FINAL REPORT

SWE573 Software Development Practice, 2023/2024

Prepared by: Yana Krasovska, student number 2022719111

Submission date: 20 May 2024

Project name: GetToGether: Community Website

Deployment URL: https://comweb.kekocat.site/

Git repository URL: https://github.com/kotobusdriver/swe573-YanaKrasovska/tree/main

Git tag version: v0.9

HONOR CODE
Related to the submission of all the project deliverables for the Swe573 2024 Spring semester project reported in this report, I, Yana Krasovska, declare that:
- I am a student in the Software Engineering MS program at Bogazici University and am registered for Swe573 course during the Spring 2024 semester.
- All the material that I am submitting related to my project (including but not limited to the project repository, the final project report, and supplementary documents) have been exclusively prepared by myself.
- I have prepared this material individually without the assistance of anyone else with the exception of permitted peer assistance which I have explicitly disclosed in this report.
Yana Krasovska

Contents

Overview

The web application GetToGether: Community Website was created with a purpose to help people with similar interests find each other and communicate in a uniquely structured way, which is different from what the social networks and most notice boards offer today. In this application, a community is defined as a user-created, thematic space centered around a specific topic or interest. Communities are structured around posts that follow a predefined set of rules and guidelines.

The objective of this student project was to develop the application in line with the requirements and following good development practices.

Software requirements specification and status of requirements

Purpose of the Web-Based Community Platform

The system of the web-based community platform shall provide a user with a wide array of pre-defined tools and options for selection to construct a community centered around a user-defined topic.

Platform Requirement

The system shall be a web-based application, accessible and operable exclusively through standard web browsers without the need for any specialized client-side software installation.

I. User Interface in the system (completed partially)

1. The system shall present information to users in a feed-like format, enabling dynamic content streaming based on user preferences and interactions. (completed partially as a feed of posts in individual community pages).
2. Feed content will be categorizable and filterable through the use of content tags and page view filters. (not completed)

II. Front and Home Page of the system (not completed)

1. For an unregistered/unlogged user, the front page shall present a selection of posts from the most active communities; this selection may include the most recent posts, most popular posts, and posts from communities geographically close to the location of this unregistered/unlogged user.
2. For registered/logged-in users, the home page shall present the updates (recent activities) from the communities this user is subscribed to, as well as posts of the users he/she follows.

3. The system shall offer personalized content recommendations to users based on their activity history and preferences to enhance user engagement.

III. Registration in the system (completed partially)

1. For a user to register, the system shall require, as a minimum, an email, a name (or nickname), a password, and a description in the user profile. (completed)
2. The system shall implement CAPTCHA verification during registration to prevent automated bot registrations. (not completed)
3. After the user has registered, the system creates his/her minimal user profile with the information the user provided during the registration. (not completed)

IV. Authentication (logging-in) in the system (completed partially)

1. The system shall authenticate users attempting to log in by the email and password they provided during registration. (completed)
2. Users shall have the ability to recover their accounts through a secure account recovery process if they forget their login credentials. (not completed)

V. Types of Users in the system (completed partially)

1. Immediately upon registering with the system and before joining any community, all users shall be of type "user".(completed)
2. Upon joining a community, a user shall be assigned the type "community member" by the system. (completed)
3. The system shall assign the role of "community administrator" to the user immediately upon him/her creating a community. (completed)
4. The community administrator may assign the role of "community moderator" to at least one community member or himself/herself. (not completed)

VI. User Communication in the System (not completed)

1. Users across the system shall be able to message one another by sending direct messages via asynchronous communication means, without the expectation of immediate response.
2. A user may follow any other user registered in the system, thus becoming a follower.
3. Where a user follows other users, the followed users' posts shall be displayed on the follower's home page.
4. Users shall be able to easily report other users' behaviour to the community moderator by sending the community moderator a direct message.

VII. Community Administration (not completed)

1. Any community may have up to five community administrators and an unlimited number of community moderators.
2. The community administrator may leave the community he/she created. Before doing so, he/she shall designate at least one member of the community as a community administrator.
3. The community administrator shall not have the right to unsubscribe community members.

VII.A. Creating a Community (completed)

1. Any user registered and logged in the system may create an unlimited number of communities. (completed)
2. When a user creates a community, the system shall assign the role "community administrator" to that user. (completed)

3. The system shall offer a template community profile to be filled in by the user in order to create a community; this will ensure consistency in appearance and structure across the communities on the platform. (completed)
4. When a user is creating a community, the system shall check that the community name is unique. (completed)
5. When a user is creating a community, the minimum information to complete the creation process shall be entering the community name, community description and setting the community visibility in the relevant fields of the community profile. (completed)
6. A community shall have two visibility modes: public and private. (completed)
7. As part of the community creation process, the community administrator shall be able to create and save a template for all posts in that community. (completed)

VII.B. Editing a Community (not completed)

1. After a community is created, the community administrator shall be able to edit the text in its profile fields; the community administrator shall not be able to edit the name of the community.

