Telerik Academy

BRIGHT CONSULTING

SIMPLY SERVICE MANAGEMENT

NGPuppies
Telerik Academy - Payment System

office@bright.consulting
www.bright.consulting
+ 359 2 971 11 17

Telerik Academy

BRIGHT CONSULTING

SIMPLY SERVICE MANAGEMENT

Spring MVC Final Project

This document describes the final project assignment for the Spring MVC course at

Telerik Academy.

Project Description

A large telecom needs a new flexible payment system. The old one is called “Puppies”, so the new one will be called **N**ew**G**eneration**Puppies**. Banks and other financial institutions (both called simply “Clients” within this document) could pay the monthly bills of their customers (simply called “Subscribers” within this document) using that application. Telecom bills should be available to the banks for payment either through GUI or through REST interface so that the clients of that application could make use of its APIs and generate automatic payments through their own systems without any human interactions.

Design and implement that Spring Application.

The application should have:

• public part (accessible without authentication)

• private part (available for registered users)

• administrative part (available for administrators only)

Public Part

The public part of your projects should be accessible without any authentication.

- The public part must have only user login API with two mandatory fields – username and password
- On successful login non-administrative users should be able to access private part
- On successful login administrative users should be able to access administrative part

Private Part (Users only)

Non-administrative users should have private part in the application, accessible when the user is authenticated

- They must have access to bill payment module where they can pay a particular bill (or selected list of bills) for their subscribers
- The client must be able to see personal details of a subscriber

- A client should be able to see a history of the payments for its subscribers sorted descending by the date of payment
- A client should be able to see the average and max amount of money payed for a subscriber for a defined period of time
- A client should be able to see a list of the services the client has paid for
- A client should be able to see some simple reports for its subscribers:
    - Top 10 subscribers with the biggest amount of money payed. If there are payments with amounts in different currencies, they should be all converted to BGN (using a static conversion rate) and summed
    - The 10 most recent payments for the particular client for all the subscribers
- Clients should be able to see and pay only the bills associated with their own subscribers
- Non-administrative users should not have the ability to change their username or password

Administration Part

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

- Administrators must be able to create non-administrative users with their own username and password.
- Administrators may be able to manage all the data related to a client
- Administrators must be able to create new administrative accounts defining their username, initial password and email address.
- Administrators must be forced to change their password on first login
- Administrators must be able to see a list of all users (no matter administrative or non-administrative)
- Administrators must be able to update information for non-administrative users
- Administrators must be able to delete both other administrative and non-administrative accounts
- Administrators must be able to generate payment with a given amount for any subscriber associated with any client.
- Administrators must have the ability to change their username, password and the email address they are registered with.

General Requirements

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

- A subscriber is uniquely identified by its phone number. It also has the following personal details related fields that could be statically populated in the database
    - First Name - mandatory
    - Last Name – mandatory
    - EGN (Uniform Civil Number) - mandatory
    - Address –optional
- For the sake of simplicity, you can assume that one subscriber can be associated with only one client.
- The client is uniquely identified by its username. It also has the following fields:
    - Password -mandatory
    - Details (like user friendly name, descriptions, etc.) -optional
    - EIK (Unified Identification Code) - mandatory
- A client can have only non-administrative account
- Administrative accounts are assigned only to the telecom employees
- For the sake of simplicity, you can assume that one client can have only one user in the system
- One billing record has the following fields:
    - Id – which should be automatically generated
    - Service – the service of the subscriber that the client pays for. Services should be selected from a predefined list of services
    - Start Date – the starting date of the period for which the subscriber has used a particular service
    - End Date – the ending date of the period for which the subscriber has used a particular service
    - Amount – the amount of money that should be paid for the particular bill

  - Currency – The currency for the amount. It should be a predefined list (e.g. EUR, USD, BGN)

- The UI and business logic module must be built independent in such a way that they could be deployed on separate servers
- The UI module and Business Logic module must communicate through REST interface and JSON
- Beautiful and responsive UI module is optional and if you support all the functionalities through REST APIs this is good enough
- Use IntelliJ IDEA or Eclipse
- Use Spring MVC framework or/and Spring Boot
- You may use Thymeleaf template engine for generating the UI
- You may build single-page application to serve the UI
- You can use JavaScript library of your choice
- Use MySQL (MariaDB) as database back-end
- Use Hibernate to access your database
- Using repositories and/or service layer is a must
- Create tables with data with server-side paging and sorting (if you need them) for every model entity
- You can use JavaScript library or generate your own HTML tables
- 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)
- You may use Spring Security for managing users and roles
- Your users should have exactly one role: either administrative or non-administrative
- Prevent yourself from security holes (XSS, XSRF, Parameter Tampering, etc.)
- Handle correctly the special HTML characters and tags

  like <script>,
, etc.

- Create unit tests for your "business" functionality following the best practices for writing unit tests (at least 80% code coverage)
- Use Inversion of Control principle and Dependency Injection technique.
- Follow OOP best practices and SOLID principles.
- Use GitHub and take advantage of the branches for writing your features.
- Create documentation of the project and project architecture (as .md file, including screenshots)

Optional Requirements (bonus points)

- Implement additional check for administrative users so that requests only from a predefined list of allowed IPs are accepted
- Implement the whole communication with the application using HTTPs protocol
- Develop JWT authentication
- Develop “forgotten password” functionality for the administrative accounts by using registered email addresses
- Develop additional verification step on top of “create new administrator” functionality so that One Time Password is sent to the registered email address of the creating administrating account and it is required to be confirmed after that in the application.
- Develop throttling functionality so that a particular client (user) has a limited number of (e.g. 3 TPS) transactions (requests) per second and after this value is reached the requests
- Expose all the business logic APIs through JMS interface (e.g ActiveMQ or other)
- 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
- Write integration tests and configure them to run as part of your CI build
- Deploy your application as Docker images
- Host your application in a public hosting provider of your choice (e.g. AWS)

Deliverables

Provide link to a GitHub repository with the following information:

- Project documentation
- Screenshots of your application
- URL of the application (If hosted online)

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

Final Projects Examples From Previous Years (.NET specific)

- http://best.telerikacademy.com/projects/557/MeetMe
- http://best.telerikacademy.com/projects/555/Car-System
- http://best.telerikacademy.com/projects/532/RememBeerMe-MVC
- http://best.telerikacademy.com/projects/515/BrumWithMe
- http://best.telerikacademy.com/projects/583/On-The-Road
- http://best.telerikacademy.com/projects/548/TravelGuide-v2
- http://best.telerikacademy.com/projects/531/Furniture-For-You-v2