Peer Review System

Final Project Assignment

Contents

General Project Requirements

Angular Final Project

This document describes the **final project assignment** for the **Angular** course at Telerik Academy.

Project Description

The members of every organization are responsible for the quality of the products. One mechanism for self-regulation and coverage of the organization's standards is the peer work review system. This could be in many forms, but in general it is a peer review of the submitted work item - code, tests, documents etc. The workflow is usually the following: user submits some work (code, documentation), assigns reviewers, reviewers can comment, approve, request changes or decline the work item. Once all reviewers approve the work item it is marked as complete. There might be different rules on how many reviewers must review the work item, the minimum approvals required (e.g. percentage of reviewers) before it is marked as approved. These rules might be enforced on different levels, i.e. team, project or company-wide.

Your task is to create such system using **Angular**. The system should allow submitting work items, assigning other users as reviewers. The reviewers can add comments, request additional changes, approve or decline the work item. Examples of such system is GitHub/Bitbucket Pull Request, GitLab Merge Request.

Public Part

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

- Application **MUST** have public homepage
- Application **MUST** have register functionality
- Application **MUST** have login functionality

Private Part (Users only)

Registered users must have private area accessible after **successful login**, where they could see all review requests that they've created, all review request where they are assigned as reviewers, a list of the teams the user is part of and optionally a list of pending team invitations (if no other notification method was implemented).

- Users **MUST** be able to create teams consisting of other users

- Users **MUST** see a list of its own peer review requests
- Users **MUST** see a list of pending reviews assigned to their team
- Users **MUST** be able to submit a work item, assign reviewers and trigger the review process
- Users **COULD** be able to see other team members review requests that the user is not assigned as a reviewer
- The team members **MUST** be able to view the newly created review request and its data. The assigned reviewers **SHOULD** receive a notification(email, popup or other in-app notification – all are fine)
- Team members **SHOULD** be able to invite other users into the team and each individual user **SHOULD** be able to leave a team. The invited users **SHOULD** receive notification.
- The review requests **COULD** have attachments functionality (upload screenshots, files, etc.).
- A user **COULD** be able to search for a review requests in the team's queue by: title, tag and reviewers.
- Once a review request status is changed all reviewers and the assignee **SHOULD** receive a notification. When a new comment is added or tag list is updated notification **SHOULD** be send as well.
- Each work item **MUST** have id, title, description, tags, assignee, reviewers, comments section, status. Once the work item is submitted for review it could be in one of the following statuses: Pending, Under Review, Change Requested, Accepted, Rejected.
- Status indicates whether it is approved by all reviewers or there are still pending reviews. There **COULD** be a team or company rule to determine if the review is its final state. For example: there might be rule that min 2 reviewers must approve the request to mark it as complete.

Administration Part (Optional requirement)

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

- Administrators **COULD** be able to create other administrators. The default admins could be pre-created, or event hardcoded in the system.
- Administrators **COULD** view all review requests and teams.
- Administrators **COULD** be able to close (accept or reject) any review request.
- Administrators **COULD** be able to add and remove any user to/from any team.
- Administrators **COULD** be able to see all teams' review requests.
- Any changes to status, assignee, ticket information (title and/or description) edits, user registration and user password change **COULD** be reflected in an audit log visible to administrators which **MUST NOT** be editable or modifiable.

Development Requirements

Your Web application should use the following technologies, frameworks and development techniques:

- Use **Angular** and **preferably Visual Studio Code**
- Create beautiful and responsive UI
  - Implement responsive UI using Bootstrap 3 or 4, or Materialize or don't use a framework at all
  - You may change the standard theme and modify it to apply own web design and visual styles
- Use modules to split your application logic
  - Core, Shared and Feature modules
- Create several **different pipes** and use them
- Create several **different directives** and use them
- Create several **modules** and use them in the **routing**
- Use guards to prevent the user to access the routes
- All of the data should be loaded from a web server using **services**
  - You can either use Firebase, Kinvey or any other back-end service.
  - Or you can use your own server written in Node.js, ASP.NET WebAPI or any other technology
- Unit test **a few components**
- Your project should pass the default TS linting configuration without any errors
- You can use Angular CLI
  - Or Webpack, SystemJS and any other module loader/bundler
- Your application should compile, work and produce an adequate result

- Use GitHub and take advantage of the **branches** for writing your features.
- **Documentation** of the project and project architecture (as .md file, including screenshots)

Optional Requirements

- Write integration tests
- Use lazy loading for the routing
    - Decide on the strategy used
- Use reactive forms
- Originality of the implementation (uniqueness)
- Host your application in the web (any public hosting provider of your choice)
- **BONUS**: “Progressify” your app (aka extend your app into a PWA – Progressive Web App)
    - There should not be any failed PWA audits in Google’s Lighthouse.
    - Opening your app on a mobile device should prompt the user to add it to the home screen.
    - Your app should display something (something simple like the logo and some text) when no network connectivity.
    - Your new and shiny PWA app should send desktop notifications in addition to the normal notifications you have implemented so far as a part of this assignment.

Deliverables

Put the following in a **ZIP archive** and submit it:

- The **source code** (everything except /bin/, /obj/, /packages/)
- The project documentation
- Screenshots of your application
- If hosted online - the URL of the application

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)
- Explain application structure and its **source code**

- Show the **commit logs** in the source control repository to prove a contribution from all team members

Many projects in the enterprise suffer from degradation of design and increasing complexity over time, leading them to develop defects and gradually become unmanageable.

We value greatly teams who are able to do their job cleanly with a logical and maintainable design, without either unnecessary abstraction or ad hoc hacks.

You need to understand the system you have created. Any defects or incomplete functionality must be properly documented and secured. It's OK if the proof of concept of 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 presentation:

- 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?