| Assignment 1 | | Project Summary | |
| --- | --- | --- | --- |
| Course | | **Web Application & Service Development with Spring Framework v5** **- 2020** | |
|  | | | |
| Project author | | | |
| № | Pseudonym | | Face-to-face/ online |
| 1 | ivanasen | | face-to-face |

| Project name | Competitive Programming System |
| --- | --- |

| 1. Short project description (Business needs and system features) |
| --- |
| The ***Competitive Programming System (CPS)*** provides the ability for users solve programming tasks in a programming language (will implement only for Python) on a remote machine. The end goal is to have a system similar to **Hackerrank** but a lot simpler of course. The system will be developed using ***Spring 5 Application Development Framework***.It will implement a web-based front-end client using ***React.js***. Each page will have a distinct URL, and the routing between pages will be done client-side. The front-end will be a Single-Page Application. Server-Side will be using ***Spring WebFlux***. The backend will be implemented as a ***REST/JSON API*** using JSON data serialization. Users can monitor who has solved a task and with how many points in real-time using **Server-Sent Events.** Creators can also view the solutions of the users. The main user roles are:  • *Anonymous User* – can only view things. Can NOT submit solutions or create new problems.  • *User* – can view things. Can submit new solutions to problems  • *Creator* – can create new problems *with descriptions and tests.*  • *Administrator* – will have full CRUD operations allowed for all resources - Users, Problems, Solutions |

| 1. Main Use Cases / Scenarios | | |
| --- | --- | --- |
| **Use case name** | **Brief Descriptions** | **Actors Involved** |
| * 1. **Browse information** | The *User* can browse the information views (Home, Problems, Solutions). | All users |
| * 1. **Register** | *Anonymous User* can register in the system by providing a valid e-mail address, first and last name, and choosing password. By default, all new registered users have *User* role.  *Administrator* can register new by entering *User Data* and choosing a Role (*User, Creator,* or *Administrator*). | *Anonymous User, Administrator* |
| * 1. **Change User Data** | *Registered User* can view and edit his personal *User Data.*  *Administrator* can view and edit *User Data* of all *Users* and assign them *Roles*: *User, Creator*, or *Administrator*. | *User, Administrator* |
| * 1. **Manage Users** | *Administrator* can browse and filter users based on different criteria: first and last name, email, Role.  *Administrator* can choose a *User* to manage, and can manage the chosen User - edit (using Change User Data UC) or delete.  *Administrator* can create a new user using *Register UC*. | *Administrator* |
|  |  |  |
| * 1. **Manage Problems** | *Creators can create new problems with descriptions, name, well-defined tests. They can also update and delete their own problems. They can view all problems.*  *Administrator*s have full CRUD operations on problems | *Creator, Administrator* |
| * 1. **Make a submission to a problem** | *Instructor* can browse her/his *Tests*, add new *Test* using *Add/Edit Test UC*, and delete a *Test*, as well as view the *Student's Test Results* for her own *Tests*.  *Administrator* can browse test of all *Instructors*, edit and delete them. | *Creator, Administrator* |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| * 1. **Browse Submission Results** | *User* can browse the results of all submissions. They can’t view the code, however.  *Creators can also view the code* | *All users* |

| 1. Main Views (Frontend) | | |
| --- | --- | --- |
| **View name** | **Brief Descriptions** | **URI** |
| * 1. **Home** | Presents the introductory information for the purpose of the system as well as detailed instructions on how to start using it. Prominently offers ability to register. | / |
| * 1. **Problems** | Presents tests available according to *User's Role* and identity. Offers abilities to browse, choose, create, read, update, delete (CRUD) Tests, as defined by *UCs* (for *Administrators* and *Instructors* only). | */problems* |
| * 1. **Submissions** | Presents tests available to particular Student (according to *Student Group* participation) allowing him/her to choose a test and complete it. | */submissions* |
|  |  |  |
| * 1. **User Registration** | Presents a view allowing the *Anonymous Users* to register | */register* |
| * 1. **Login** | Presents a view allowing the users to login | */login* |
| * 1. **User Data** | Provides ability to view and edit personal *User Data* | */personal* |
|  |  |  |
|  |  |  |
| * 1. **Users** | Presents ability to manage (CRUD) Users and their User Data (available for *Administrators* only, as described in UCs). | */users* |
| * 1. **About** | Presents information about the project and his owner. | */about* |

| 1. API Resources (Backend) | | |
| --- | --- | --- |
| **View name** | **Brief Descriptions** | **URI** |
| * 1. **Users** | GET *User Data* for all users, and POST new *User Data* (Id is auto-filled by *OKTS* and modified entity is returned as result from POST request). Available only for *Administrators*. | */api/users* |
| * 1. **User** | GET, PUT, DELETE *User Data* for *User* with specified *userId*, according to restrictions described in UCs. | */api/users/{userId}* |
| * 1. **Login** | POST *User Credentials* (e-mail address and password) and receive a valid *Security Token* to use in subsequent API requests. | */api/login* |
| * 1. **Logout** | POST a logout request for ending the active session with *OKTS,* and invalidating the issued *Security Token*. | */api/logout* |
| * 1. **Problems** | GET *Problems*, and POST new *Problems* (Id is auto-filled by *server* and modified entity is returned as result from POST), according to *User's Role* and identity security restrictions. | */api/problems* |
| * 1. **Problem** | GET, PUT, DELETE *Problem* for *Problem* with specified *problemId*. | */api/problems/{problemId}* |
| * 1. **Submissions** | GET submissions and POST new *Submission* (Id is auto-filled by *server* and modified entity is returned as result from POST request). This results in the code being executed | */api/submissions* |
| * 1. **Submission** | GET, PUT, DELETE *Submission*for *Test* with specified *testId*. | */api/submissions/{submissionId}* |
| * 1. **Submission Code** | GET *Submission Code* (according to *User's Role* and identity) ) for *Submission* with specified *submissionId* | */api/submissions/{submissionId}/code* |
|  |  |  |
| * 1. **Active Submissions** | SSE event streaming of Students’ progress on currently active *Submissions* (separate event pushed for each *Submission result*), according to *User's Role* and identity security restrictions. | */api/active-submissions* |