User

<Class>
User

- id: Long
- username: String
- password: String
- introduction: String
- createdDate: LocalDateTime
- threads: List<Thread>
- posts: List<Post>

Thread

<Class>
Thread

- id: Long
- title: String
- content: String
- category: String
- createdDate: LocalDateTime
- code: String
- user: User
- posts: List<Post>

Post

<Class>
Post

- id: Long
- content: String
- useful: boolean
- createdDate: LocalDateTime
- code: String
- user: User
- thread: Thread

LoginController

<Class>
LoginController

- displayLogin(): String

IndexController

<Class>
IndexController

- displayIndex(): String

SecurityConfiguration

<Class>
SecurityConfiguration

- userDetailsService(): UserDetailsService
- authProvider(): DaoAuthenticationProvider
- configure(): void

UserRepository

<Interface>
UserRepository

- getUserByUsername(String username): User
- getUserById(Long id): User
- findAll(String category): List<User>
- getPoints(Long id): Long

ThreadRepository

<Interface>
ThreadRepository

- countThreadByUser_Id(Long id): Long
- findThreadById(Long id): Topic
- findThreadByCategoryOrderByCreatedDateDesc(String category): List<Thread>
- findThreadByUser_IdOrderByCreatedDateDesc(Long id): List<Thread>

PostRepository

<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

RegisterController

<Class>
RegisterController

- + userRepository: List<int>
- + passwordEncoder: PasswordEncoder
- displayRegister(): String
- registerUser(String username, String password, String introduction): View

ProfileController

<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

ThreadController

<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

ThreadsController

<Class>
ThreadsController

- + threadRepository: ThreadRepository
- + postRepository: PostRepository
- + displayAllThread(): String
- + displayThreadByCategory(String category): String
- + displayThreadByUser(String id): String
- + setHeader(String category): String

PostsController

<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

ForumApplication

<Class>
ForumApplication

- + main() : static void

Model Details

User, Thread, Post

MVC

AnswersComtroller, IndexController, LoginController, ProfileController, RegisterController, ThreadController, ThreadsController

RegisterView, TopicsView, ProfileView, TopicView, LoginView, indexView

Controller Details

View Details