Senior Project Submission

ProjectManager Web Application Report

Author:
Biplab Thapa Magar

Instructor:
Professor Victor Miller

Introduction

This paper documents ProjectManager Web Application, along with its design, implementation, and a guide on how to use it. The application is divided into two parts: the server side and the client side. The server side handles the business logic, user accounts logic, models, and the storage of data, while the client side, which is rendered on a web browser, is where users interact with the application. Reflecting this split into two parts, the ProjectManager web application primarily uses two frameworks for its implementation, one framework for each part of the project. The two frameworks are React on the client side and ASP.NET Core on the server side.

About ProjectManager

The ProjectManager application is a web application for managing the workflow and delegation when working on any type of project. It is particularly made for people who want to work on a project in teams and communicate effectively with each other regarding project tasks and team management.

In the application, each user starts by creating a password-protected user account. This ensures that other users cannot access data they are not supposed. User accounts store all the information regarding a user, including his name and bio. Once a user is logged in to his account, he can access all his projects and tasks and start working on them.

Every user can create new projects to work on. These projects can be of any type, depending on what they are working on. When a user creates a project, he/she can invite other users to the project so that they can work on the project together.

Since each project has many users, and since each member must not be able to make changes to the project, each project user is assigned one of two roles specific to the project. A user can either be a project Administrator, or a Member. The Administrator has the ability to make changes to the project, manage the users in the project, and delete the project. The members only have the ability to view project info, create-edit tasks, and comment on tasks. A user's role in one project does not affect his role in another project. So, a user can have different roles within different projects.

A Task in a project is one of the primary units of the ProjectManager application. A task represents something that needs to be done when working on a project. A task can be assigned to the users who are a part of the project team. When tasks are assigned to users, they appear in the Tasks section of the user's page. Tasks have several attributes, including a name, a description, a task type, an urgency level, and a status, which serve to define the task. The task type attribute is used for categorizing tasks. ProjectManager allows projects to have their own set of task types, allowing for more flexibility. It is up to project administrators to define the task types used for the project. Each task can also be commented on, so that users can communicate with each other, giving feedback and collaborating, when working on a task.

To help team management and coordination, the ProjectManager web application keeps track of all the users' activities in the projects. A user's comments, task assignments, task updates, and so on are displayed in the projects page so the users of the project can see what work is being done by whom

Technologies Used

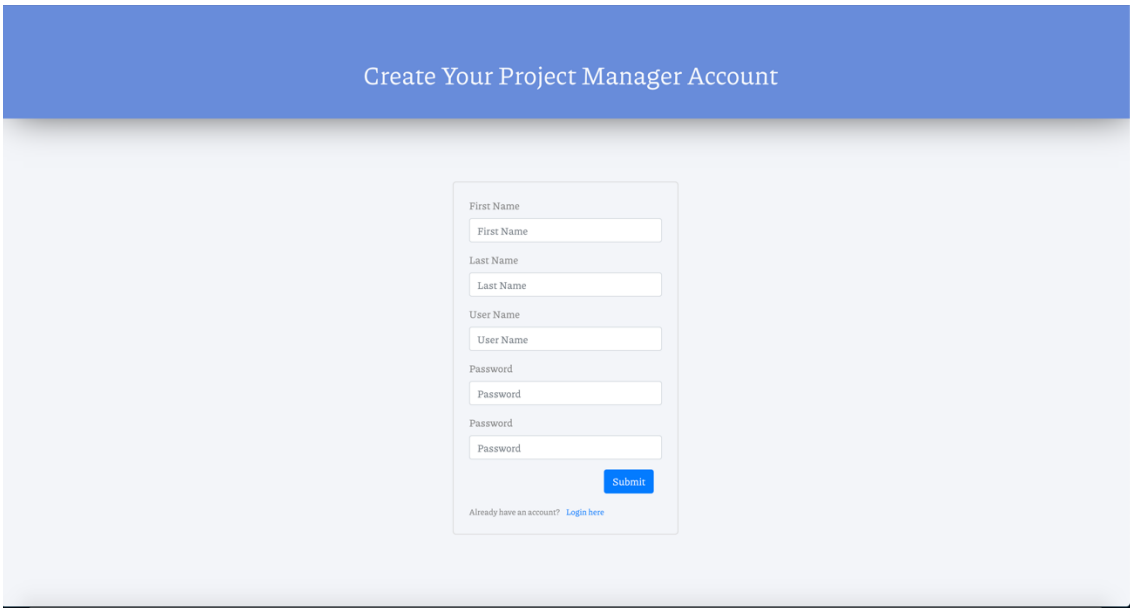
As mentioned earlier, the ProjectManager application primarily uses two frameworks. ReactJs is used on the front-end. It is a framework for building UI components that show up in the user's browser and delineate the user's interaction with the application. ReactJs is based on the JavaScript programming language.

