**e-Project**

**Sprint #2 Report**

**Team Members:**

Lazaron Shyta - 100195078

Gabriel Kuka - 100189808

**Intro**

The set of user stories assigned for the sprint:

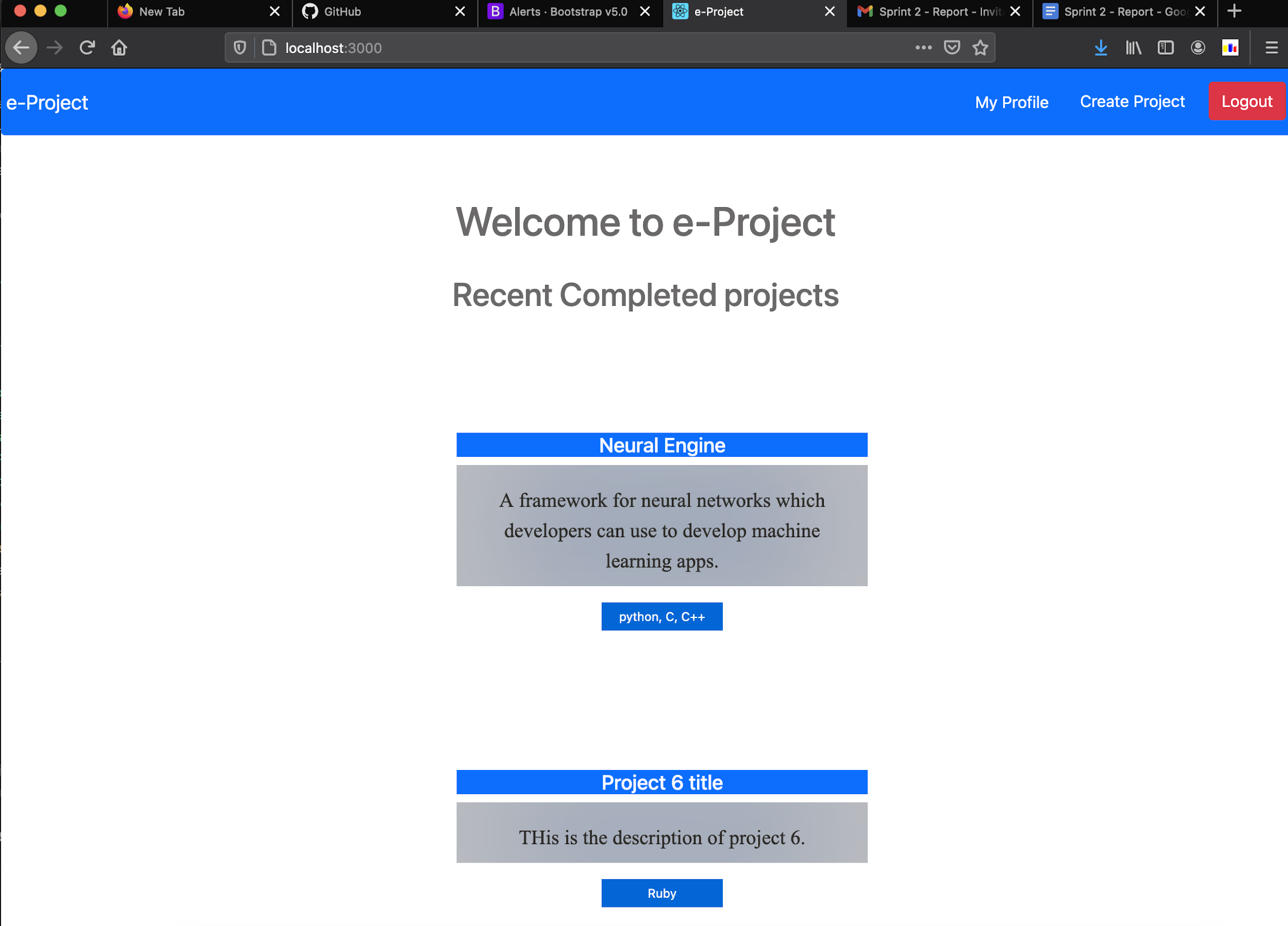
1. Integrate Bootstrap in the website
2. Add Feature to Create Projects
3. List The 5 recently Completed

