Telerik Academy

Safety Car

Team Project Assignment

1. Project Description

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

1.1. Overview

The goal of the project is to develop **SAFETY CAR** web application. It is insurance-oriented application for the end users. It allows simulation of amount and request insurance policies online. The application functionalities are related to three different type of users: public, private and administrators.

The **public part** of application should be visible without authentication. It should provide the following functionalities:

- Simulate of car insurance offer based on some input criteria
- Creation of new account and log-in functionality

The **private part** of application is for registered users only and should be accessible after successful login. The main functionalities provided for this area are:

- Send request for approval of offer (request policy)
- List history of user's requests

The **administrators** of the system should have permission to manage all major information objects in the system. The main functionalities provided for this area are:

- List of requests created in the system
- Approval or rejection of the requests

1.2. Business Glossary

Term	Description
Offer (Quote)	An estimate of expected value of insurance policy
Insurance Policy	A legal contract between insurer and insured person
Net premium	The amount of insurance policy to be paid by the client before taxes
Total Premium	The final amount of insurance policy to be paid by the client after taxes

2. Functional Description

In the following sections different functionalities will be described in more details.

2.1. Create Account

This functionality should be accessible only from **public par**t of the application.

Implement a page to register user in the system. The minimal data to be entered are: a valid email address as username and password.

Optional functionalities:

- Completion of registration via email confirmation link
- Provide constraints to create safe password

2.2. Simulate Offer

This functionality should be accessible from **public and private part**.

Implement a page – **simulation form**, where user should input specific parameters for its car and visualize the amount of expected total premium.

Simulation form should contain these business inputs:

- car (brand and model)
- cubic capacity
- first registration date
- driver age
- accidents in previous year (yes/no)

After click on simulate button, the expected premium should be calculated and shown on the screen.

The calculation of premium amount (**totalPremium**), is based on the following rules:

totalPremium = netPremium + taxAmount, where:

- netPremium = baseAmount * accidentCoeff * driverAgeCoeff
- taxAmount = 10% of netPremium

The calculation of net premium (**netPremium**) is based on the following parameters:

- **baseAmount** - amount which is calculated depending on two criteria from input form: cubic capacity and first registration date. Both input criteria parameters are range based (see details below)
- **accidentCoeff** – increase amount with 20% if there is an accident in previous year. The answer of form input parameter to be used.
- **driverAgeCoeff** - increase amount with 5% if driver is under 25 years old

For the calculation of the base amount you can use the following values in the multicriteria range table:

cc_min	cc_max	car_age_min	car_age_max	base_amount	Legend
0	1047	0	19	403.25	**Input criteria:** cc_min - cubic capacity (minimum) cc_max - cubic capacity (maximum) car_age_min - age of car (minimum) car_age_max - age of car (maximum) **Output parameters:** base_amount - base premium amount
0	1047	20	999	413.25	
1048	1309	0	19	529.63	
1048	1309	20	999	539.63	
1310	2356	0	19	690.96	
1310	2356	20	999	700.96	
2357	2880	0	19	862.86	
2357	2880	20	999	892.86	
2881	4188	0	19	957.89	
2881	4188	20	999	987.89	
4189	5497	0	19	1076.62	
4189	5497	20	999	1106.62	
5498	999999	0	19	1193.25	
5498	999999	20	999	1263.25	

Example: cubic capacity = 1400, car age = 10 => base amount = 690.96.

When the total premium amount is calculated, the user has the possibility to request this offer for approval. If the user has not been logged in, the system should force him to log in. If the user is logged, he automatically will be redirected to the page for policy request preparation.

Optional functionalities:

- Apply business controls to check data validity

2.3. Request policy

This functionality should be accessible only from **private part** of the application.

In order to issue a final policy, additional details are needed. A new page to be created in order to gather the following information:

- effective date of the policy
- attachment of image of vehicle registration certificate
- communication details: email, phone, postal address

All details from the simulated offer should be visible in the page.

Once the policy is requested it should be stored in the system for further management. At that stage request becomes in state “pending” and should be treated by the administrators of the system.

Optional functionalities:

- Apply business controls to check data validity

2.4. User's Request History

This functionality should be accessible only from **private part** of the application.

Implement a page to display list of all policy requests for the logged user with their details. The history of all requests to be shown (pending, approved and rejected) and chronologically sorted.

Optional functionalities:

- The user is able to cancel requests which are pending (not treated by administrator)

2.5. Manage Requests

This functionality should be accessible only from **administration part** of the application.

Implement a page to display list of all pending requests in the system with their details. The administrator should be able to accept or reject the request.

Optional functionalities:

- Possibility to filter requests by different criteria (user, request date, etc.)
- The data of multicriteria range table to be configured and loaded from external source
- Send email to the user in case of approval/rejection of the request

3. 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.

3.1. Web Application

Behind every great **web application** is a great **backend**. And the first impression your application creates is via the **frontend**.

It is your choice whether to use **Spring REST API + JavaScript UI (single page app)** or **Spring MVC + Thymeleaf** to create it.

Don't forget to follow good coding practices when implementing the REST API or the MVC pattern.

3.2. Database

All **SAFETY CAR** data should be stored in **MySQL database.**

4. Technical Requirements

Your project should meet the following requirements (these requirements will be used by TA trainers during project defense):

- Use **IntelliJ IDEA**
- For the web application (UI) you can choose from two approaches
    - Use **Spring MVC** Framework with **Thymeleaf** template engine for generating the UI
    - Research and build Single Page Application to serve the UI with JavaScript library of your choice. Please, keep in mind that we have not covered this in our sessions, so it is totally up to you how to implement the authentication and other SPA specifics that depend on the library you choose.
- Use **AJAX** form communication in some parts of your application
- Use **Dependency Inversion** principle and **Dependency Injection** technique following the best practices
- Use **MySQL (MariaDB)** as database back-end
    - Use **Hibernate** to access your database
        - If you do the research, you may use JPA or some other Hibernate-based framework.
    - Using **repositories and/or service layer** is a must
- Create **tables with data** with **server-side paging and sorting** for every model entity
    - You can use JavaScript library or generate your own HTML tables
- Create beautiful and interactive web UI
    - You may use **Bootstrap** or **Materialize**
    - You may change the standard theme and modify it to apply own web design and visual styles

- Apply **error handling** and **data validation** to avoid crashes when invalid data is entered (both client-side and server-side)
- Use **Spring Security** for managing users and roles
    - Your registered users should have at least one of the two roles: **user** and **administrator**
    - If you've chosen to develop SPA, you'd need to implement **JWT Authentication**.
- Create **unit tests** for your "business" functionality following the best practices for writing unit tests (**at least 80% code coverage**)
- Follow **OOP** best practices and **SOLID** principles.
- Use **Git** & **GitLab** for source control management – be careful with commit messages
- Use **Trello** for project management
- Create **user documentation / manual** of the application

5. 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)
- Use Dependency Inversion principle and Dependency Injection technique following the best practices (optional)

6. Deliverables

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

- Link to the Trello board
- Project documentation
- Screenshots of your application
- URL of the application (if hosted online)

7. 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**