On the backend, the application server is built using the ASP.NET Core framework, which is a framework based on the C# programming language. The backend provides a Web API, which is an interface that exposes a series of endpoints to the client side. The client side can use these endpoints to send and receive data back and forth from the server. The ASP.NET Core provides the framework for implementing such Web APIs. On top of ASP.NET Core, the backend also makes use of Entity Framework Core, which is an extension of ASP.NET Core and is used to create, manage, and access database. It greatly streamlines the process of setting up databases, defining the relationships between various tables, and updating database schemes when needed. Finally, the backend also makes use of the Identity system, which is used to manage user accounts. Identity is used to create user accounts and sign users in.

Manual for use

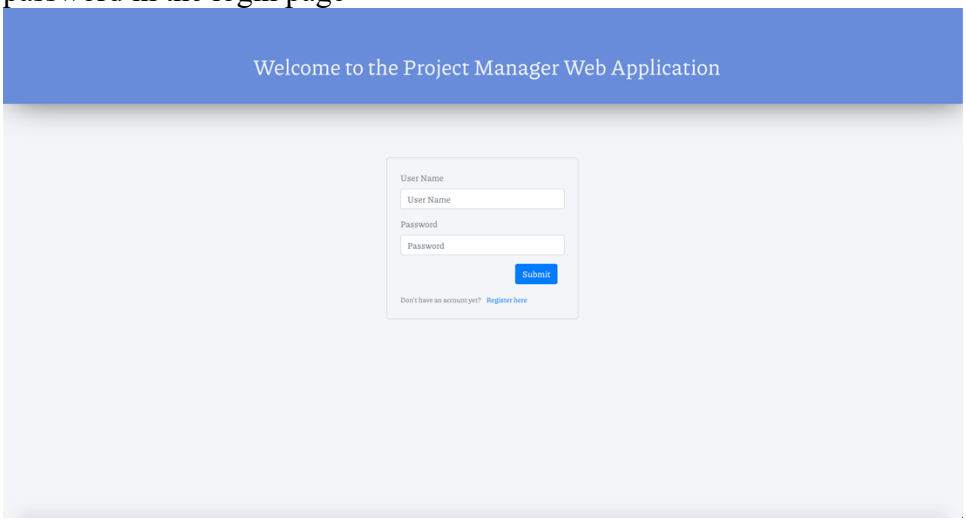
The ProjectManager application is a web application. Therefore, its layout is structured like a standard web application on the internet would. The following lists out all the features for the web application.

When a person first enters the ProjectManager web application, they are prompted to log in. If they do not an account already, they can register an account.

