend -
Spring Framework and its components
- Spring Boot - To configure and launch the application
- Java Persistence API(JPA) in Spring data - To perform CRUD operations
- Spring Security - For login and authentication of users
- Spring MVC - For creating REST endpoint to perform program logic

Data Source -

MySQL - To store the content and other user-related details

Project Requirements

The application responsibility can be broken down into three major components:

1. User management
2. User interface/User experience
3. Orchestration/flow logic

User management

The user management of in the application can further split into three components:

1. Authentication
2. Authorization
3. Secure storage

Authentication

The application will contain an register and login page from where the user can gain access to the discussion board. The registration page will allow the user to create his/her username, password and quick introduction. After which they will be able to login to the system. The authentication details of the user is stored in a secured format leveraging the Spring security framework.

Authorization

The application will be built in such a way to not allow unauthorized access to the discussion board. The user is required to create a profile and login to be able to post, answer questions and partake in other activities. This is done to assure secure access and mediate control.

Secure Storage

Since the Username and Password are sensitive details of a user that are to be stored securely, we will be using the Spring security framework to hash and store the details in a save way.

User interface/ User experience

The user interface/user experience responsibility of the application can be broken down into 4 sections.

Login/Register screen

The registration screen will present to the user a form like interface, where he/she can enter in their username, password and introduction to create an account.
The login screen will provide an interface for the user to type in their username and password to login to their application.

Dashboard

Once the user logs in, he/she is presented with a small dashboard containing personal information and activity statistics. Below which is a provision for the user to create a new topic or category for discussion.

Discussion overview screen

The user can navigate to the discussion overview screen, to look at the list of all the current discussions, with condensed details overview like title, author and date created.

Topic view

From the discussion overview screen the user will be able to click and view a discussed topic. This view will provide all teh details of the discussion, like questions and various answers. After which the user can add a comment, delete a comment or upvote a particular answer.

Orchestration/flow logic

The application is built using Spring boot, leveraging JPA to perform CRUD operations, Spring MVC for REST endpoints, Spring security for authentication and authorization. MySQL is used as the data store to store all relevant information using JPA.
The application is built to accommodate multiple users and functions in a safe and secured manner. The goal of the application is to provide a seamless user experience for discussing threads.

Registration/Login logic

Leveraging the Spring security feature we will control the registration and login of the user in the application. The application will take in the username, password and introduction details from the front end as input. Then hash the password value and store the user details in the DB securely. Also flow control restrictions are put in place to not let the user access the application if not logged in. The scenario where the user has not registered but tries to login is also taken into account and the user is redirected to the registration page to setup an account.

Dashboard

The dashboard is broadly split into three components personal information, discussion stats and provision to create new topic. The personal information component shows details like username, introduction and date of joining. The discussion statistic component shows details like number of answered question and questions asked. Finally the topic creation section allows user to chose a topic to post a question on and add details to post the question, which is then stored in DB with all relevant details.

Discussion list logic

The discussion list view shows all the current or particular domain discussion threads as a list view to the user after fetching the relevant information from the database. The list view contains details like topic title, user who created the topic, time of creation and a button to expand the topic discussion.

Topic view logic

The expanded topic view first fetches all the details with respect to that particular discussion like the question, answer, upvotes and renders the detail on the front end. Following this the user has a provision to add an answer to the question, upvote an answer, modify or delete his own answer. All of these actions and data changes are accordingly reflected in the backend.

Logout

When the logout button is clicked by the user, the current user session is expunged and the view returns back to the landing page

Use case

The application is designed to support multiple users to independently use the application. There are no definitve distinction between the users like a super user and a casual user. Every individual is treated independently. The users can leverage the application to discuss, answer and upvote questions.

Registration use case

When the user accesses the system, he/she is presented with the landing page, from where they can chose to register or login.
The registration form will provide means for user to register using Spring security and handle different requirements and conditions.
The login screen will provide user to add username and password details to login to the application. In case the user does not exist the user is redirected to the registration page.
Possible issues that we can run into while building this portion of the application is not being able to integrate the Spring security logic with the application flow control or not being able to connect and store the details in DB. However, work arounds are testing components individually and integrating in a phase wise manner.

Dashboard use case

After the user logs into the application, he is presented with a dashboard containing personal information, statistics of discussion till date and a provision to add new threads for discussion. The entire orchestration with regards to loading the details from the data store and to present in the view and control the flow is done leveraging Java persisten API and Spring MVC framework. The application in itself is run using Spring boot.
The possible issues that we can run into include the inability to connect and fetch data from the data store or connectivity issues or sessions not maintained properly. Possible work arounds include building the components in an incremental fashion, writing unit tests covering all possible scenarios and testing thoroughly integrating everything and finally have proper error handling mechanism to debug through various issues.

