Telerik Academy

Social Network

Team Project Assignment

Java Team Project

This document describes a **team project assignment** for the **Java cohort** at Telerik Academy.

Project Description

Your task is to develop a **SOCIAL NETWORK** web application. The **SOCIAL NETWORK** application enables you to:

- Connect with people
- Create, comment and like posts
- Get a feed of the newest/most relevant posts of your connections.

The exact theme of the application is intentionally not specified and is limited only to your imagination and abilities. It could be inspired by Facebook and have its focus on the feed generation or be targeted to travelers and have its posts in the format of reviews or be a place to keep in touch with you fellow students.

Project Requirements

Note: The document contains mocks that are used for visualizing the described functionality. Even though the team might choose to implement a similar design, the mocks should not be interpreted as design requirements and constraints.

UI

Create a “don’t-make-me-think”, simple, intuitive UI, so you have enough time to implement the backend carefully and adequately.

- Use **Spring MVC Framework with Thymeleaf** template engine for generating the UI.
- You may change the standard theme and modify it to apply own web design and visual styles. For example you could search and use some free html & css template to make your web application look good.
- You may use Bootstrap or Materialize

Web application

Public Part

The public part of your projects should be **visible without authentication**. This includes:

- Application home page – a landing page with links to the login and registration forms, profile search and public feed
- Login form
  - (optional) Forgotten password
- Registration form
  - (optional) Email confirmation
- Public profiles – shows the profile information, that the user decided to be public (name, profile picture etc.) and his/hers public posts ordered chronologically
- Search profiles – search for profiles based on name and/or email
- Public feed – chronologically ordered public posts. More details about the posts follow in the next sections.

Private Part (Users only)

Registered users should have private part in the web application accessible after **successful login**.

Registered users have a profile administration page where they can edit their personal and login information. This includes:

- Change name
- Upload a profile picture
- Set visibility of picture (public – visible for everyone including non-registered users, connections only – visible only for connected profiles)
- (optional) Change password
- (optional) Change email
- (optional) Any personal information that may be relevant to your social network platform (age, nationality, job etc.)

Registered users have interactions with other users. While viewing another user's profile, the user can request to connect or to disconnect (depending on the current status). Connection requests need to be approved by the other user before they become active, but disconnections don't. The roll of connections will become clear in the next sections.

Registered users can create posts. Similarly to the picture visibility settings, when creating the post, the user can choose for it to be either public (visible for everyone, including non-registered users) or connections only (visible only for his/hers connections). The content of the post depends on the type of social network you are creating. The base requirement for a post is that it contains text, however, as an optional task you can add more functionality to the post. Some suggestions are:

- (optional) Upload a picture
- (optional) Upload a song

- (optional) Upload a video
- (optional) Location

Registered users have a personalized post feed, where they can see a feed formed from their connection's posts. The base requirement is that the feed is generated by chronologically ordering all the posts from your connections, however, you can optionally create a more complex algorithm for feed generation. Some suggestions are:

- (optional) You can take into account the number of interactions with the post (comments, likes etc.) and place posts with more interactions higher in the feed
- (optional) You can generate the feed based on user location, and show higher in the feed posts, that are tagged closer to the user
- (optional) Add the algorithm to the public feed section as well

Registered users have interactions with other user's posts. They can:

- Like a post – Registered users can like or unlike (if you have already liked it) a post. The post total like count is showed along the post content.
    - (optional) Include the comment like count in the feed generation algorithm
- Comment a post – Under each post there is a comment section. Registered users can comment on a post. The newest n comments (choose the count to what seems best to you, but there has to be a limit) are shown along the post content. Users can expand the comment section of the post in order to read older comments.
    - (optional) Include comment reply functionality

Administration Part

System administrators should have administrative access to the system and permissions to administer all major information objects in the system. They should be able to:

- Edit/Delete profiles
- Edit/Delete post
- Edit/Delete comments

REST API

In order to work with collaboration with the QA team or provide other developers with your service, you will need a REST API.

