Description

The objective of this exercise is to implement a top-up functionality using technologies mentioned below. Candidate should use provided High Level Design as reference, following the document as close as possible. If any information is missing, candidate should clearly identify it and describe the assumptions he took.

Celfocus provides a full functioning backend layer, based on Spring Boot, which the candidate should use to run the application - https://bitbucket.org/celfocus/cf-interview/overview. The final solution should be submitted to the same repository.

Technologies

Frontend

HTML5, CSS3, JavaScript

Candidate can use also one of the following JavaScript frameworks: jQuery, Vue.js, React, Angular;

Backend

Java, Json, XML, Soap, Rest

Candidate can use also one of the following Java frameworks: Spring

Candidate should consult with Celfocus if he intends to use any framework that is not explicated listed.

The submission will be evaluated based on the criteria identified in the next pages, so candidate should read them carefully.

Any question related to the exercise should be submitted to luis.alberto.almeida@celfocus.com. Response will be provided in timely fashion (within Portugal business hours).

Evaluation Criteria

1. Completeness
   How close the submission matches the High Level Design and overall exercise intent?

2. Readability
   How easy is the site for another experience developer to "read"- are functions well named, is there good commenting in the code base, is the code well formatted across different developers?

3. Adaptability
   How easy is it to make minor, common changes to the site? For example, how easy is it to add new fields, add additional content or choices, or change the user interface?

4. Usability
   Is the site easy to use by core users? Is the site optimized for the most common users? Are there common data validation tools to avoid bad data entry (such as requiring date formats?) Does the site conform to common standards of user interface design and accessibility? Does it look professional and trustworthy for the audience?

5. Code quality
   Does the code follow best practices and coding guidelines? Are unit tests and exception handling implemented to make sure that any error is detected and treated in a correct way?

6. Performance
   How fast is the application? Does it hang or time out? Do the pages' load in a timely fashion? Does it scale when loads get high or when running CPU intensive functions?

7. Security
   What are the security and privacy protections in place against unlawful access? Are there logs, password encryption, SSL certificates, or separation of personally identifiable information? Have common security processes been implemented against common hacking vectors, such as sanitizing your inputs?

Additionally, the following should be taken in consideration by the candidate to make sure the code submitted is readable and in general easy to analyze by the evaluating team:

1. Effective class, method, and variable names
   Names chosen for classes, methods, and variables should effectively convey the purpose and meaning of the named entity;

2. Effective top-down decomposition of algorithms
   Code duplication should be avoided by factoring out common code into separate routines. Routines should be highly cohesive. Each routine should perform a single task or a small number of highly related tasks. Routines that perform multiple tasks should call different subroutines to perform each subtask. Routines should be relatively short in most cases;

3. Code layout should be readable and consistent
   The layout of your code should be readable and consistent. This means things like placement of curly braces, code indentation, wrapping of long lines, layout of parameter lists, etc.;

4. Effective source tree directory structure
   The source code for your project should be effectively organized into subdirectories. Something along the lines of that discussed in class would be appropriate;

5. Effective file organization
   Your source code should be effectively organized into multiple files. Each class should be placed in a separate file. Lumping all of your code in one or two files is not acceptable;

6. Correct exception handling
   Your program should handle exceptions properly to treat any error in graceful way;

7. Good unit test cases
   Unit test cases should be implemented to cover all application layers. In this exercise at least java unit tests should be implemented;