

Forum Project Requirement

- 📄 Overall Requirements
 - Frontend
 - Backend
- 📄 Database
 - MySQL Tables:
 - MongoDB Documents:
- 📌 Post Stages
- 🚫 User Groups and Permissions
 - Visitor / Banned User:
 - Normal User without email verification:
 - Normal User:
 - Normal User who is the owner of the post:
 - Admin user:
 - Super Admin user:
- 📄 Pages
 - Login Page
 - Register Page
 - User Home Page
 - User Profile Page
 - Contact Admin Page
 - Post Detail Page
 - Home Page (Admin)
 - Message Management Page (Admin)
 - User Management Page (Admin)
- Email Token Validation:
- Navigation Bar
- Security
- AOP
- Mirco Services
 - Requirements
 - Services Example
- Caching
- User Input Checking
- Unit Testing
- Project Collaboration
- Open Questions
- ⚠️ Out of Scope

📄 Overall Requirements

⚠️ All required functions (any requirement without [Bonus]) should be implemented

Frontend

- Use `React` to implement the frontend
- Security protection for endpoints

- Ensure that there is a `Global Navigation Bar` present on every page
- [Bonus] Add `CSS / Bootstrap` styling to polish your UI

Backend

- Use `Spring Boot`, `Spring AOP`, `Hibernate`, `Criteria`, and `MongoDB` to develop the backend
 - Use `MongoRepository` for Post document and use `Hibernate/Criteria` for all `MySQL` tables.
 - You **CAN NOT** use `JPA Repository` for `MySQL` related tables.
- Use `MongoDB` and `MySQL` for the databases
 - Use `AWS RDS` to host `MySQL` database(s), `Mongo Atlas` to host `MongoDB`
- Follow `MircoServices` architecture
 - Use `RabbitMQ` for async communication (send emails), and use `Feign Client` for sync communication
- `Spring MVC` and `RESTful API` architecture are **REQUIRED**. You **CAN NOT** use `Spring Thymeleaf`
- Implement endpoint protection with `Spring Security + JWT`
- Implement `Spring Caching` and `Spring Validation`
- Use `JUnit` and `Mockito` for unit testing.
- Use `AWS S3 Bucket` for file storage
 - Used for user profile image and post attachments
- Use `Spring Email` for sending emails

Database

 You need to have four databases and each database can only hold one table

MySQL Tables:

1. User (userId, firstName, lastName, email, password, active, dateJoined, type, profileImageUrl)
2. History (historyId, userId, postId, viewDate)
3. Message (messageId, userId, email, message, dateCreated, status)

MongoDB Documents:

4. Post (postId, userId, title, content, isArchived, status, dateCreated, dateModified, images, attachments)
 - Contains list of PostReply (replyId, userId, postId, comment, isActive, dateCreated)
 - Each reply also contains a List of SubReply
 - Note that SubReplies can only be added to a PostReply

```

1 {
2   postId: String,
3   userId: Long,
4   title: String,
5   content: String,
6   isArchived: Boolean,
7   status: String / Enum,
8   dateCreated: Date,
9   dateModified: Date,
10  images: Array[],
11  attachments: Array[],
12  postReplies: Array[{
13    userId: Long,
```

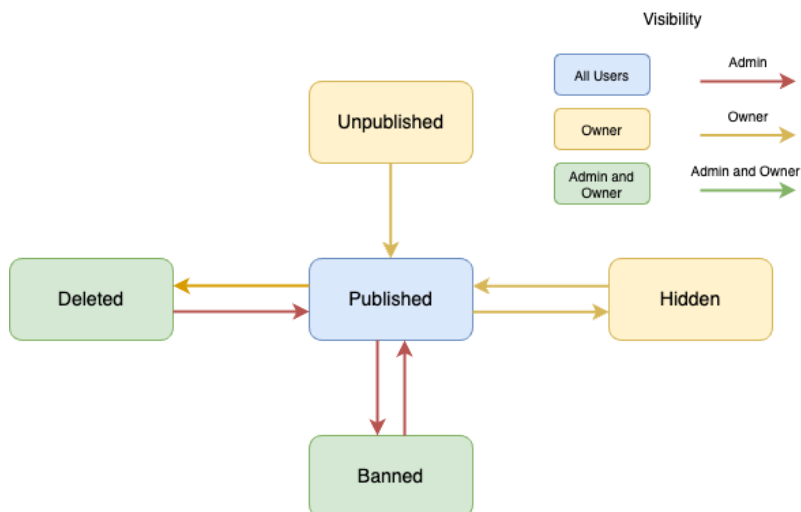
```

14     comment: String,
15     isActive: Boolean,
16     dateCreated: Date,
17     subReplies: Array[{
18         userId: Long,
19         comment: String,
20         isActive: Boolean,
21         dateCreated: Date,
22     }]
23 }
24 }