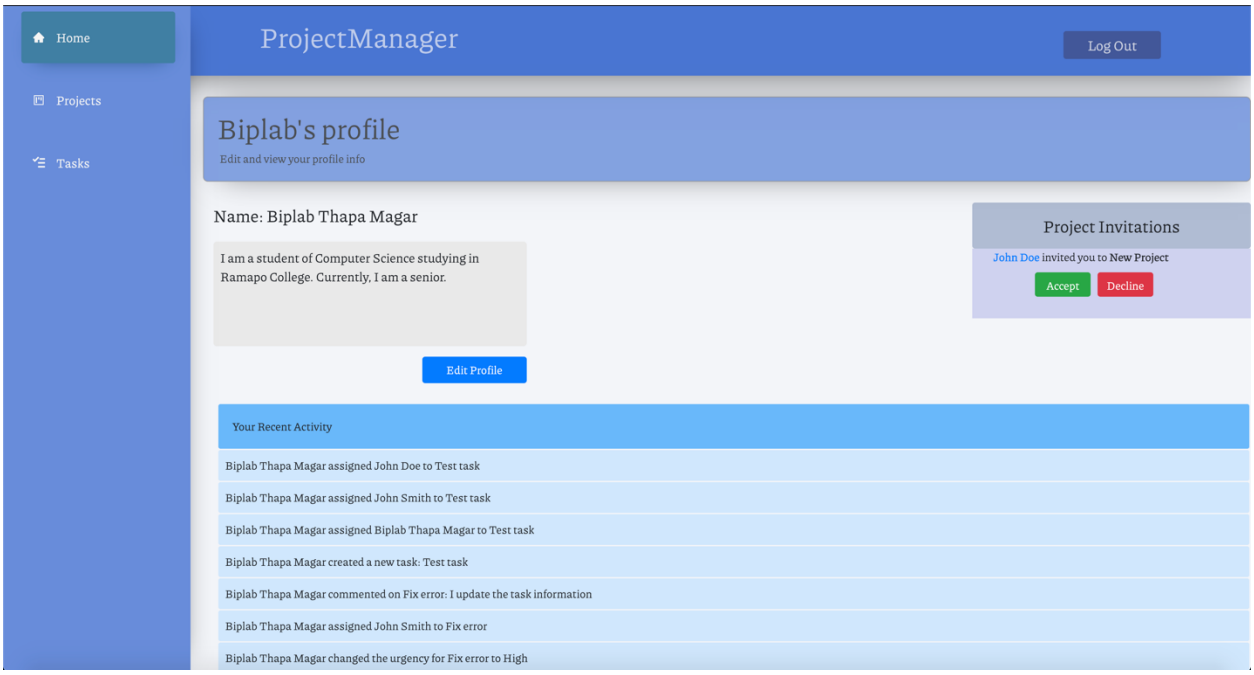


Users can enter their information in the registration page. They must make sure to pick a unique username, which they will use to log in during subsequent usage of the application.

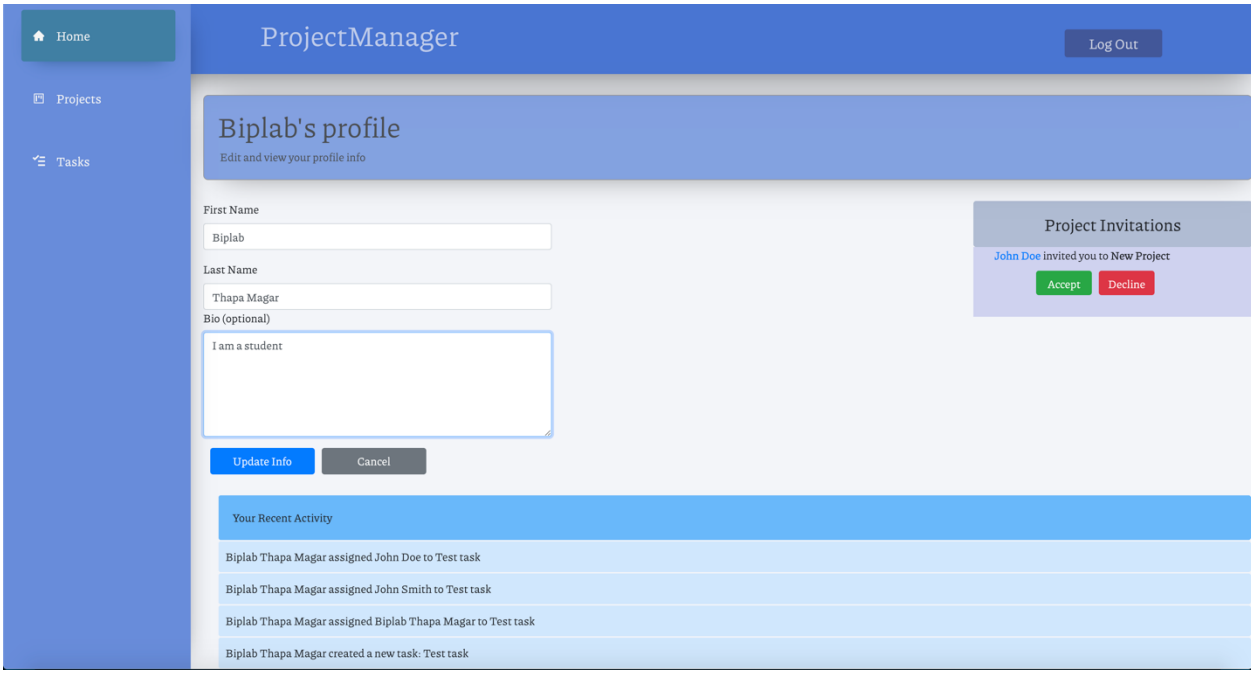
If the user already has an account registered, they can login by entering their username and password in the login page



Once a user has signed in or registered, they are brough to the home page, which displays their profile, their recent activities, and any pending invitations to project they have received.