**1 Integrating Bootstrap in the website**

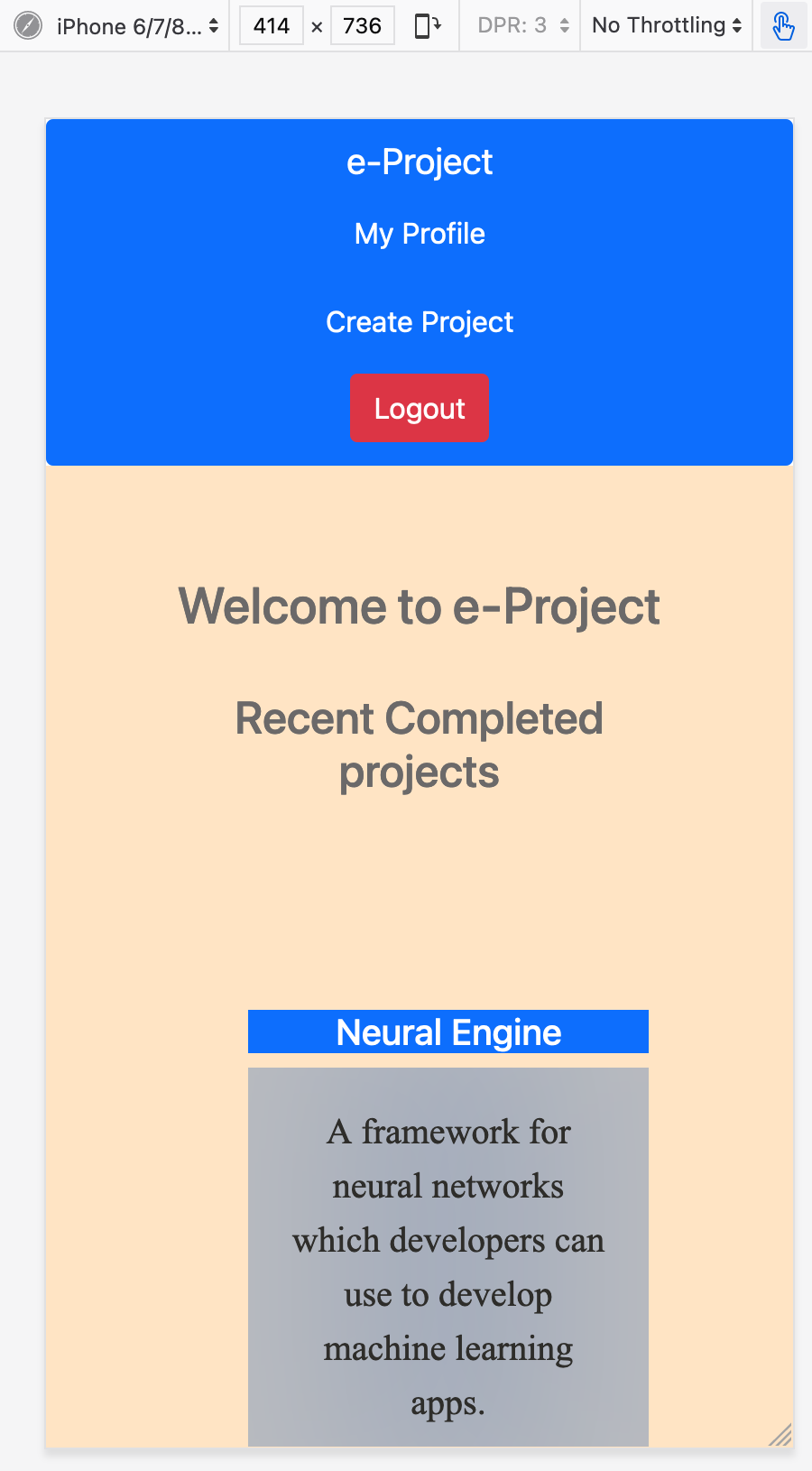
We integrated Bootstrap 5 in our React Frontend. For this, we installed the following dependency in our package.json: bootstrap: "^5.0.0-alpha1", also include it in the index.js file: import '../node\_modules/bootstrap/dist/css/bootstrap.min.css'

and for certain components in our application, we used the Bootstrap 5 documentation. Since React still does not natively support Bootstrap 5, we had to use the default Bootstrap 5 class names. For example the button syntax is <button type='submit' className='btn btn-primary'></button>.