```

📌 Post Stages

1. **Unpublished:** The user has started a draft but has not published this post yet
 - Only the owner of the draft can view the unpublished post.
 - The admin cannot view posts at this stage
2. **Published:** The user has published this post, which is visible to everyone
 - The user can choose to turn off the ability to reply to this post by archiving the post
 - The user can choose to hide this post (Published → Hidden)
3. **Hidden:** The post is not visible to the public
 - Only the owner of the post will be able to see and modify the hidden post
 - The admin cannot view posts at this stage
4. **Banned:** The admin has taken down this post and thus it will only be visible to the Admins and the owner of the post
 - The post owner cannot modify the content or reply to the post. However, they may choose to delete the post
5. **Deleted:** The post has been deleted by the owner of the post
 - Only post owner able to delete the post. The post owner is still able to see it, but cannot modify the deleted post
 - Admins can recover the post



👤 User Groups and Permissions

Visitor / Banned User:

- Only able to see the login, register and contact us page.

Normal User without email verification:

- Only able to view the Published posts, but not able to create or reply to any post.

Normal User:

- View a **Published** post by clicking on one of the posts in the list
 - That will bring the user to a page that shows the specific post with all of the replies listed.
- Create / Reply to a **Published** post
 - They can delete replies made by them
 - [Bonus] Admin and the owner of the post have the ability to delete other user's replies

Normal User who is the owner of the post:

 All of these additional features should be buttons that are only visible to the owner of the post

- Modify the content of the original post (*Title, Description*)
 - If the title or the description of the post was changed after the original post, add a Last edited: date at the end of the description.
- Change the accessibility from **Published** to **Hidden**
- Allow or not allow other users to reply to this post (archive mode)
- Delete the whole post, while retaining the post in the database, and only modify the state of the post.
 - The user will not be able to recover this post
- [Bonus] Delete other users' replies, while retaining the reply in the database, and only modifying the state of the reply.
 - Note that only the admin or the owner of the post can do this and the message cannot be recovered

Admin user:


 All of these additional features should be buttons that are only visible to the admin / super admin

- Change the accessibility of the post:
 - Ban the post (**Published** -> **Banned**)
 - Unban the post (**Banned** -> **Published**)
- Admin will still have access to deleted posts and also recover them
- Admin will be able to see user management page to ban / unban a user
- Admin are unable to modify posts created by users
- [Bonus] Delete other people's replies

Super Admin user:

- There can be only one super administrator, and their credentials are hardcoded in the database.
- The super admin has the authority to promote a user to the admin role through the user management page.

Pages

 Admin has the access to all user pages and able to create a post

- Login Page `/users/login`
- Register Page `/users/register`

- User Home Page `/home`
- User Profile Page `/users/{id}/profile`
- Contact Admin Page `/contactus`
- Post Detail Page `/posts/{id}`
- Home Page (Admin) `/home`
- Message Management Page (Admin) `/messages`
- User Management Page (Admin) `/users`

Login Page

- This is the default page when the application starts
 - Before being able to view or make a post, a user has first to register and log in. Users can directly click on the register link (provided on either the **Login Page** or the nav bar) and be directed to a Register Page to create an account
- Users should be able to log in with their email and password
- Users will not be able to access any pages other than the **Register Page**, **Login Page**, and **Contact Admin Page** unless they are logged in

Register Page

- A register verification code will be emailed to the user, without verify the email address, the user are not able to create or reply to any post
- The registration process requires users to provide personal information such as their *first name*, *last name*, *email*, and *password*
 - You can set a default user profile image
- Emails are unique to each user, therefore make sure to prevent two accounts from having the same email