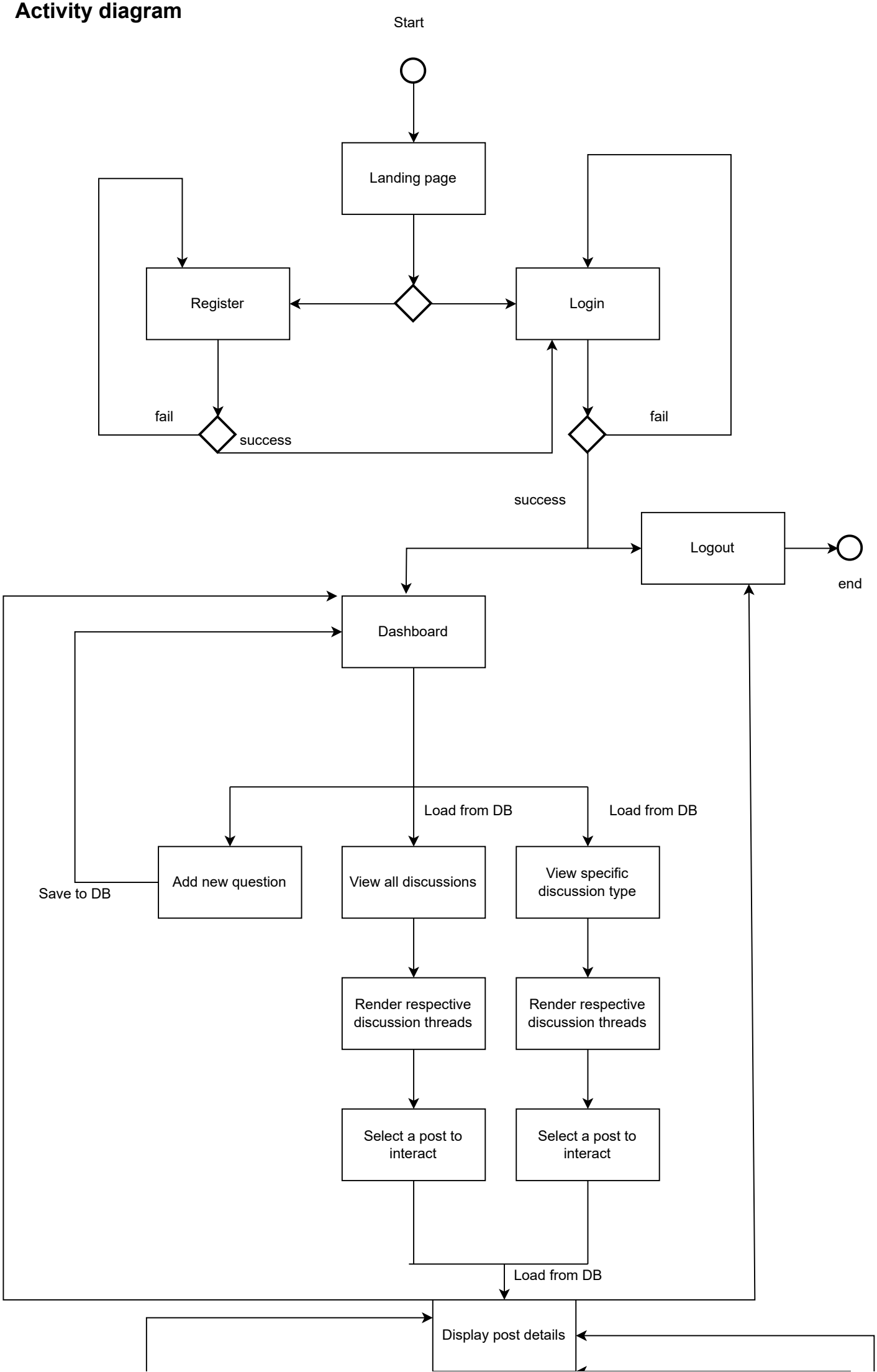
Topic view and interaction use case

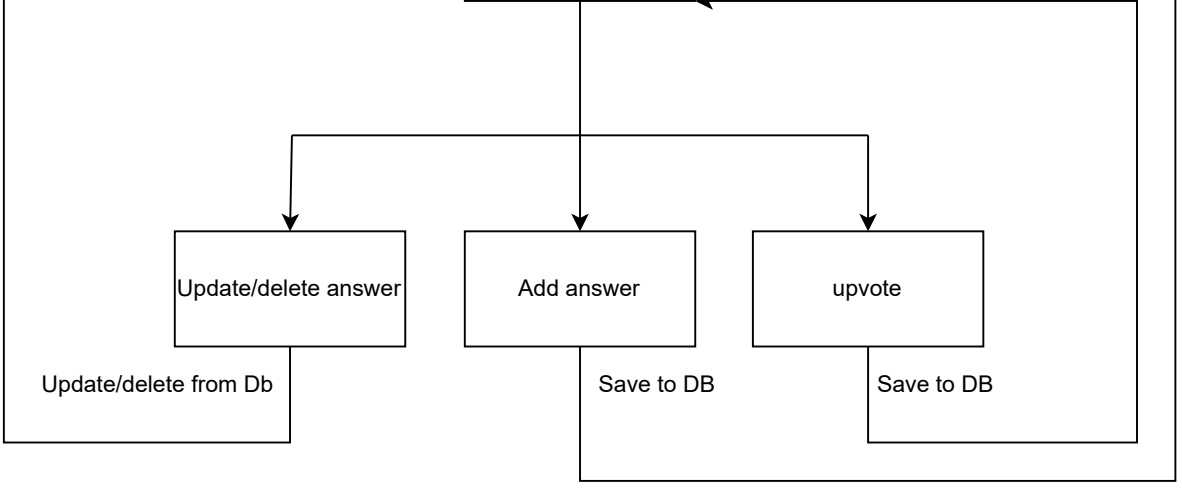
From the dashboard the user can choose to view all the threads in the discussion board or chose a particular category to inspect. A view is present with all the discussion threads, which the user can chose to expand and interact. In that particular thread the user can chose to view, answer the questions and upvote answers. The loading of data from the data store is done using Java persistence API and all the endpoints for interaction and updates are done using REST endpoints exposed using the Spring MVC framework into the data store. Possible issues can again involve inability of interaction between components and best approach to work around is to build through the application incrementally integrating it. Best approaches would also involve having proper unit test cases and testing as the application builds through the various software development cycle.