That's a preview of landing page in a full width screen:

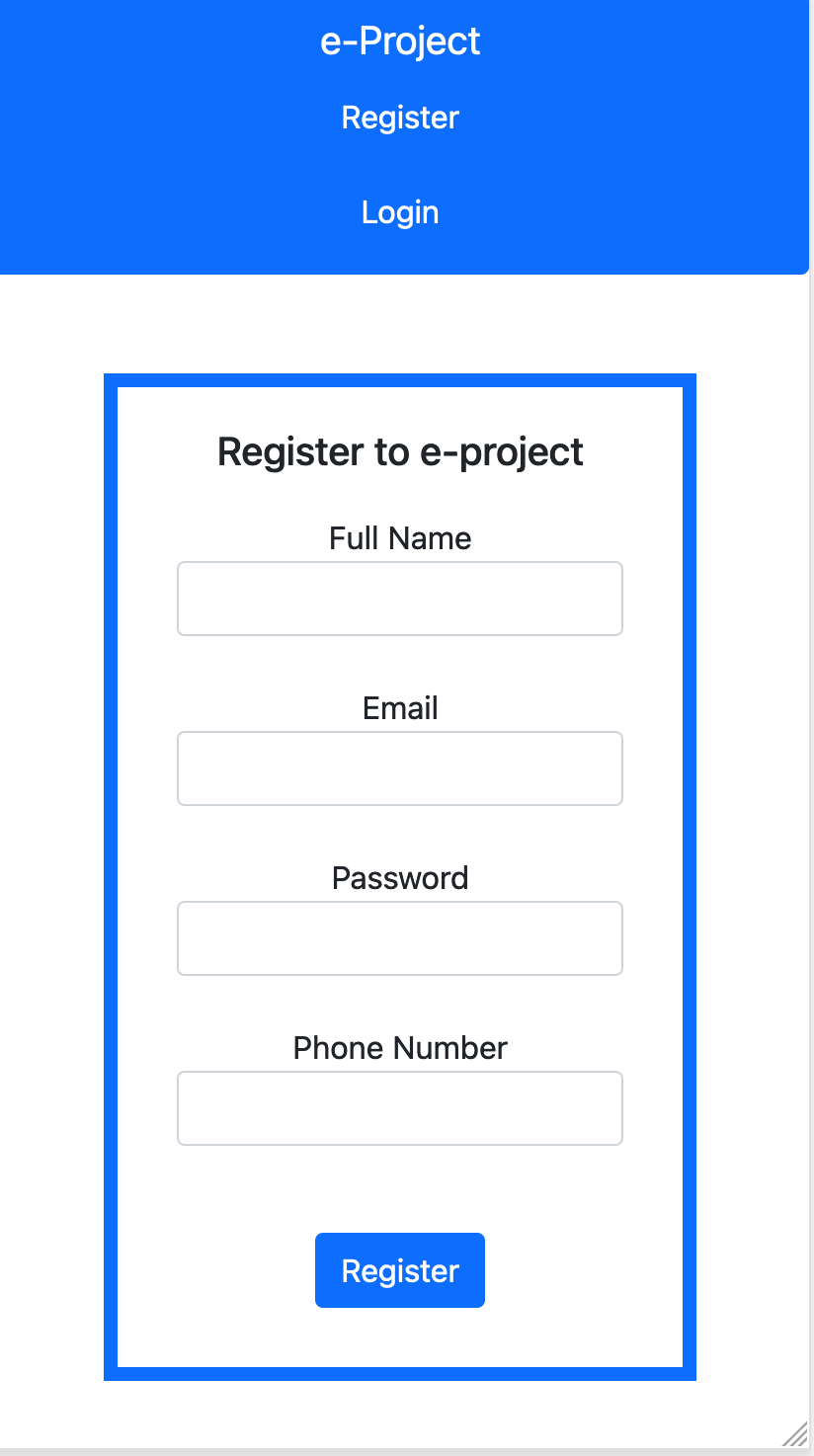
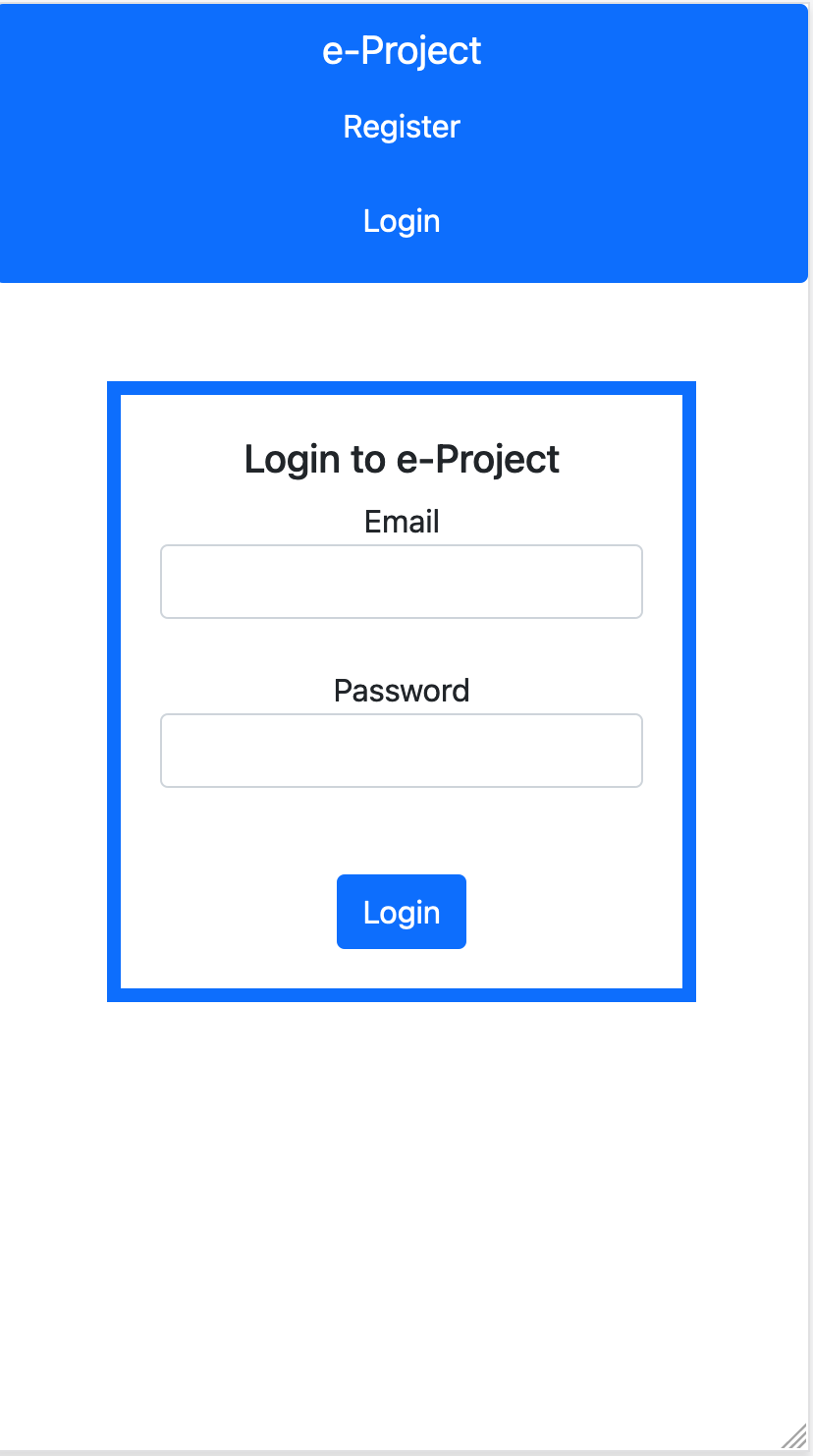