VII.C. Community visibility (completed partially)

1. When creating a community, the user may make the community either public or private. (completed)
2. A public community shall be open for subscription to all registered and logged system users. (completed)
3. All posts in the public community shall be public and visible to all system users irrespective of the registration or logged-in status. (completed)
4. If the community is private, only its name and description shall be public and visible to all system users irrespective of registration or login status. (not completed)
5. In a private community, all user posts shall be private and visible only to the community members. (not completed)
6. In a private community, all comments and reactions to posts shall also be private and visible only to the community members.(not completed)

VII.D. Deleting a Community(not completed)

1. The system shall not provide any mechanism to delete a community once it is created.

VII.E. Archiving a Community (not completed)

1. The system shall provide a process for community administrators to archive the community.
2. This process shall include confirmation steps to prevent accidental archiving.
3. When archived, the community shall become read-only for all users, including community members and administrators.
4. The system shall ensure that no new posts, comments, or reactions can be added to the archived community
5. Archived public community content shall remain viewable and searchable for the system users.
6. Archived private community content shall remain viewable and searchable for its community members only.

VII.F. Moderation within Communities (not completed)

1. Community rules and guidelines shall be prominently displayed for new members upon joining and accessible at all times to all community members.
2. A community administrator may assign the role of community moderator to any community member, as well as to himself/herself.
3. A community moderator may delete posts and comments that contain offensive content.
4. Types of offensive content shall be described in the community rules.
5. A moderator shall be able to suspend community members by temporarily blocking their ability to post new content, react to new content, or reply to new content.

6. A moderator shall hold an exclusive right to unsubscribe a community member.

VIII. Communication in Communities

VIII.A. Subscribing to Communities (completed partially)

1. A registered and logged-in user of the system may subscribe to and become a member of any number of communities. (completed)
2. A community member may unsubscribe and stop being a member of any and all communities he had been a member of before. (not completed)
3. Subscription to private communities for new members shall be by invitation only. (not completed)
4. A member of a private community shall be able to send an invitation to another user registered in the system to subscribe to that private community. (not completed)

VIII.B. Contributing to Communities

1. User contribution posts in communities shall be enforced to conform to the post template(s) for this community set by the community administrator. (completed)
2. Before posting a post to a community, the system shall verify that the post has a title, its body meets the minimum required character length, and that the text in the body does not exceed the maximum length. (not completed)
3. A community member may edit and delete their posts at any time; upon doing so, the system shall tag the post as "edited", or shall replace the post with a "deleted" tag, as appropriate. (completed partially)

VIII.C. Interaction within Communities(not completed)

1. Community members shall be able to leave comments on posts in the communities.
2. The system shall support multimedia posts, allowing users to include images, videos, and links within their posts.
3. Community members shall be able to leave reactions on posts in the communities.
4. These reactions shall be represented by a predefined set of emoticons, enabling users to convey a range of emotions towards the post.
5. The post writer shall be notified about comments and reactions to his/her post.
6. Community members shall be able to flag posts and comments that contain offensive content.
7. The system shall notify the community moderator(s) about flagged posts and comments by sending them an automated direct message with the following structure: "[Community member username] flagged [link to flagged content]"

IX. Content Management within Communities

IX.A. Tagging Content(not completed)

1. A user may choose to append a tag to his/her post and shall be free to define a tag.
2. When a user creates a post, the system shall offer him/her a list of tags previously defined by this user, as well as a list of top 10 popular tags within the community.

IX.B. Searching and Filtering Content

Basic Search (completed partially)

1. The system shall allow text search across posts, comments, community, and user profiles in all public communities, including archived ones. (not completed)
2. Basic Search shall provide an option to include or exclude content from archived communities in the search results. (completed for communities only)
3. Users shall have the ability to sort search results by relevance, date (newest or oldest first), and popularity (e.g., number of reactions or comments). (not completed)

Advanced Search (not completed)