Other project 5 diagrams

https://github.com/vishalprabha/CSCI5448-OOAD/tree/main/Projects/Project5

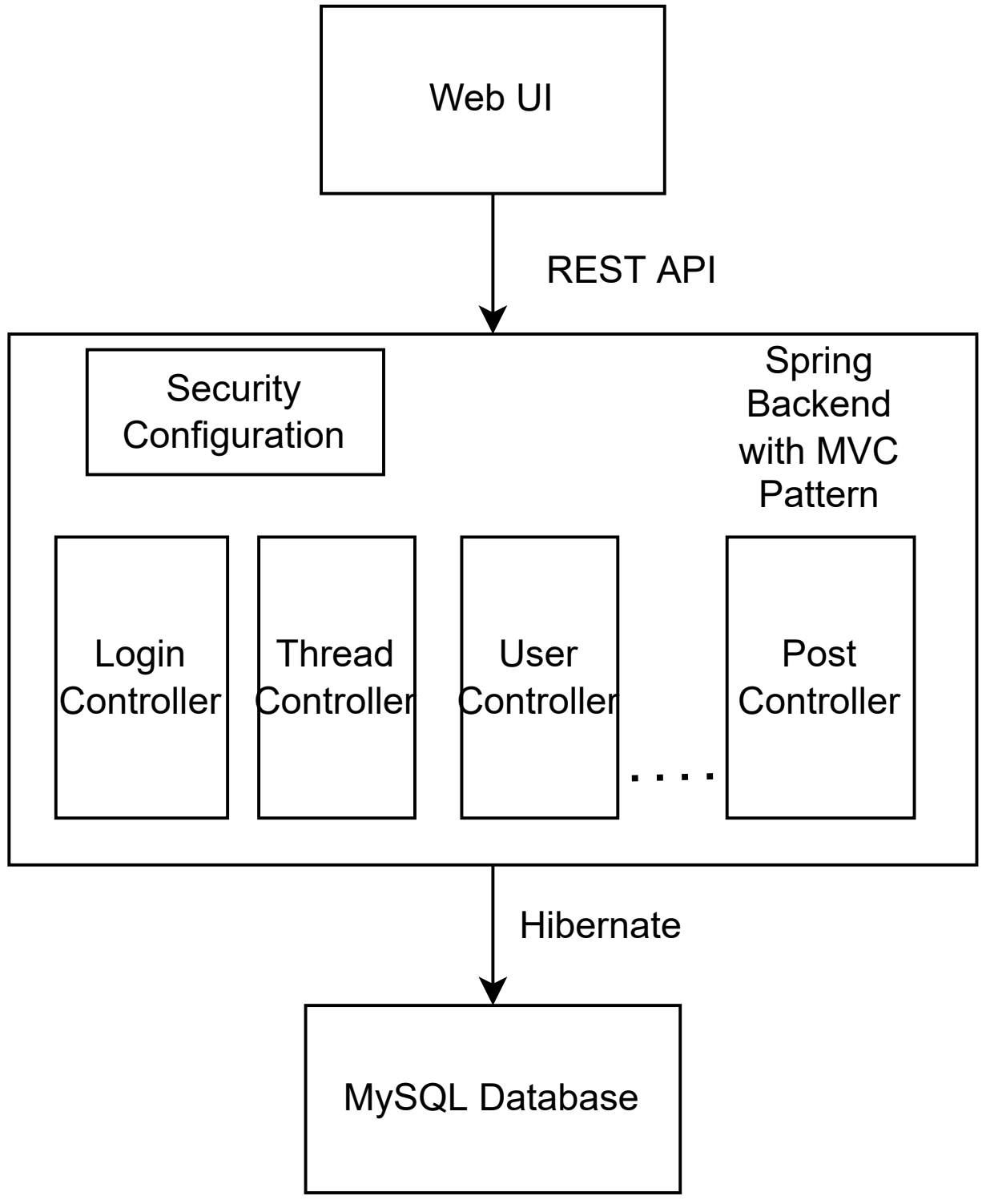
Activity diagram



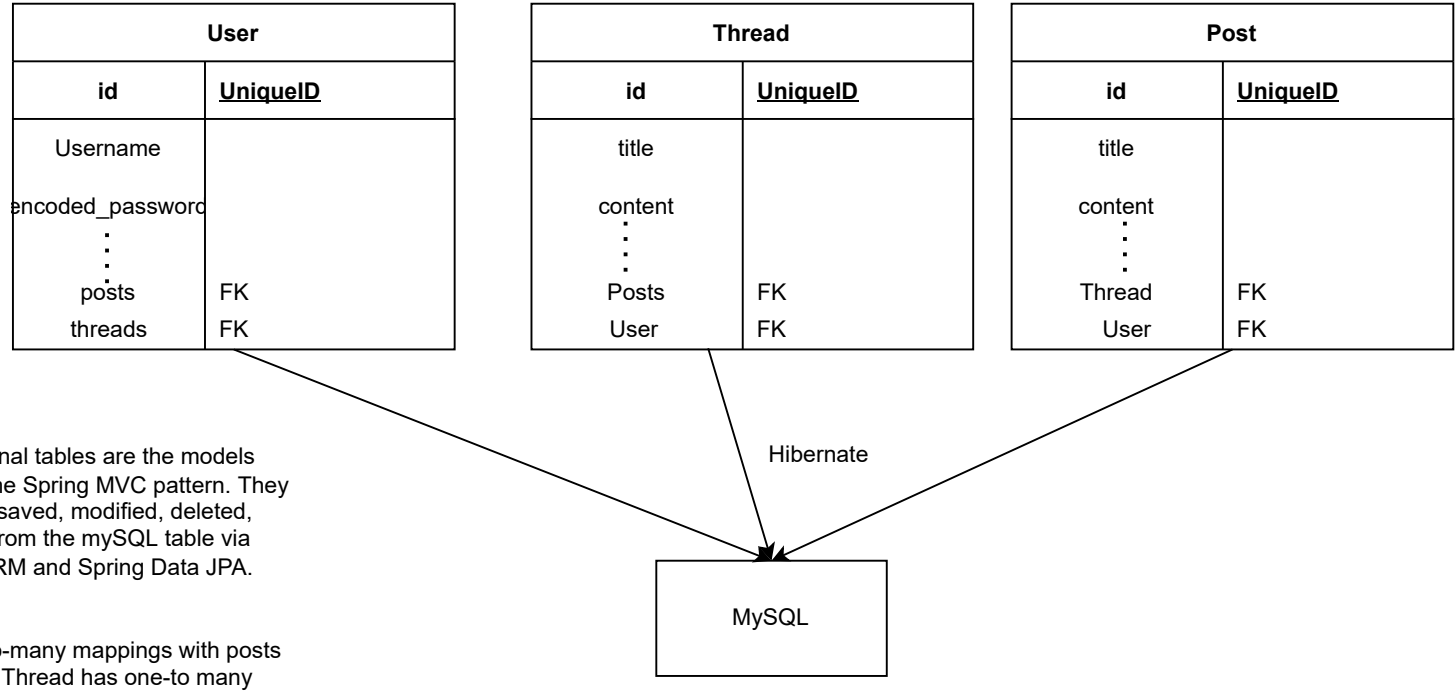

Update/delete answer
Add answer
upvote
Update/delete from Db
Save to DB
Save to DB

Architecture Diagram

The web pages will be rendered via rest calls and login authentication and password encoding will be handled by spring