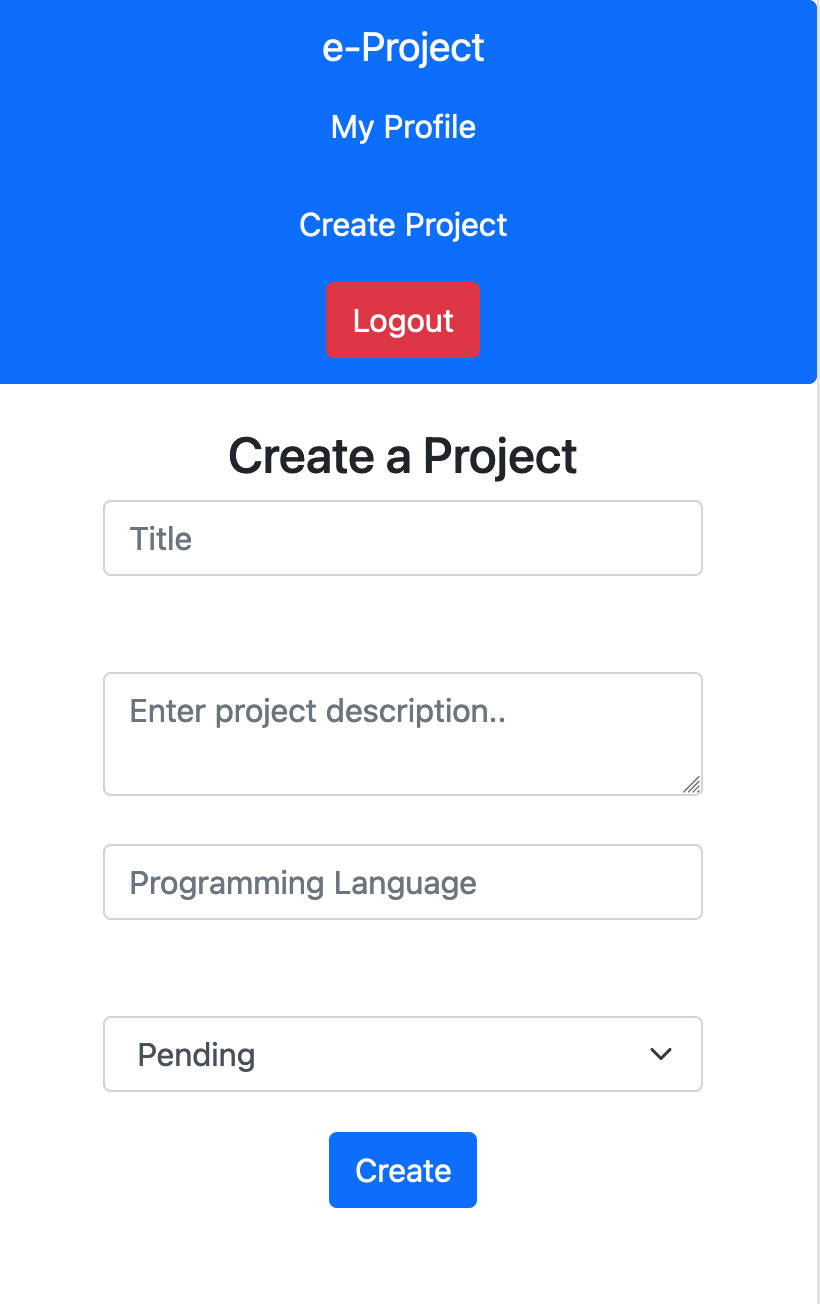
We also added some responsiveness in terms of design; when a user uses a phone or a small screen the following is shown (iPhone 6, 7, 8, 8 plus):



As can be seen, when opening the page in a small screen, we made the background change color, and the navbar to show the option vertically rather than horizontally.