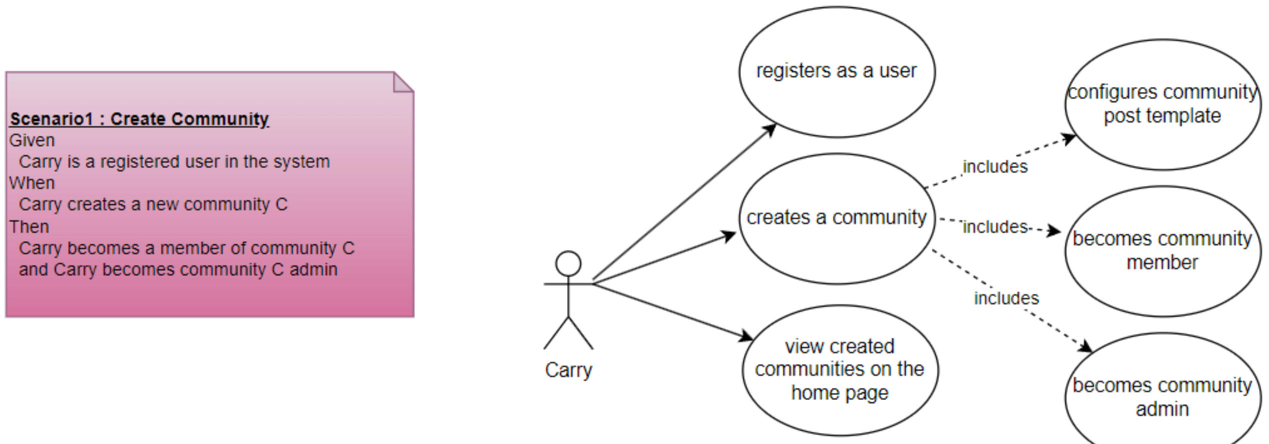
1. Sorting as described in Basic Search 2. shall apply to advanced search.
2. The system shall allow users to perform advanced searches using a combination of text, tags, author names, and time frames(e.g., created or updated within a time frame).
3. Advanced Search shall offer filters to limit searches by content type (e.g., posts, comments, user profiles, and community descriptions).
4. The system shall enable users to filter search results by a specific community by typing in the community name in the specially designated field on the advanced search pane.
5. In the Advanced Search interface users shall be able to use search operators (e.g., AND, OR, NOT).

X. Types of Data in the System (types of data: text, date, and file)

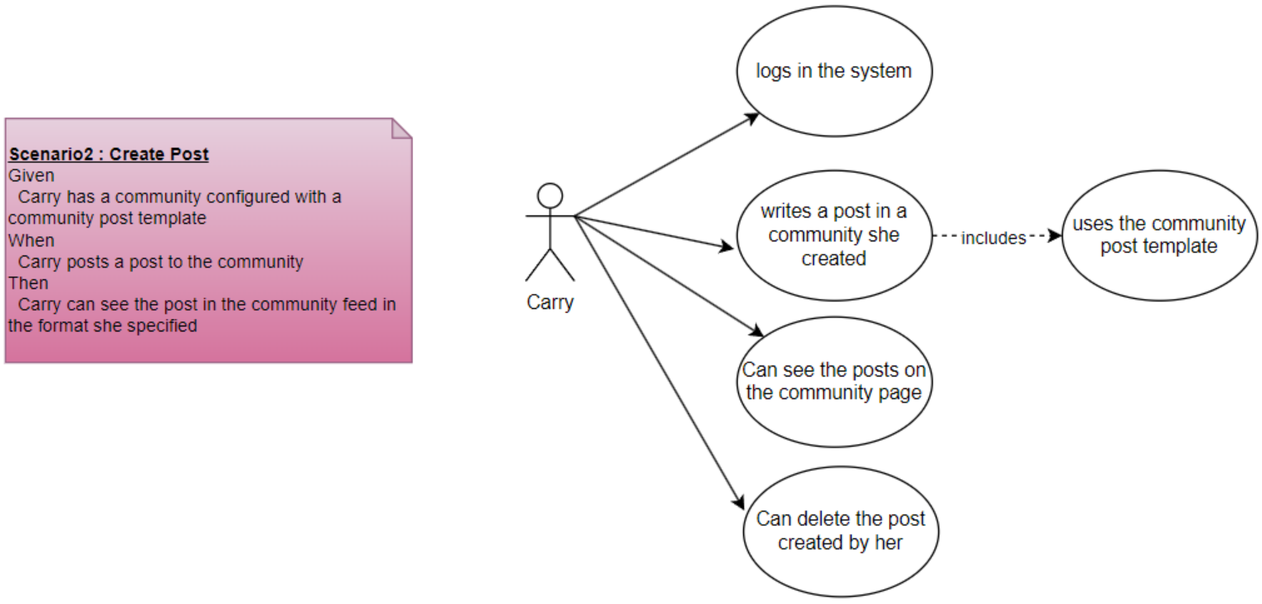
1. Textual Content: posts, comments, community descriptions, user profiles, messages, and tags. Text fields have minimum and maximum length requirements to ensure clarity and conciseness.
2. Multimedia Content: images, videos, and links within posts, allowing communities to share and discuss visual and multimedia resources effectively.
3. User Data: registration information, including email, name or nickname, password, and a brief description in the user's profile. Also includes user types (user, community member, community administrator, community moderator) and their interactions within the system (follows, messages, etc.).
4. Community Data: community names, descriptions, visibility settings (public or private), and post templates designed by community administrators to standardize content creation within the community.

