|  |  |  |  |
| --- | --- | --- | --- |
| Assignment 1 | | Project Summary | |
| Course | | React.js - 2020 | |
|  | | | |
| Project author | | | |
| № | Pseudonym | | Face-to-face/ online |
| 1 | Instructor | | face-to-face |

|  |  |
| --- | --- |
| Project name | Cooking Recipes |

|  |
| --- |
| 1. Short project description (Business needs and system features) |
| The main goal of the project is to implement a web application allowing the Users (HomeCooks and Administrators) to share different Recipes online, and Comment about their personal experience and ideas, when cooking and enjoying them. The main user roles are:  • *Anonymous User* – can only read the Recipes;  • *Registered User* – can read and comment the Recipes. This user should be able to manage their own personal data and change their passwords;  • *Administrator* (extends *Registered User*) – can read, add, edit and delete the Recipes. This user should be able to manage all User's data. |

|  |  |  |
| --- | --- | --- |
| 1. Main Use Cases / Scenarios | | |
| **Use case name** | **Brief Descriptions** | **Actors Involved** |
| * 1. **Browse information** | The *Anonymous* *User* can only read the Recipes. | 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 *HomeCooks* role.  *Administrator* can register new by entering *User Data* and choosing a Role (*HomeCooks* or *Administrator*). | *Anonymous User, Administrator* |
| * 1. **Change User Data** | *Registered User* can view and edit her personal *User Data.*  *Administrator* can view and edit *User Data* of all *Users* and assign them *Administrator* *Role.* | *Registered User, Administrator* |
| * 1. **View Recipes** | *Registered User* can read and comment the Recipes. | *Registered User* |
| * 1. **Add/Edit/Delete Recipes** | *Administrator* can read, add, edit and delete the Recipes. | *Administrator* |

|  |  |  |
| --- | --- | --- |
| 1. Main Views (SPA Frontend) | | |
| **View name** | **Brief Descriptions** | **URI** |
| * 1. **Home** | The user can see all available Recipes. | / |
| * 1. **Recipe** | Presents a view with a selected Recipe. | */*recipe |
| * 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** | Presents 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). | */users* |
| * 1. **Add recipe** | Allows the Administrator to add a new Recipe. | */add* |
| * 1. **Edit recipe** | Allow the Administrator to edit a Recipe. | */edit* |

|  |  |  |
| --- | --- | --- |
| 1. API Resources (Node.js 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 decribed 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. **Student Groups** | GET *Student Groups*, and POST new *Student Group* (Id is auto-filled by *OKTS* and modified entity is returned as result from POST request), according to *User's Role* and identity security restrictions. | */api/groups* |
| * 1. **Student Group** | GET, PUT, DELETE *Student Group* (including assigned students) for *Student Group* with specified *groupId*. | */api/tests/{groupId}* |
| * 1. **Tests** | GET users (according to *User's Role* and identity) and POST new *Test* (Id is auto-filled by *OKTS* and modified entity is returned as result from POST request). | */api/tests* |
| * 1. **Test** | GET, PUT, DELETE *Test Data* (including *Questions* and *Answers*) for *Test* with specified *testId*. | */api/tests/{testId}* |
| * 1. **Test Results** | GET *Test Results* (according to *User's Role* and identity) ) for *Test* with specified *testId*, and POST new *Test Result* (Id is auto-filled by *OKTS* and modified entity is returned as result from POST request). | */api/tests/{testId}/results* |
| * 1. **Test Result** | GET, PUT, DELETE *Test Result* (according to *User's Role* and identity) for *Test* with specified *testId* and *Test Result* with specified *testResultId.* | */api/tests/{testId}/results/ {testResultId}/* |
| * 1. **Active Tests** | SSE event streaming of Students’ progress on currently active *Tests* (separate event pushed for each *Question Answer*), according to *User's Role* and identity security restrictions. | */api/active-tests* |