The **REST API** should leverage **HTTP** as a transport protocol and clear text **JSON** for the request and response payloads.

API documentation is the information that is required to successfully consume and integrate with an API. Use Swagger to document yours.

Note: Please keep in mind that your web application should be build using **Spring MVC Framework with Thymeleaf template engine**. The REST API is only for external usage of your services.

Database

The data of the application MUST be stored in a relational database - **MySQL**.

You need to identify the core domain objects and model their relationships accordingly.

Technical / Development Requirements

General development guidelines include, but are not limited to:

- Use **IntelliJ IDEA**
- Following **OOP** principles when coding
- Following **KISS, SOLID, DRY** principles when coding
- Following **REST API** design best practices when designing the REST API (see Apendix)
- Following **BDD** when writing tests
- You should implement sensible **Exception handling** and propagation
- Every time you use System.out.println or e.printstacktrace() in a none-joking, serious manner, a kitten **dies**

Backend

- The minimum **JDK version is 1.8**
- Use tiered project structure (separate the application **components in layers**)
- Use **SpringMVC** and **SpringBoot** framework
- For Persistence use **MySQL/MariaDB**
- Use **Hibernate/JPA (and/or Spring Data)** in the Persistence layer
- Use **Spring Security** to handle user registration and user roles
    - Your registered users should have at least one of the two roles: **user** and **administrator**
- Service layer (e.g. “business” functionality) should have **at least 80% test unit coverage**

Frontend

- Use **Spring MVC Framework with Thymeleaf template engine** for generating the UI
- Use **AJAX** form communication in some parts of your application

Deliverables

Provide link to a **Git** repository with the following information:

- Each project source code MUST be available in a dedicated **GitLab/GitHub** repository and **Trello board**
- Commits in the GitLab repository should give a good overview of how the project was developed, which features were created first etc. and the people who contributed.

Contributions from all team members MUST be evident through the git commit history (so don't squash commits)!
- Read https://dev.to/pavlosisaris/git-commits-an-effective-style-guide-2kkn and https://chris.beams.io/posts/git-commit/ for a guide to write good commit messages
- The repository MUST contain the complete application source code and any run scripts
- The project MUST have at least **README.md** documentation describing how to build and run the project
- Screenshots of the major application user facing screens with some data on them
- **URL** of the application (if hosted online)
- Link to the **Trello** board

Optional Requirements (bonus points)

- Integrate your app with a **Continuous Integration server** (e.g. Jenkins or other). Configure your unit tests **to run on each commit** to your master branch
- Host your application's backend in a public hosting provider of your choice (e.g. AWS, Azure)

Public Project Defense

Each student must make a **public defense** of its work to the trainers, Partner and students (~30-40 minutes). It includes:

- Live **demonstration** of the developed web application (please prepare sample data).
- Explanation of the application structure, major architectural components and selected source code pieces demonstrating the implementation of key features

Expectations

You MUST understand the system you have created.

Any defects or incomplete functionality MUST be properly documented and secured.

It's OK if your application has flaws or is missing one or two MUST's. What's not OK is if you don't know what's working and what isn't and if you present an incomplete project as functional.

Some things you need to be able to explain during your project defense:

- What are the most important things you've learned while working on this project?
- What are the worst "hacks" in the project, or where do you think it needs more work?
- What would you do differently if you were implementing the system again?

Appendix

Guidelines for designing good REST API

https://blog.florimondmanca.com/restful-api-design-13-best-practices-to-make-your-users-happy

Guidelines for URL encoding

http://www.talisman.org/~erlkonig/misc/lunatech%5Ewhat-every-webdev-must-know-about-url-encoding/

Always prefer constructor injection

https://www.vojtechruzicka.com/field-dependency-injection-considered-harmful/

Design mock-ups

https://projects.invisionapp.com/share/NAUATD0DBJQ#/screens/387803914