Design documentation

Use case diagrams corresponding to usecase scenarios

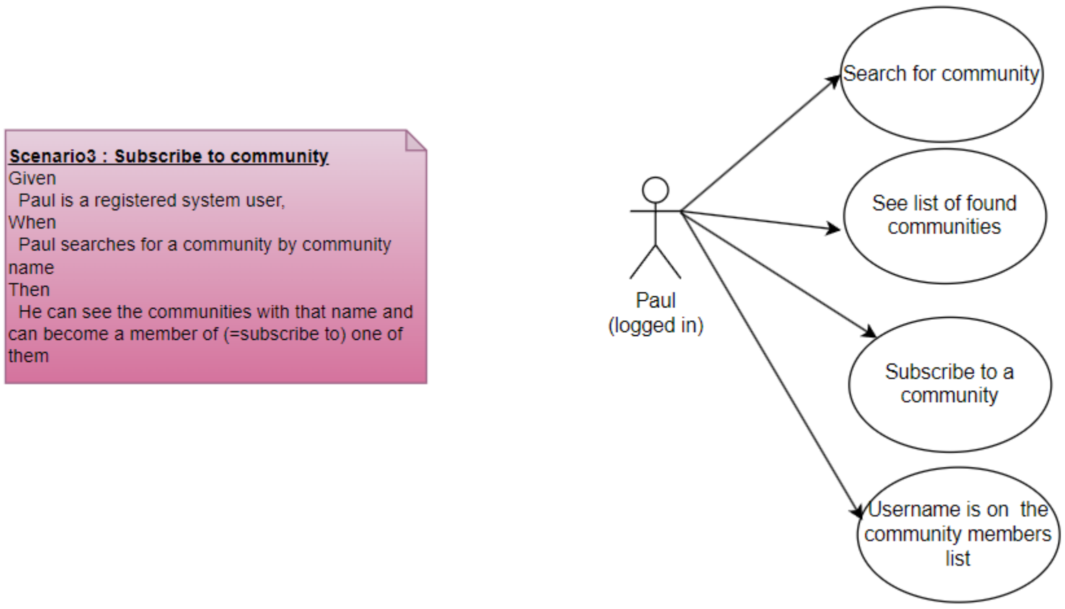


Scenario1 : Create Community
Given
Carry is a registered user in the system
When
Carry creates a new community C
Then
Carry becomes a member of community C
and Carry becomes community C admin



Scenario2 : Create Post
Given
Carry has a community configured with a community post template
When
Carry posts a post to the community
Then
Carry can see the post in the community feed in the format she specified

Carry
logs in the system
writes a post in a community she created
includes
uses the community post template
Can see the posts on the community page
Can delete the post created by her



Scenario3 : Subscribe to community
Given
Paul is a registered system user,
When
Paul searches for a community by community name
Then
He can see the communities with that name and can become a member of (=subscribe to) one of them

Paul (logged in)
Search for community
See list of found communities
Subscribe to a community
Username is on the community members list

System architecture

The system architecture contains three main parts: web user interface, backend, and database. The web UI is a React-based single-page application (SPA) served with Node.js. The backend is a Spring Boot application developed with Java, and the database is a Postgres database.

Implementation details

Backend is using a layered architecture starting from REST controllers that accept requests from the UI, going through workflows to execute use cases. Workflows delegate database-related operations to the repositories layer that enables database persistence. Reusable functionalities are handled by Service classes. A high-level representation of this information flow and a sequence diagram modelled on Scenario 1 is provided below:

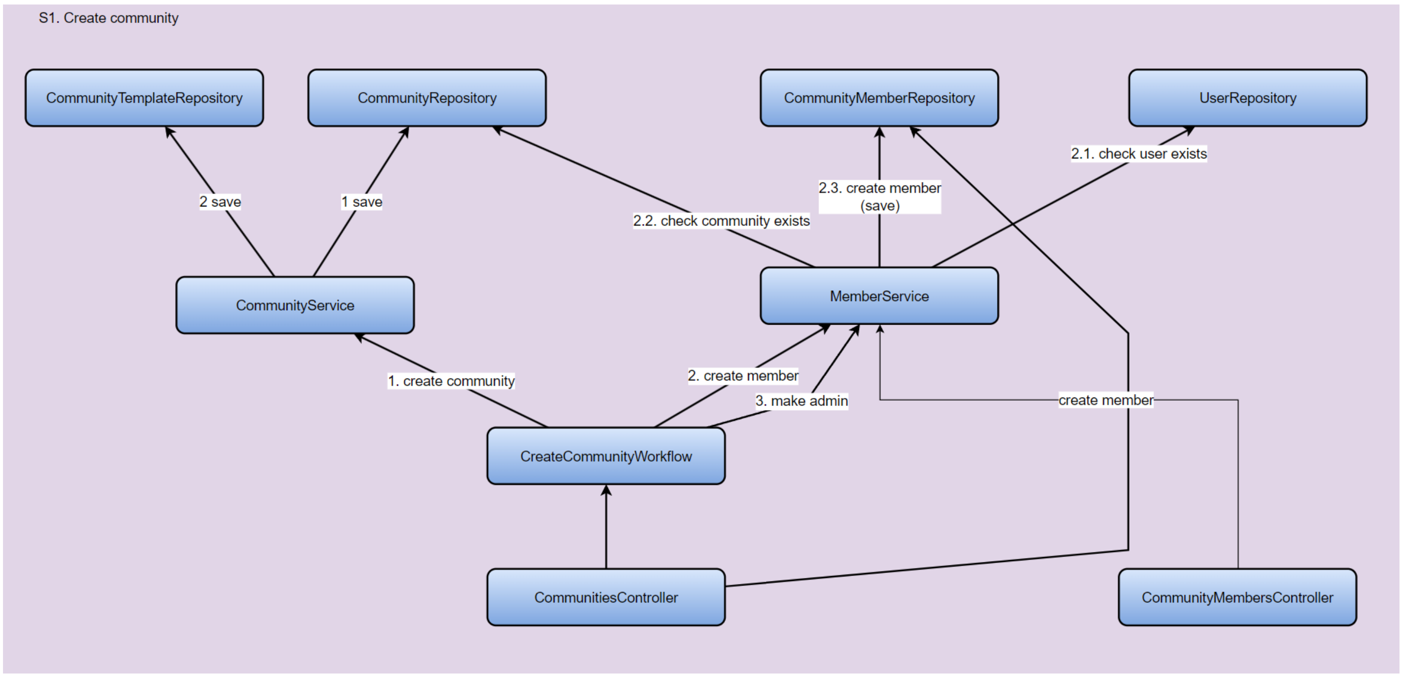


Figure 1. High-level representation of class calls for Scenario 1

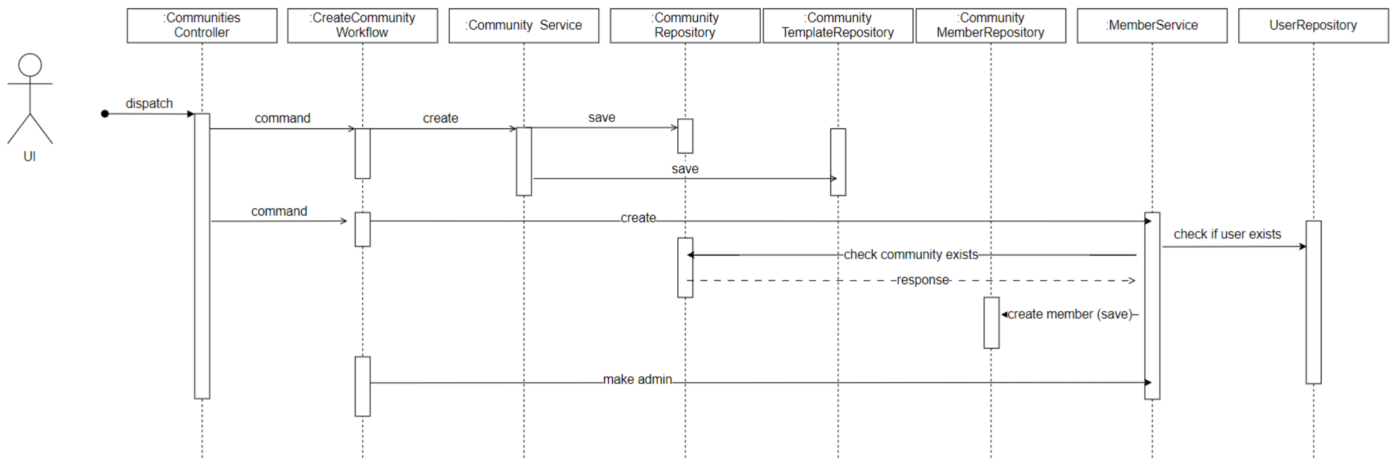


Figure 2. Sequence diagram for Scenario 1

Flyway library is used for DB schema creation, persistence, and migration, overriding the automatic creation of the DB schema by Hibernate. Flyway accepts a developer-generated DDL scripts to create schema tables.

Frontend React.js library is used for a components-based single-page application that runs in the browser. The UI is using HTML, JavaScript, and CSS as main technologies. The web-application is served by Node.js.

