|  |  |  |  |
| --- | --- | --- | --- |
| Assignment 1 | | Project Summary | |
| Course | | Fullstack Application Development with Node.js + Express.js + React.js - 2022 | |
|  | | | |
| Project author | | | |
|  | Name | | FN |
|  | Dimitar Mihaylov | | 71773 |

|  |  |
| --- | --- |
| Project name | Agile Issue Tracking Assistant |

|  |
| --- |
| 1. Short project description (Business needs and system features) |
| Every project needs a board that allows issue tracking and agile project management. The ***A(g)ile (I)ssue T(r)acking (A)ssistant (gira)*** provides project managers the ability to create and manage projects, keep track of issues that the developer team is working on. 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. The main user roles are:  • *Anonymous User* – can only view dashboard and login pages.  • *Developer* – can create/take an *Issue*– move it across the board  • *Project Manager (PM)* – can create *Projects, Issues,* assign users to *Projects* that he manages.  • *Business Analyst (BA) – can create Issues, approve/decline success of Issues*  • *Administrator* – can create, edit *Users* and *Projects* |
|  |

|  |  |  |
| --- | --- | --- |
| 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 (Dashboard, Login, About) in *gira*, | 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 *Developer* role.  *Administrator* can register new Developer by entering *User Data* and choosing a *Project* to assign him to. Also can create *Project manager*.  *PM* can assign existing *Developer* to *Project* | *Anonymous User, Administrator, PM* |
| * 1. **Change User Data** | *User* can view and edit his personal *User Data*  *Administrator* can view and edit *User Data* of all *Users* and assign them to *Projects* or promote them to *PM* or *Administrator*. | *Registered User, Administrator* |
| * 1. **Manage Users** | *Administrator, PM* can browse and filter users based on different criteria: first and last name, email, Project.  *Administrator* can choose a *User* to manage, and can manage the chosen User - edit (using Change User Data UC) or delete.  *PM* can assign/remove a user to/from a *Project.*  *Administrator* can create a new user using *Register UC*.  *Administrator* can edit *PM’s* assigned projects | *Administrator, PM* |
| * 1. **Manage Projects** | *Administrator* or *PM* can browse and filter *Projects* based on different criteria: name*,* creation date, etc.  *Administrator* or *PM* can create a new *Project*  *Administrator* can choose a *Project* to manage, and can add or remove *Developers* or *PMs* to the *Project,* or delete it if necessary. | *Administrator, Instructor* |
| * 1. **Add/Edit Issue** | *PM* or *Administrator* specifies/edits *Issue* meta-data such as: name, priority, description, tags, and assigns it to a *Developer* or leaves it unassigned | *Instructor, Administrator* |
| * 1. **Manage Issues** | *Developer* can assign an *Issue* to himself, move it through different stages on the board, complete issue.  *PM* and *Administrator* can do the same | *PM, Administrator, Developer* |
| * 1. **Complete Issue** | *PM or Administrator* can complete any *Issue.*  *Developer* can complete only issues that are assigned to him | *Administrator, PM, Developer* |

|  |  |  |
| --- | --- | --- |
| 1. Main Views (SPA Frontend) | | |
| **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. **Projects** | 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). | */projects* |
| * 1. **Board** | Board of a project with all Issues separated by stages (todo, in progress, in review, done) | */project/{id}/board* |
| * 1. **User Registration** | Presents a view allowing the *Anonymous Users* to register in *gira*, as well as to enroll to a *Student Group* by presenting a *Token Key* for that group. | */register* |
| * 1. **Login** | Presents a view allowing the users to login. | */login* |
| * 1. **User Data** | Presents ability to view and edit personal *User Data*, enroll to more *Student Groups* by presenting a *Token Key* for particular *Student Group*, as well as deregister from *gira*. | */profile* |
| * 1. **Dashboard** | *Developer*- List of *Developer’s* issues and the stages they are in.  *PM* - List of *PM* projects | */dashboard* |
| * 1. **Users** | Presents ability to manage (CRUD) Users and their User Data (available for *Administrators* only, as described in UCs).  Ability to assign *Developer* to *Project* available for *Administrator* and *PM* | */users* |
| * 1. **About** | Presents information about the *gira* 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* (Id is auto-filled by *gira* and modified entity is returned as result from POST request). for *Administrators* and *PMs*.  *PMs* can only see unassigned *Developers* or assigned to his *Projects* | */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 *gira,* and invalidating the issued *Security Token*. | */api/logout* |
| * 1. **Projects** | GET *Projects*, and POST new *Project* (Id is auto-filled by *gira* and modified entity is returned as result from POST request), according to *User's Role* and identity security restrictions. | */api/projects* |
| * 1. **Project** | GET, PUT, DELETE *Project* with specified *projectId*.  DELETE unassigns the Project from the Developers/PMs | */api/projects/{projectId}* |
| * 1. **Issues** | GET *Issues* (according to *User's Role* and identity) and POST new *Issue* (Id is auto-filled by *gira* and modified entity is returned as result from POST request). | */api/projects/{projectId}/issues* |
| * 1. **Issue** | GET PUT DELETE *Issue* with specified *issueId* in specified project with *projectId*. | */api/projects/{projectId}/issues/{issueId}* |
| * 1. **Sprint** | GET all sprints for given project visible by Developers, PMs, Administrators  POST create new sprint – possible only for PM and Admin | */api/projects/{projectId}/sprints* |
| * 1. **Sprint** | GET one sprint visible by Dev,PM,Admin  PUT – PM, Admin | */api/projects/{projectId}/sprints/{sprintId}* |