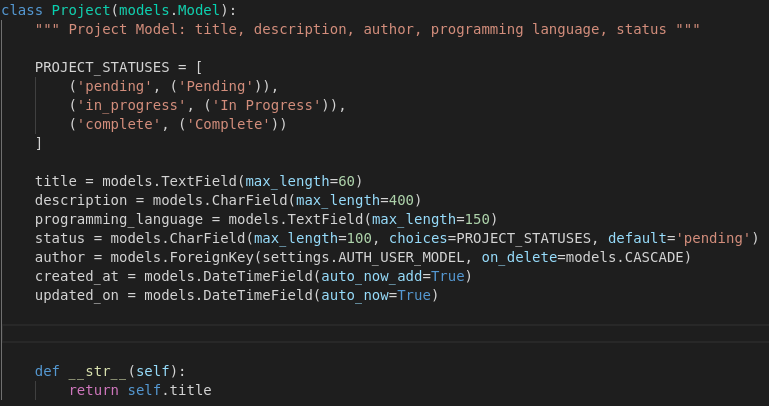
Also, all the forms are in a bootstrap style::





**2 Add Feature to Create Projects**

For this user story we developed a separate django app to handle the backend processing of user projects. Because the last two user stories consisted in managing projects, we added the CRUD operations and a model class to represent a user project.

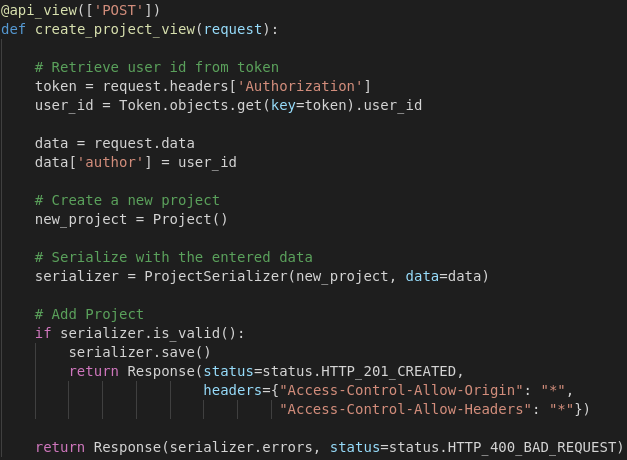


Besides the title, description, author, programming language, and project status, we created two additional fields that store the time when the project was created and the time when the project was most-recently updated. These fields are “*created\_at*” and “*updated\_on*”. Because all of this data might not be unique for a particular project, we do not assign a primary key. The Django REST API automatically creates an “*id*” field which auto increments and is unique among all the projects.

Users and projects have a many-to-one relationship. This means that every project has one author and one author may have many projects. If a user is deleted from the database, the user’s projects must be deleted (redundant data). For this reason, at the author field, we supply an additional parameter called “on\_delete” which determines the action to be taken when the author is deleted from the database. The “*CASCADE*” constant in the models class tells the database to delete every project associated with the author.

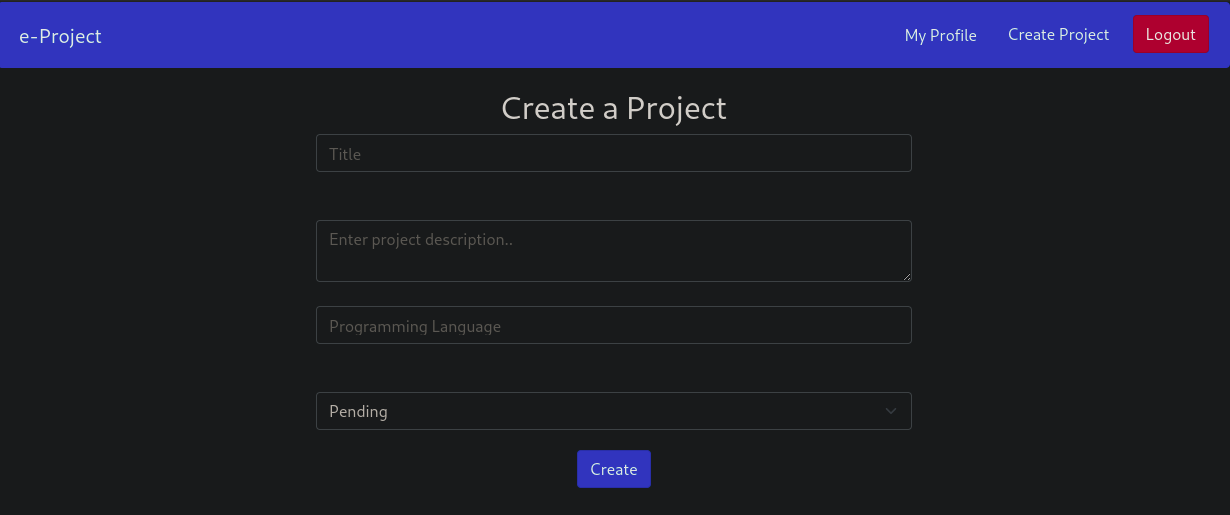
Moreover, the status can either be “*pending*”, “*in progress*”, or “*complete*”. The project model class contains a list of three constants for each of these statuses. By default, the project status is pending.

