Telerik Academy

Social Network

Build community and stay connected

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 not specified intentionally 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 your fellow students.

Functional Requirements

Entities

- Each **user** must have a username, password, email, and a photo.
    - Username must be unique and between 2 and 20 symbols.
    - Password must be at least 8 symbols and should contain capital letter, digit and special symbol (+, -, *, &, ^, …)
    - Email must be valid email and unique in the system.
- Each **post** must have content (text/photo/etc.), comments and likes.

Public Part

The public part must be accessible without authentication i.e., for anonymous users.

Anonymous users must be able to register and login.

Anonymous users must see a landing page with links to the login and registration forms, profile search and a public feed. Public profiles must show the profile information, that the user decided to be public (name, profile picture etc.) and their public posts ordered chronologically. Search profiles must search for profiles based on name and/or email. Public feed must show chronologically ordered public posts.

Private part

Accessible only if the user is authenticated.

Users must be able to login/logout and update their personal information including name, profile picture, picture visibility (public/connections). Users should optionally change password and email, and any additional details as age, nationality etc.

Users must have interactions with other users. While viewing another user's profile, the user must be able to 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 role of connections will become clear in the next sections.

Users must be able to 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 their 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 must contain text, however, as an optional task you could add more functionality to the post. Some suggestions are:

- Upload a picture.
- Upload a song.
- Upload a video.
- Location.

Users must 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 could optionally create a more complex algorithm for feed generation. Some suggestions are:

- 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.
- You can generate the feed based on user location, and show higher in the feed posts, which are tagged closer to the user.
- Add the algorithm to the public feed section as well.

Users must have interactions with other user's posts:

- Users must be able to like or unlike (if you have already liked it) a post. The post total like count must be shown along the post content.

- Under each post there is a comment section. Users must be able to comment on a post. The newest N comments (choose the count to what seems best to you, but there must be a limit) are shown along the post content. Users can expand the comment section of the post to read older comments.

Administrative part

Accessible to users with administrative privileges.

Admin users must 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.

Optional features

Email Verification – In order for the registration to be completed, the user must verify their email by clicking on a link sent to their email by the application. Before verifying their email, users cannot make transactions.

Identity Verification – In order for the user registration to be completed, the user must submit a photo of their id card and a selfie. Users with administrator rights should have a page where they can view all users waiting for verification, review the photos they submitted and approve or reject them. Before being approved, users cannot make transactions.

*Note: **DO NOT** upload actual photos of id cards!*

REST API

To provide other developers with your service, you need to develop a REST API. It should leverage HTTP as a transport protocol and clear text JSON for the request and response payloads.

A great API is nothing without great documentation. The documentation holds the information that is required to successfully consume and integrate with an API. You must use Swagger to document yours.

The REST API provides the following capabilities:

1. Users
   - CRUD Operations (must)
   - Add/view/update/delete credit/debit card (must)
   - Block/unblock user (must)
   - Search by username, email, or phone (must)
2. Transactions
   - Add money to wallet (must)
   - Make transaction (must)
   - List transactions (must)
   - Filter by date, sender, recipient, and direction (in/out) (must)

- Sort by date or amount (must)

3. Transfers
   - Withdraw (must)

Technical Requirements

General

- Follow OOP principles when coding
- Follow KISS, SOLID, DRY principles when coding
- Follow REST API design best practices when designing the REST API (see Appendix)
- Use tiered project structure (separate the application in layers)
- The service layer (i.e., "business" functionality) must have at least 80% unit test code coverage
- Follow BDD when writing unit tests
- You should implement proper exception handling and propagation.
- Try to think ahead. When developing something, think – "How hard would it be to change/modify this later?"

Database

The data of the application must be stored in a relational database. You need to identify the core domain objects and model their relationships accordingly. Database structure should avoid data duplication and empty data (normalize your database).

Your repository must include two scripts – one to create the database and one to fill it with data.

Git

Commits in the GitLab repository should give a good overview of how the project was developed, which features were created first and the people who contributed. Contributions from all team members must be evident through the git commit history! The repository must contain the complete application source code and any scripts (database scripts, for example).

Provide a link to a GitLab repository with the following information in the README.md file:

- Project description
- Link to the Swagger documentation (must)
- Link to the hosted project (if hosted online)

- Instructions how to setup and run the project locally.
- Images of the database relations (must)

Optional Requirements

Besides all requirements marked as should and could, here are some more *optional* requirements:

- Use branching while working with Git.
- Integrate your app with a Continuous Integration server (e.g., GitLab's own) and configure your unit tests to run on each commit to the master branch.
- Host your application's backend in a public hosting provider of your choice (e.g., AWS, Azure, Heroku).

Teamwork Guidelines

Please see the Teamwork Guidelines document.

Appendix

- Guidelines for designing good REST API
- Guidelines for URL encoding
- Always prefer constructor injection
- Git commits - an effective style guide
- How to Write a Git Commit Message

Legend

- Must – Implement these first.
- Should – if you have time left, try to implement these.
- Could – only if you are ready with everything else try these.