Database schema

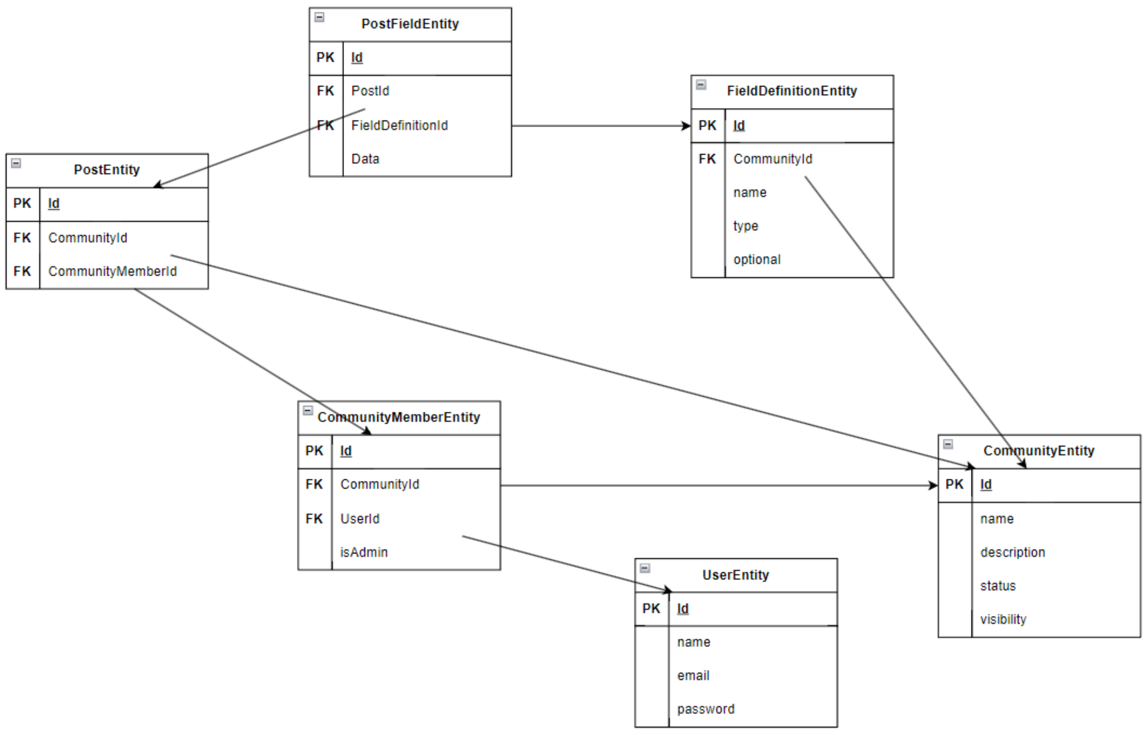


Figure 3. Database schema of the application

Community post template schema

Being a key element of the requirements, the community post template was implemented according to the dataflow diagram below. The post template is customizable and is created once during the creation of the community. The post template cannot be modified once created.

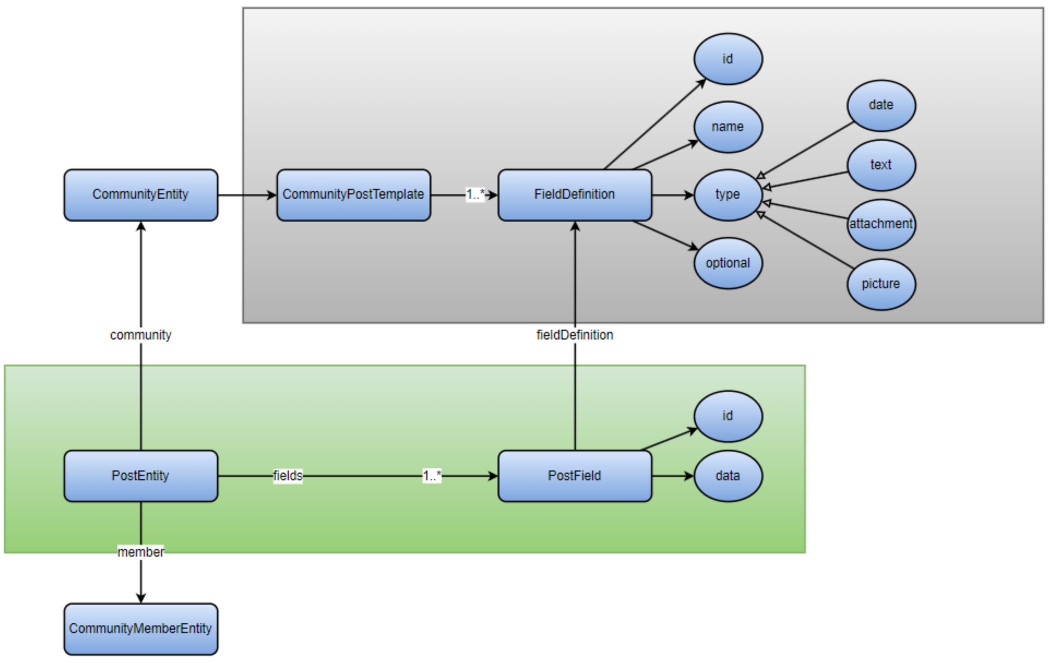


Figure 4. Community post template data flow diagram

At the time of creation of a community user provides a list of field definitions that will be used to define fields in the community post template. Each field definition contains information about the name of the field to display to the users, the type of the data that this field contains, and the optionality of the field for the post. Every time a new post is created, the definitions for the associated fields are validated with the provided user input. In the UI part, the dynamic nature of community posts required a dynamic rendering of UI components for fields based on the fields type. This required special logic (implemented by using switch cases) in both post creation components and rendering post components.