The creation of a project in the database is handled by the “*create\_project\_view*” function.

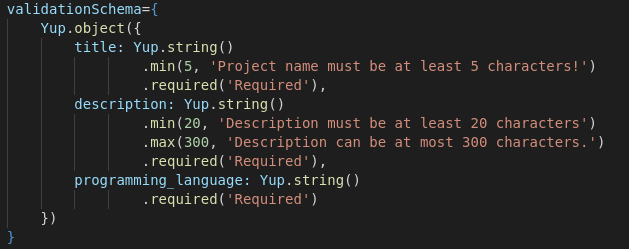


The first line containing “api\_view([‘POST’])” is a python decorator in django which determines what kind of request will this method answer to. Since we are creating an object, the method request should be POST. The function takes as argument the data entered by the user. From that data, we derive the token, in order to get the user id. We need the user id to associate a user with the current project being created. This token is not only used to associate the project with the user but also to prove that the user is authenticated. An anonymous user cannot create a project. Next, we use the serializer methods to check the validity of the entered data and if everything is in compliance, the project is added to the database and a status with code 201 is returned to the user. The end point for this action is “domain/projects/create”.

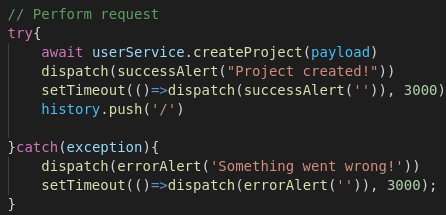
The front end side of this user story consists of a React component. This component contains the form with the appropriate data fields.



All of the fields are validated according to the requirements. The validation is done through Yup, a javascript library. The title’s minimum size is 5 characters. The description’s minimum size is 20 and maximum size is 300 characters.



The following code executes when we submit the form:



The website tries to send the payload (entered input), displays a success message, and redirects the user to the homepage. Otherwise, we see an error message.

The following method sends the post requests to the server side:

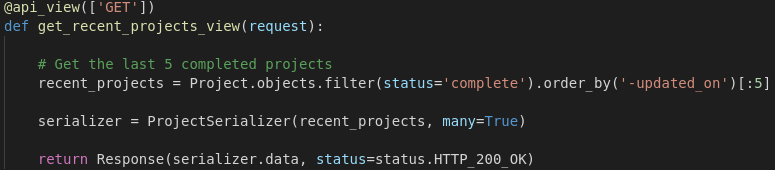


Since this is a network request, the method is executed asynchronous. We retrieve the token stored in the local storage and add it to the header’s request. To send the request, we use axios library.

**3 List The 5 recently Completed**

This user story is fairly straightforward. Everytime the user lands on the homepage, the client (browser) performs a GET request to the server to retrieve the data. The data is then displayed accordingly.

The following code represents the method that handles the request on the server-side. Again, we determine the request method using the “*api\_view*” python decorator. The server then retrieves the projects and filters them by saving only the 5 recent completed ones. The function oder\_by returns an array. We use array slicing to retrieve the last 5 results. This data is then serialized into JSON and sent to the client together with an HTTP 200 OK status.



The list of projects displayed on the screen is wrapped under a React component in the frontend side of the application. This component is shown regardless of whether the user is authenticated or not. The get request to retrieve the data is done again through axios GET request.

**Individual Contribution:**

* Lazaron Shyta: Bootstrap implementation, project retrieval from the database.
* Gabriel Kuka: CRUD operations on projects, project form and data validation.