|  |  |  |  |
| --- | --- | --- | --- |
| Assignment 1 | | Project Summary | |
| Course | | Fullstack Application Development with Node.js + Express.js + React.js + MongoDB (MERN) - 2020 | |
|  | | | |
| Project author | | | |
| № | Gordon Ramsay | | Face-to-face/ online |
| 1 | Trayan Iliev | | face-to-face |

|  |  |
| --- | --- |
| Project name | Food Waste Reducer |

|  |
| --- |
| 1. Short project description (Business needs and system features) |
| With the world’s population on the constant rise, we need to come up with radical ideas to reduce our carbon footprint and lead a more sustainable lifestyle. The ***Food Waste Reducer (FWR)*** provides ability for restaurants to register, create a “leftover’s” menu for the given day, and either give away their daily food waste or sell it for a fraction of the real price. The customer can simply register/log in, find a tasty meal, book it and go to restaurant’s location to pick it up. In addition to that ***FWR*** allows administrators to manage restaurant’s profiles and their menus/posts. 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***. ***Redux*** will be used, in case ***FWR*** needs a more specific state management. The backend will be implemented as a ***GraphQL***/***REST/JSON API*** using JSON data serialization. All the data will be stored in ***MongoDB***. The main user roles (actors in UML) are:  • *Anonymous User* – can only view the information pages and search for a specific restaurant without being able to book a meal or comment on restaurant’s page.  • *Restaurant* (extends *Registered User*) – can make a Menu and specify pick-up time for the meals  • *Customer* (extends *Registered User*) – can book meals and post useful comments  • *Administrator* (extends *Registered User*) – can manage (create, edit user data and delete) all *Registered Users*, as well as *Menus, Posts, Comments*. |

|  |  |  |
| --- | --- | --- |
| 1. Main Use Cases / Scenarios | | |
| **Use case name** | **Brief Descriptions** | **Actors Involved** |
| * 1. **Browse restaurants and their daily offers** | The *User* can browse the home and restaurant views (Home, Restaurant’s Name, About) in *FWR*, can view restaurant’s menu and their comment sections. | All users |
| * 1. **Register** | *Restaurants* can register in the system by providing a valid e-mail address, name of the restaurant, and choosing password.  Customers can also register by providing a valid e-mail address, first and last name, and a password.  *Administrator* can register new by entering *User Data* and choosing a Role (*Restaurant, Consumer*, or *Administrator*). | *Restaurant, Customer, Administrator* |
| * 1. **Change User Data** | *Restaurants* can view and edit their personal *User Data* as well as enter more meals to their menus.  *Administrator* can view and edit *User Data* of all *Users* and assign them *Roles*: *Restaurant, Customer* or *Administrator*. | *Restaurant, Administrator* |
| * 1. **Manage Users** | *Administrator* can browse and filter users based on different criteria: restaurant’s 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. | *Administrator* |
| * 1. **Manage Food Categories (main page)** | *Administrator* can browse and filter *Food Categories* based on name.  *Restaurants* can create new Food Categories.  *Administrator* can choose a *Food Category* to delete if necessary. | *Administrator, Restaurant* |
| * 1. **Manage Restaurant’s Menu (restaurant’s view)** | *Restaurants* can browse and update their *Menu*, add new *Posts* using *Add/Edit Menu UC*, and delete a *Post*, as well as view Customers’ commentsfor their own *Menu* and for a particular *Post.*  *Administrator* can browse menus of all *Restaurants*, edit and delete them if necessary. | *Restaurant, Administrator* |
| * 1. **Add/Edit Comments** | *Restaurant* or *Administrator* can edit or delete comments under Posts or Menus. | *Restaurant, Administrator* |
| * 1. **Block customer’s comments** | If a *Customer* posts comments, which violate the terms and conditions, Administrator can block him/her/it from commenting any further in the web app. | *Administrator* |

|  |  |  |
| --- | --- | --- |
| 1. Main Views (SPA Frontend) | | |
| **View name** | **Brief Descriptions** | **URI** |
| * 1. **Home** | Presents the different food categories, posts for each category, pick-up time and the restaurant owning it. Prominently offers ability to register. | / |
| * 1. **Restaurants** | Presents all registered restaurants. Offers abilities to browse, choose (for Customers), and - create, read, update, delete (CRUD) Restaurants, as defined by *UCs* (for *Administrators* only). | */restaurants* |
| * 1. **Restaurant** | Presents information about a specific restaurant, it’s menu, posts, pick-up time and comment section.  Offers CRUD abilities for Menus and Posts only to the specific Restaurant User. | */restaurantName* |
| * 1. **User Registration** | Presents a view allowing the *Restaurants and Consumers* to register in *FWR*. | */register* |
| * 1. **Login** | Presents a view allowing the users to login. | */login* |
| * 1. **Dashboard** | Presents in real time the Restaurant’s pick-up times. | */dashboard* |
| * 1. **About** | Presents information about the *FWR* project and his owner. | */about* |

|  |  |  |
| --- | --- | --- |
| 1. API Resources (Node.js Backend) | | |
| **View name** | **Brief Descriptions** | **URI** |
| * 1. **Restaurants** | GET *Restaurant Data* for all Restaurants | */api/restaurants* |
| * 1. **Restaurant** | GET, PUT, DELETE *Restaurant Data* for *Restaurant* with specified *restaurantId*, according to restrictions decribed in UCs, and POST new Menu (Id is auto-filled by *FWR* and modified entity is returned as result from POST request). | */api/restaurants/{restaurantId}* |
| * 1. **Register** | POST the required Data for Restaurants or Customers and receive a valid *Security Token* to use in subsequent API requests. | */api/register* |
| * 1. **Login** | POST *Restaurant* or Customer *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 *FWR,* and invalidating the issued *Security Token*. | */api/logout* |
| * 1. **Food categories** | GET *Food Categories*, and POST new *Food categories* (Id is auto-filled by *FWR* and modified entity is returned as result from POST request), according to *User's Role* and identity security restrictions. | */api/food-categories* |
| * 1. **Food category** | GET, PUT, DELETE *Food Category* (including assigned meals) for *Food Category* with specified *categoryId*. | */api/ food-categories /{categoryId}* |
| * 1. **Food category - meals** | GET all the Meals for the given Food Category | */ api/ food-categories /{categoryId}/meals* |
| * 1. **Menus** | GET, PUT, DELETE Menus and POST new Meals (Id is auto-filled by FWR and modified entity is returned as result from POST request). | */api/menus* |
| * 1. **Comments** | GET, DELETE all the Comments for a specific restaurant and POST new Comment (Id is auto-filled by *FWR* and modified entity is returned as result from POST request). | */api/{restaurantId}/comments* |
| * 1. **Comment** | GET, PUT, DELETE a Comment with specified commentId. | */api/{restaurantId}/comments/{commentId}* |