User Home Page

- Show the list of **Published** posts, each element would look something like this:
 - By default, the list is sorted by the most recent post
 - Only show user name, date, post title
 - **[Bonus]** Add other filters
 - Provide a button that will sort all the posts in Descending order/Ascending order depending on the replies of each post
 - Provide a button that will sort all the posts in Descending order/Ascending order depending on when the post was created
 - Allow the users to find all the posts related to a specific creator.
 - Provide a reset button to reset the filters
- A button to create a new post, which will open a new page or a pop window for users to create a new post
 - Users can publish the post or save it as a draft for later publishing or editing.

User Profile Page

This page will contain the following functionalities:

- A section to include the user's:
 - Profile image (using S3 bucket)
 - First name
 - Last name
 - Registration date
 - Top 3 posts by the user that are sorted by the number of replies in descending order

- List of all drafts (unpublished posts) that belong to the current user
- A section to show the current user's view history of posts
 - Only display **Published** posts
 - By default, the most recently viewed post should be on top
 - [Bonus] Add the functionality to
 - Find all viewed posts that contain a certain word.
 - Find all viewed posts for a specific date.
- A button for users to edit their profile, which will open a new page or popup window
 - Users would be able to update their profile image
 - If users need to update their email, they need the verification code to complete the update and the user group will become unverified user.


Contact Admin Page

- This will be the page that users will use to contact the admins.
 - example: Admin banned a post due to some reason, the user wants to ask for forgiveness.
- Each request will contain the following information:
 - Subject
 - Email address
 - Message

Post Detail Page

- The post detail page presents the following information:
 - Must have: Post Title, Post Description, User Name and User Profile Image, Post Date
 - Update Date (if the user has made any changes to the post)
 - Attachments (if any)
- Additionally, the page includes the following sections for replies:
 - Replies List: Displays the names and profile images of users who have replied, along with their corresponding messages.
 - Sub-replies List (nested within a reply): Lists the names, profile images, and messages of users who have responded to a specific reply.
- Replying to a post:
 - Users can reply to a published post (posts that is either by them or by someone else)
 - Only one layer of nested reply

Home Page (Admin)

 The admin page and user home page must share the same URL

- Different from the **User Home Page**, the **Admin Home Page** will contain 3 separate functionalities:
 - a. The same feature as the **User Home Page** (filter buttons and list of **Published** posts) but admins will have an additional column at the end to allow them to ban and unban posts
 - b. The same requirements for the **User Home Page** but instead show a list of all **Banned** posts and the button at the end of the row should be "unban" instead of "ban"
 - c. The same requirements for the **User Home Page** but instead show a list of only **Deleted** posts
 - So each row should not contain the "unban" button but instead, have a "recover" button that will make the post available again

Message Management Page (Admin)

- This page will show a table of all messages sent by users
- It will include the following information:
 - Date created
 - Subject
 - Email address
 - Message
 - Status button
 - [Bonus] To allow admins to mark the status of the request as Open/Close
 - The buttons shown must match the status of the request

User Management Page (Admin)

- This page will show a table of all Users including admins
- It will include the following information
 - userId
 - Full name
 - Email
 - Date Joined
 - Type
 - Status button
 - To allow admins to mark the status of the User as Active/Banned
 - Admins cannot ban other admins
 - The buttons shown must match the status of the request
 - Banned users cannot log in

Email Token Validation:

There are two approaches to implementing email token validation:

1. Code Entry Page: Provide a dedicated page where users can enter the 6-digit code that was sent to their email. Upon successful validation, the user will be granted normal user privileges.
 2. Tokenized URL: Instead of a code entry page, send a URL containing a token to the user's email. When the user opens the URL, the validation process will be automatically completed. Once validated, the user will be able to access the application as a normal user.
- [Bonus] Token Expiry: To enhance security, tokens should have a limited validity period. Consider making the tokens valid for 3 hours only, after which they will expire and no longer be usable.
 - [Bonus] Token Reusability: For multiple requests within the token's validity period, consider returning the same token to the user. This approach ensures consistency and avoids generating multiple tokens for the same user during the validation period.