security. Spring MVC will be used across all the user profiles, threads and posts classes and the model is saved to the MySQL DB via Hibernate ORM by respective class registers.

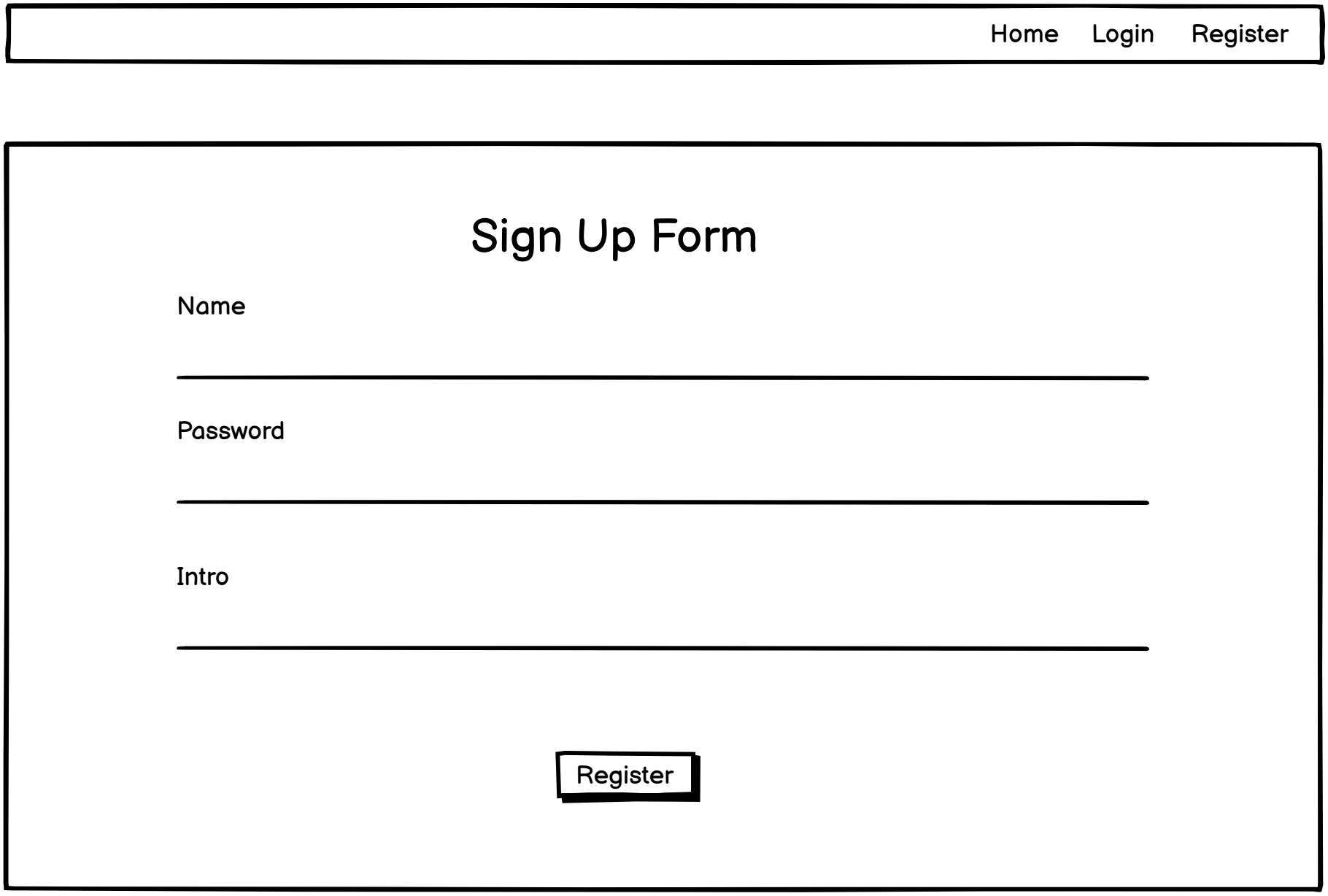


Data Storage Diagram



All the relational tables are the models represented in the Spring MVC pattern. They are created, saved, modified, deleted, retrieved to/from the mySQL table via Hibernate ORM and Spring Data JPA.