Deployment details

The application is running on a VM in the home server with access to it provided via a CloudFlare tunnel and facilitated by Nginx which acts as a gateway (see Figure 5).

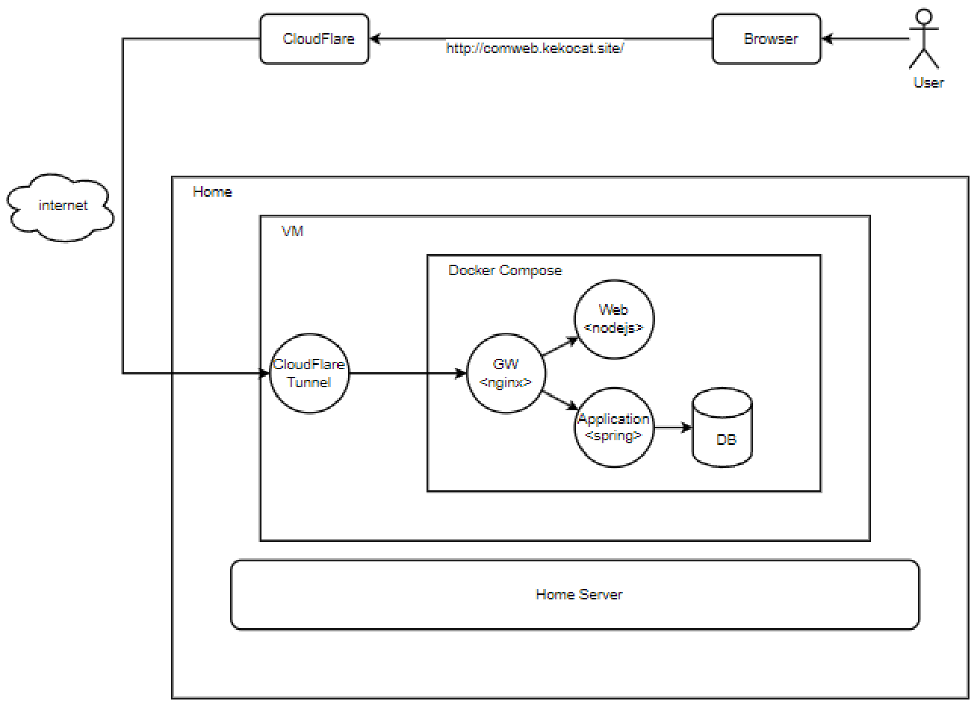


Figure 5. Deployment scheme

Docker instructions

To run the application on your local machine, follow these instructions:
1. Install Docker Desktop
Firstly, if you are on Windows install/update WSL 2 by typing the following command in a terminal

```
C:\Users\Lenovo> wsl --update
```

Then, download the appropriate installation binary from https://docs.docker.com/desktop/
After the download is completed go on to install Docker Desktop. At the end of the installation you will need to restart your system.
Make sure to check that Docker Engine is running by opening the Docker Desktop application.

2. Open a terminal in the Java source folder root, where docker-compose.yaml file resides.

3. Run the following command in the terminal to start the database and the application

```
C:\Users\Lenovo> docker-compose up
```

4. Wait for container images to be pulled from internet and the application to be built and started. This may take some time.

5. Open http://localhost:19000 in your browser to view the application

6. To stop the application use Ctrl+C to stop containers in the terminal.

Testing

Tests implemented

The testing approach followed in this project – only the backend- focuses on the end-to-end scenarios for general flows and unit testing of smaller but complicated logic. The scenarios tested can be found below:

- **Create user scenario:** This scenario invokes the usecase for creation of a user and then validates the usecase by attempting to log in using the credentials used during user creation.
- **Create community scenario:** This scenario extends upon the create user scenario and continues with the community creation usecase. To assert the correctness of the business logic executed as part of the community creation usecase, the existence of the new community is checked and the membership and existence of admin rights of the creating user are validated.
- **Subscribe to community scenario:** This scenario builds on top of the create community scenario, by extending it with creation of another user who later on subscribes to the aforementioned community. To validate the success of the subscription logic, the membership information is checked for the new user.
- **List community members scenario:** This scenario also builds on top of the previous scenario, where we subscribe a new user to a community. To validate, the list of members for the community is checked to contain both the creating user and the new user that subscribed to the community.
- **Send post scenario:** This scenario starts with the scenario where a community is created. Afterwards, it continues with the creating user sending a post for this community.
- **Delete post scenario:** This scenario extends the send post scenario by later on deleting the created post.

