|  |  |  |  |
| --- | --- | --- | --- |
| Assignment 1 | | Project Summary | |
| Course | | Fullstack Application Development with Node.js + Express.js + React.js - 2019 | |
|  | | | |
| Project author | | | |
| № | Pseudonym | | Face-to-face/ online |
| 1 | HHT | | face-to-face |

|  |  |
| --- | --- |
| Project name | Online Car Manager |

|  |
| --- |
| 1. Short project description (Business needs and system features) |
| The Internet age changed the way we learn and test our knowledge. **The Online Car Manager**provides ability for car owners to track their car history, and for service owners to compete. In addition to that it allows users to register, and administrators to manage them. The system will be developed as a *Single Page Application (SPA)* using ***React.js*** as front-end, and ***Node.js + express*** as backend technologies. Each view will have a distinct URL, and the routing between pages will be done client side using ***React Router***. The backend will be implemented as a ***REST/JSON API*** using JSON data serialization. There will be also a real-time event streaming from the server to the web client using ***Socket.IO*** and ***Server Sent Events (SSE)/WebSocket*** in order to allow the Instructor to monitor test completion progress of students in real time. The main user roles (actors in UML) are:   * Car Owner(User Type 1) - This user represents a normal person who has car/s. Main features:   + Personal diary to keeps a record of his/her car/s   + Possibility to choose from all service owners offering services * Service Owner(User Type 2) – This user represent a service owner who offer paid services. Main features:   + Competitive market   + Manage daily/week/month appointments   + Manufacture different repair specifications for different type/model cars * Arbitrator(Admin) – Main features:   + Resolving problems between car owners and service owners   + Controlling the monopoly laws |

|  |  |  |
| --- | --- | --- |
| 1. Main Use Cases / Scenarios | | |
| **Use case name** | **Brief Descriptions** | **Actors Involved** |
| * 1. **Browse information and try sample tests** | The *User* can browse the information views (Home, Tests, About) in *OKTS*, and can choose to try sample tests using the *Complete Test UC* . | All users |
| * 1. **Register** | *Three separate type of registration:*   * *Car Owner – required valid e-mail address, mobile phone other personal info.* * *Service Owner – required valid license for car service* * *Arbitrator – required special code for registration* | *Car Owner,*  *Service Owner,*  *Arbitrator* |
| * 1. **Add car** | *Car Owner can add car to the cars which he manage.* | *Car owner* |
| * 1. **Write a note** | *Car Owner with at least one car can add notes for maintenance.* | *Car Owner* |
| * 1. **Compare services** | *Car Owner can compare different services in order to find the most suitable one.* | *Car Owner* |
| * 1. **Make appointment** | Car Owner can choose from all of the available services in the market and make an appointment for the most suitable one. | *Car Owner* |
| * 1. **Add service** | *Service Owner can add new service by providing legal documents for the service* | *Service Owner* |
| * 1. **Make repair configuration** | *Service Owner can add special repair configuration for specific type/model car or make general price list or both.* | *Service Owner* |
| * 1. **Manage appointments** | *Service Owner can manage the appointments for the day/week/month* | *Service Owner* |
| * 1. **Add record in car dairy** | *Service Owner must add a record in the car dairy of the Car Owner and guarantee for the used parts* | *Service Owner* |
| * 1. **Validate Info** | *Arbitrator decides wherever the given info from the Car Owners and Service Owners is valid* | *Arbitrator* |

|  |  |  |
| --- | --- | --- |
| 1. Main Views | | |
| **View name** | **Brief Descriptions** | **URI** |
| * 1. **Home** | Presents the introductory information for the purpose of the system as well as detailed instructions how to start using it. Prominently offers ability to register. | / |
| * 1. **Car Diary** | Presents all custom made notes and notes made by mechanic | */diary* |
| * 1. **Compare Services** | Presents basic info for all services and possibility to compare several services | */compare* |
| * 1. **Appointments** | Presents *all upcoming appointments* | */calendar* |
| * 1. **User Registration** | Presents a view allowing the user to choose as what wants to register | */register* |
| * 1. **Login** | Presents a view allowing the users to login. | */login* |
| * 1. **User Data** | Presents the main information about the profile and some basic statistical data. | */personal* |
| * 1. **Service management** | Presents the current service price list with possibility to make special custom configuration for repair. | */service-manager* |
| * 1. **Dashboard** | Presents in real time progress of the current repair made by any service to any car. | */dashboard* |
| * 1. **About** | Presents information about the project and his owner. | */about* |

|  |  |  |
| --- | --- | --- |
| 1. API Resources (Node.js Backend) | | |
| **View name** | **Brief Descriptions** | **URI** |
| * 1. **Users** | GET *User Data* for all users, and POST new *User Data.*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* |