Navigation Bar

The navigation bar should be displayed on every page of the application. Its content should dynamically adjust to show only the necessary buttons based on the current page and the user's group.

Security

Protect your application endpoints using Spring Security + JWT:

1. **Authentication:** Visitors cannot access any endpoints other than for login, registration or contact us usage.
2. **Authorization:** A user with the corresponding authorities can access certain pages that they are granted access to.

AOP

- Utilize `ControllerAdvice` in your code to centrally handle exceptions. Implement this approach for at least three different exceptions.
- Create customized exceptions as needed to address specific scenarios or requirements in your application. Custom exceptions can be defined to encapsulate specific error conditions or provide more meaningful error messages.

Mirco Services

Requirements

- Each microservice must have its own database.
- A microservice can only access its own database and cannot define entities or foreign key constraints that reference tables belonging to other microservices. If information from another service needs to be referenced, only the IDs of associated objects can be stored.
- Spring Security must be implemented on each microservice to secure the endpoints.
 - The Email Service should not expose any endpoints and should only consume messages from RabbitMQ.
 - The File Service can be designed in different ways:
 - Front-end connectivity: The front-end can directly connect to the File Service to upload files and submit the URI in the form.
 - Composite service handling: The file can be submitted along with the form to the composite service. The composite service will then send the file to the File Service, retrieve the URI, and save the record.

Services Example

1. Gateway Service:
 - Responsible for routing and handling incoming requests.
 - Does not connect directly to any specific database.
2. Authentication Service:
 - Handles user authentication and authorization.
 - Can only make API calls to the User Service.
3. User Service:
 - Manages user-related operations such as user profile, registration, etc.
 - Connects to the user database for user-specific data.
4. History Service:
 - Manages post history functionality.
 - Connects to the history database.
5. Post and Reply Service:
 - Manages posts and replies functionality.
 - Connects to the post database (MongoDB) for storing post-related data.
6. Message Service:
 - Handles contact us functionality.
 - Connects to the message database for storing messages.
7. File Service:
 - Manages file-related operations such as uploading, etc.

- Connects to an S3 bucket for storing files.

8. Email Service:

- Consumes messages from RabbitMQ for sending emails.
- Does not connect directly to any specific database.

9. Other Composite Service:

- Cannot connect to any databases directly.
- Can make API calls to interact with other services for data retrieval or operations.

Caching

Find at least one suitable place to implement Spring caching in a method that meets the following criteria:

1. Long Execution Time: The method requires a significant amount of time to execute.
2. Infrequent Value Changes: The result of the method does not change frequently.
3. High Request Volume: The method is expected to receive a large number of requests.

To implement Spring caching, remember to use different annotations such as `@Cacheable`, `@CachePut`, and `@CacheEvict` to define caching behavior.

User Input Checking

It is essential to validate all user input and not blindly trust it. User input should be checked not only in the request body but also in path variables and request parameters.

To implement effective input validation, consider the following ways:

1. Spring Validation: Utilize Spring's validation to validate the request body. This helps ensure that the input adheres to the specified rules and constraints.
2. Path Variables and Request Parameters: Validate and sanitize any user input received through path variables and request parameters. Apply appropriate checks and throw exceptions as needed.
3. JWT Token as User Identification: Instead of relying on the user ID in the path variable, trust the information provided in the JWT token to determine the current user. Verify the validity and integrity of the token, and extract relevant user details from it for authentication and authorization purposes.

Unit Testing

- Use `JUnit` and `Mockito` for writing unit tests.
- Employ JaCoCo to achieve a minimum of **95%** code coverage for the following components:
 - `UserController`
 - `UserService`
 - `UserDAO`

Project Collaboration

- Utilize GitHub as the centralized repository for the project.
 - Create separate repositories for each service/component of the project.
- Adopt Jira as the team's ticket management tool to track tasks and issues.
- Each team member is responsible for writing both backend and frontend code.
- Share a Postman collection with the team to facilitate API testing and documentation.
- Share a final Entity-Relationship (ER) diagram with the team to illustrate the database structure and relationships.

Open Questions

Questions	Answers

 Out of Scope