In addition to the scenario tests, two unit tests are introduced to further validate individual services that are responsible for creating a community and posts respectively.

- **CommunityServiceTest:** This is the unit test class developed to test the correct interactions between classes that are responsible for creating a community. This test ensures that the responsible service interacts with its dependencies in a correct fashion.
- **PostServiceTest:** This is the unit test class developed to test the correct interactions between classes that are responsible for creating a post.

The implementation of the tests depends on three additional testing libraries, namely: Junit, H2 and Mockito. Junit is used for testing and assertions. Mockito is used for mocking dependencies of unit tests. H2 is an in memory relational database, and it is used as the backing database for the scenario tests.

Scenario tests are run as Spring Boot integration tests, and they encompass the entire execution flow starting from the REST controller layer all the way to the persistence and in memory database H2. They provide a thorough and robust mechanism to exercise and test important execution flows in the backend.

Test results report

Built by: maven

Surefire Report

Summary

[Summary] [Package List] [Test Cases]

Tests	Errors	Failures	Skipped	Success Rate	Time
14	0	0	0	100%	13.57 s

Note: failures are anticipated and checked for with assertions while errors are unanticipated.

Package List

[Summary] [Package List] [Test Cases]

Package	Tests	Errors	Failures	Skipped	Success Rate	Time
swe.spec.units	2	0	0	0	100%	0.990 s
swe.spec.scenarios	12	0	0	0	100%	12.58 s

Note: package statistics are not computed recursively, they only sum up all of its testsuites numbers.

swe.spec.units

-	Class	Tests	Errors	Failures	Skipped	Success Rate	Time
✓	CommunityServiceTest	1	0	0	0	100%	0.676 s
✓	PostServiceTest	1	0	0	0	100%	0.314 s

swe.spec.scenarios

-	Class	Tests	Errors	Failures	Skipped	Success Rate	Time
✓	CreateCommunityTest	2	0	0	0	100%	12.14 s
✓	CreateUserTest	2	0	0	0	100%	0.031 s
✓	DeletePostTest	2	0	0	0	100%	0.165 s
✓	ListCommunityMembersTest	2	0	0	0	100%	0.092 s
✓	SendPostToCommunityTest	2	0	0	0	100%	0.077 s
✓	SubscribeToCommunityTest	2	0	0	0	100%	0.070 s

Test Cases

[Summary] [Package List] [Test Cases]

CreateCommunityTest

✓	contextLoads	0.782 s
✓	create_community	0.688 s

CreateUserTest

✓	contextLoads	0.006 s
✓	create_user	0.010 s

DeletePostTest

✓	contextLoads	0.007 s
✓	delete_post	0.146 s

ListCommunityMembersTest

✓	contextLoads	0.006 s
✓	list_community_members	0.075 s

SendPostToCommunityTest

✓	contextLoads	0.005 s
✓	send_post	0.057 s

SubscribeToCommunityTest

✓	contextLoads	0.005 s
✓	subscribe_to_community	0.051 s

CommunityServiceTest

✓	should_create_community	0.171 s

PostServiceTest

✓	should_create_post	0.030 s

Data to test the system

Username: Tester, email: tester@test.me password: tester

Manual test scenarios file

Link to manual user test scenarios: https://github.com/kotobusdriver/swe573-YanaKrasovska/wiki/Application:-user-test-scenarios

User manual

The application works with all major browsers (Chrome, Explorer, Firefox). Currently, an unlogged user can view communities and posts in them, and search for communities using basic search.

Search for a community

Logged and unlogged users can utilize Search box at the top to search for communities by a keyword in the community name.

Register in the system

To start using the application, a user must first register and then log in, as most functions are available to registered users only. After providing name, username, and password the system checks that they are unique.

Logging in

To log in, the user must enter the email and password they used to register. If the combination is wrong, the system will display "Invalid login" message.

Create a community

After logging in, the user can create a community. To do so, the user clicks Create new community on the home page. After entering community name, description, and choosing visibility, it is possible to create a custom post template which will be used by all members of this community. Currently it is not possible to modify or delete the post template once it is created. To set post field types, the user should use radio buttons; it is also possible to set whether a field is a mandatory one to be filled or an optional one. Automatically, the user who creates the community becomes its member and admin.

Once a community with its respective template is created, the user is redirected to the home page from where the user can see the card and the link to the newly created community.

Join a community

To be able to write to a community, the user must first subscribe to it by clicking the Subscribe button. Immediately after that a post form will become available based on the template set for this community. The user can also see their username appear in the list of this community members, next to the community name.

Write a post to a community

After subscribing to the community, the user can use the post form to contribute to the community. The user can delete the posts that they wrote to the community.

Link to video of the system

Link to video of the system: https://drive.google.com/file/d/1hxakrEdNpWtnSzzEUMibRoIjfudg2NmN/view