User has one-to-many mappings with posts and threads. Thread has one-to many mapping with posts and many-to-one mapping with user. Post has many-to-one mapping with thread and user.


Home
Login
Register
Sign Up Form
Name
Password
Intro
Register

Dashboard

Home Profile Logout

Profile Details

Name Nick
Intro Student
Joined 03.24.2022
Threads 3
Answers 4

Add thread

Category

Title

Content

Submit

All threads

Posts	Thread	Author	Created
1	MVC pattern	Nick	04:15 - 03.01.2022
1	Java8	Mary	01:10 - 03.24.2022
2	Dependency Injection	Lucas	12:10 - 03.24.2022
3	TDD	Max	08:10 - 03.24.2022

PostsView

Home Profile Logout

MVC Pattern

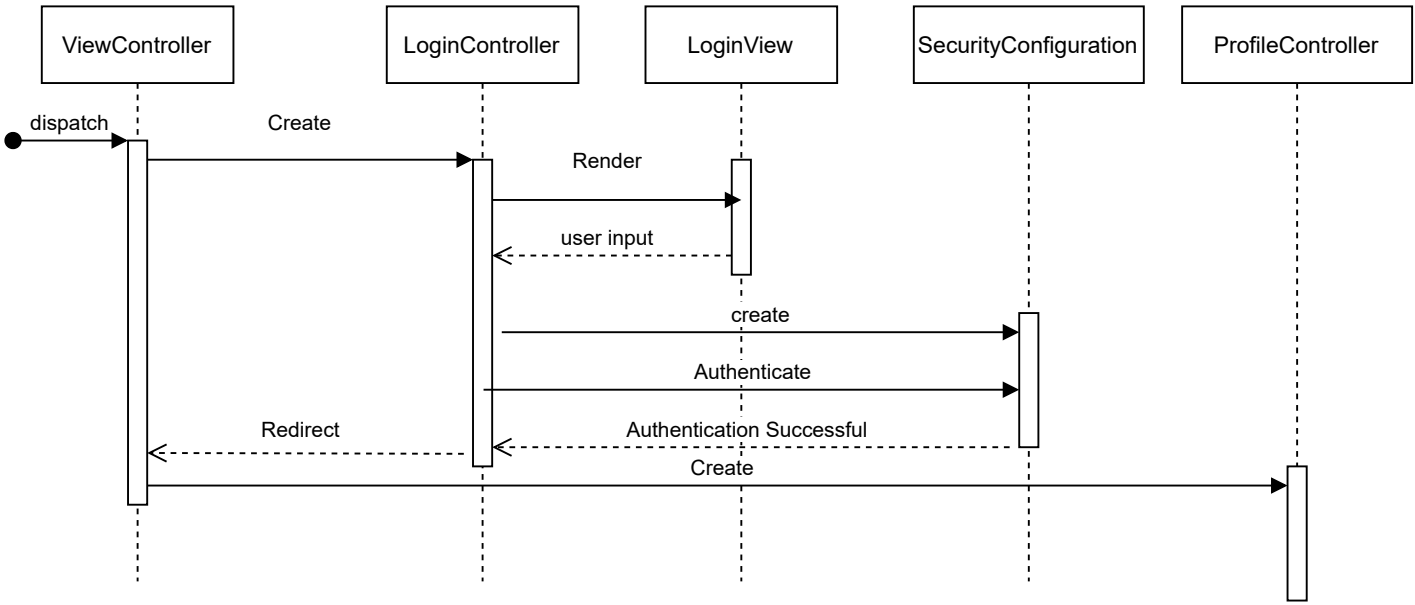
MVC Pattern stands for Model-View-Controller Pattern. This pattern is used to separate application's concerns.
Model - Model represents an object or JAVA POJO carrying data. It can also have logic to update controller if its data changes.
View - View represents the visualization of the data that model contains.
Controller - Controller acts on both model and view. It controls the data flow into model object and updates the view whenever data changes. It keeps view and model separate.

Useful

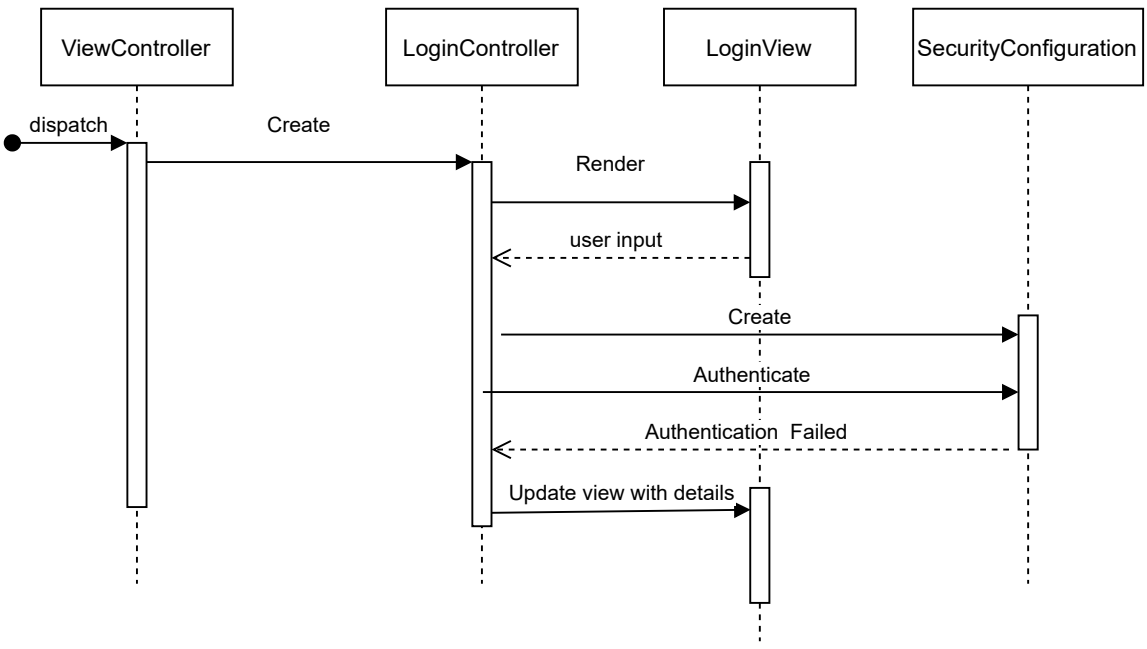
04:15 - 03.01.2022

User Interaction Diagram - 1

Login Successful Sequence

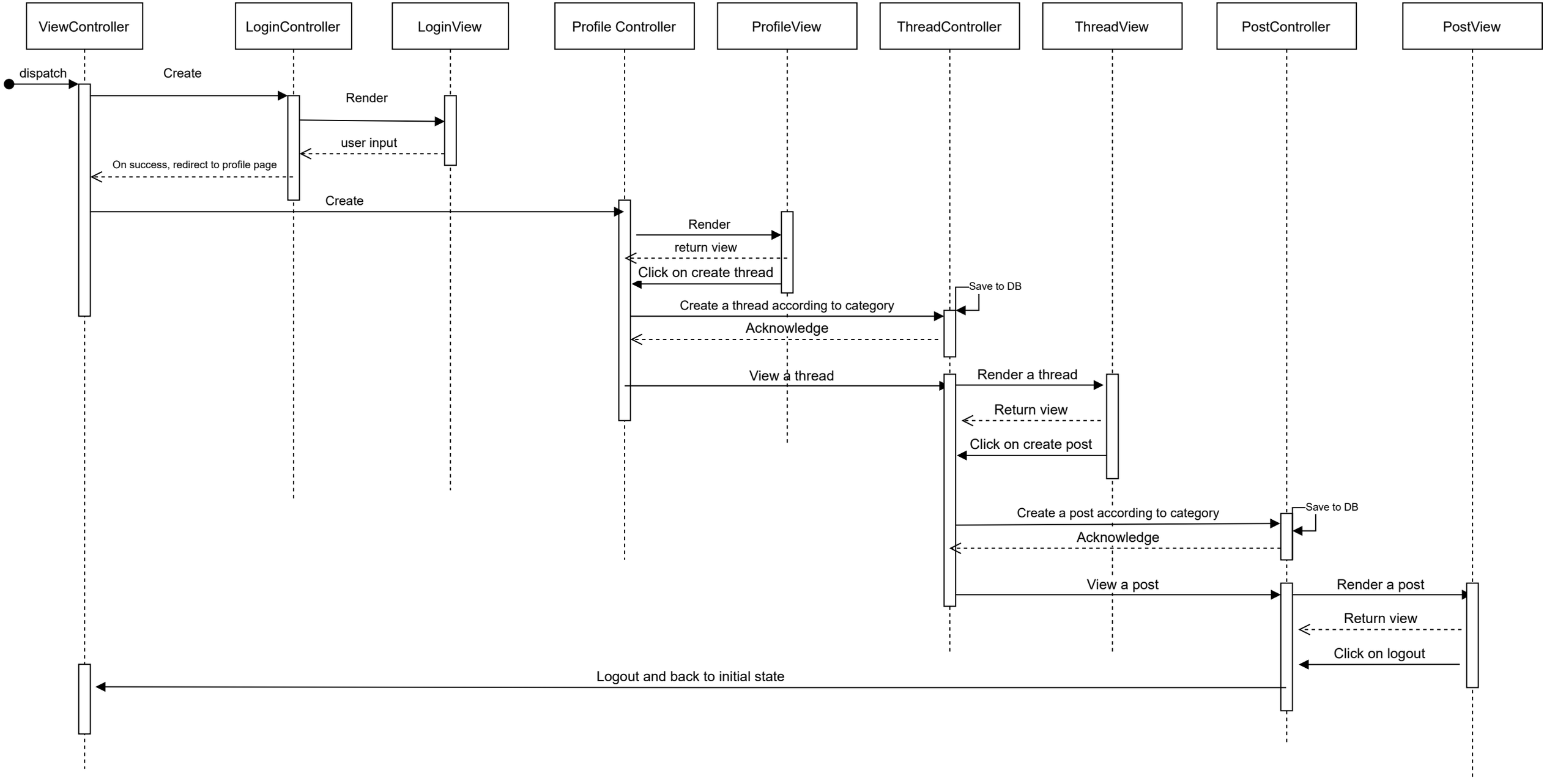


Login Failure Sequence



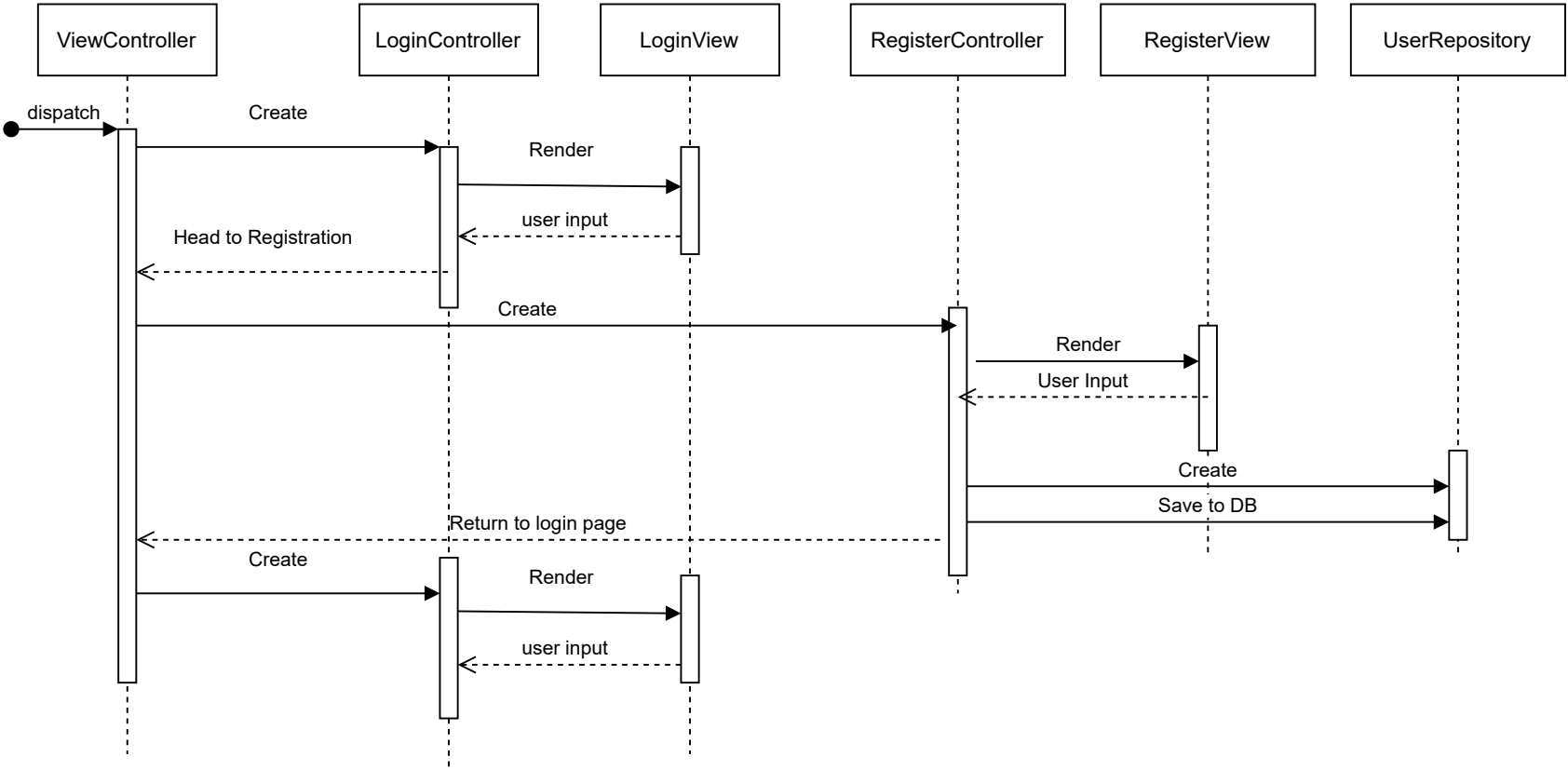
User Interaction Diagram - 3

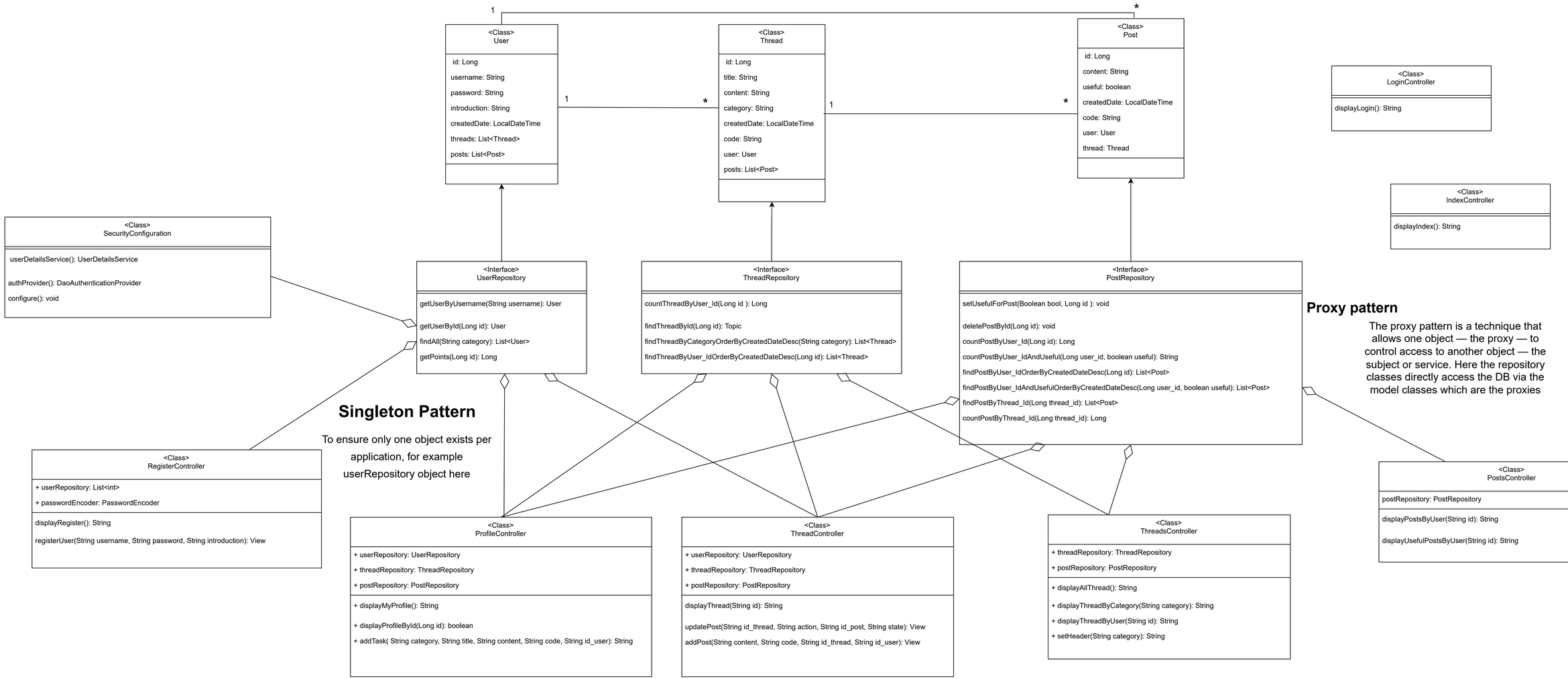
Happy flow of logging in and creating a post and a thread



User Interaction Diagram - 2

Registration Sequence




1
*
<Class>
User
id: Long
username: String
password: String
introduction: String
createdDate: LocalDateTime
threads: List<Thread>
posts: List<Post>
1
*
<Class>
Thread
id: Long
title: String
content: String
category: String
createdDate: LocalDateTime
code: String
user: User
posts: List<Post>
1
*
<Class>
Post
id: Long
content: String
useful: boolean
createdDate: LocalDateTime
code: String
user: User
thread: Thread
<Class>
LoginController
displayLogin(): String
<Class>
IndexController
displayIndex(): String
<Class>
SecurityConfiguration
userDetailsService(): UserDetailsService
authProvider(): DaoAuthenticationProvider
configure(): void
<Interface>
UserRepository
getUserByUsername(String username): User
getUserById(Long id): User
findAll(String category): List<User>
getPoints(Long id): Long
<Interface>
ThreadRepository
countThreadByUser_Id(Long id): Long
findThreadById(Long id): Topic
findThreadByCategoryOrderByCreatedDateDesc(String category): List<Thread>
findThreadByUser_IdOrderByCreatedDateDesc(Long id): List<Thread>
<Interface>
PostRepository
setUsefulForPost(Boolean bool, Long id): void
deletePostById(Long id): void
countPostByUser_Id(Long id): Long
countPostByUser_IdAndUseful(Long user_id, boolean useful): String
findPostByUser_IdOrderByCreatedDateDesc(Long id): List<Post>
findPostByUser_IdAndUsefulOrderByCreatedDateDesc(Long user_id, boolean useful): List<Post>
findPostByThread_Id(Long thread_id): List<Post>
countPostByThread_Id(Long thread_id): Long
Proxy pattern
The proxy pattern is a technique that allows one object — the proxy — to control access to another object — the subject or service. Here the repository classes directly access the DB via the model classes which are the proxies
Singleton Pattern
To ensure only one object exists per application, for example userRepository object here
<Class>
RegisterController
+ userRepository: List<int>
+ passwordEncoder: PasswordEncoder
displayRegister(): String
registerUser(String username, String password, String introduction): View
<Class>
ProfileController
+ userRepository: UserRepository
+ threadRepository: ThreadRepository
+ postRepository: PostRepository
+ displayMyProfile(): String
+ displayProfileById(Long id): boolean
+ addTask(String category, String title, String content, String code, String id_user): String
<Class>
ThreadController
+ userRepository: UserRepository
+ threadRepository: ThreadRepository
+ postRepository: PostRepository
displayThread(String id): String
updatePost(String id_thread, String action, String id_post, String state): View
addPost(String content, String code, String id_thread, String id_user): View
<Class>
ThreadsController
+ threadRepository: ThreadRepository
+ postRepository: PostRepository
+ displayAllThread(): String
+ displayThreadByCategory(String category): String
+ displayThreadByUser(String id): String
+ setHeader(String category): String
<Class>
PostsController
postRepository: PostRepository
displayPostsByUser(String id): String
displayUsefulPostsByUser(String id): String
Mediator Pattern
The controllers in our MVC model act as the mediator between models and views. They limit direct communication between the different components in the system and control the flow
<Class>
ForumApplication
+ main() : static void
Model Details
User, Thread, Post
MVC
AnswersComtroller, IndexController, LoginController, ProfileController, RegisterController, ThreadController, ThreadsController
Controller Details
RegisterView, TopicsView, ProfileView, TopicView, LoginView, indexView
View Details