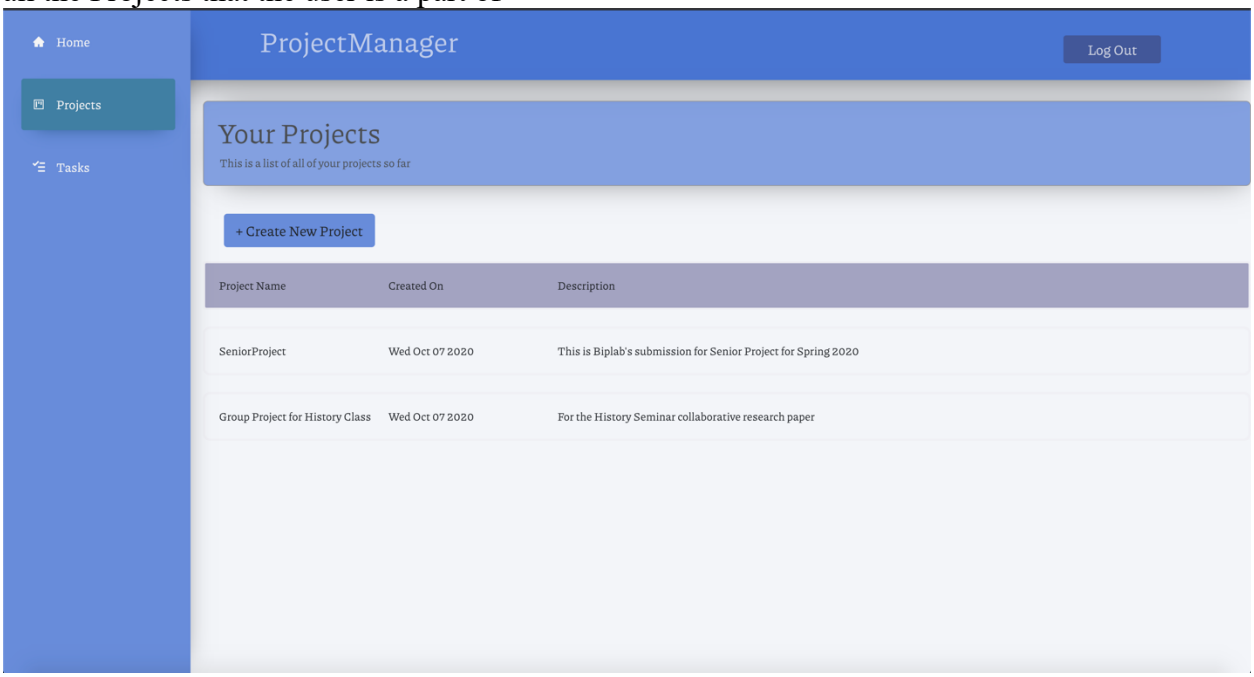


The user can update his user information by clicking on the Edit Profile button.

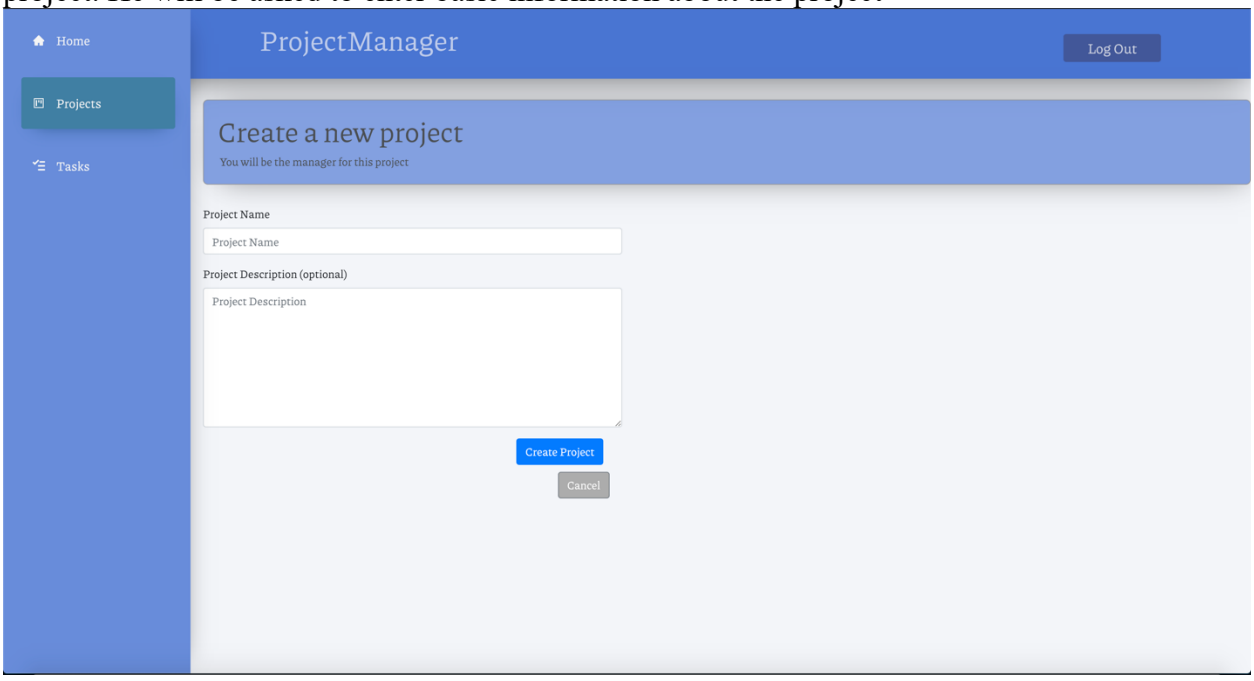


After adding his information, he can click on the Update Info button to save changes

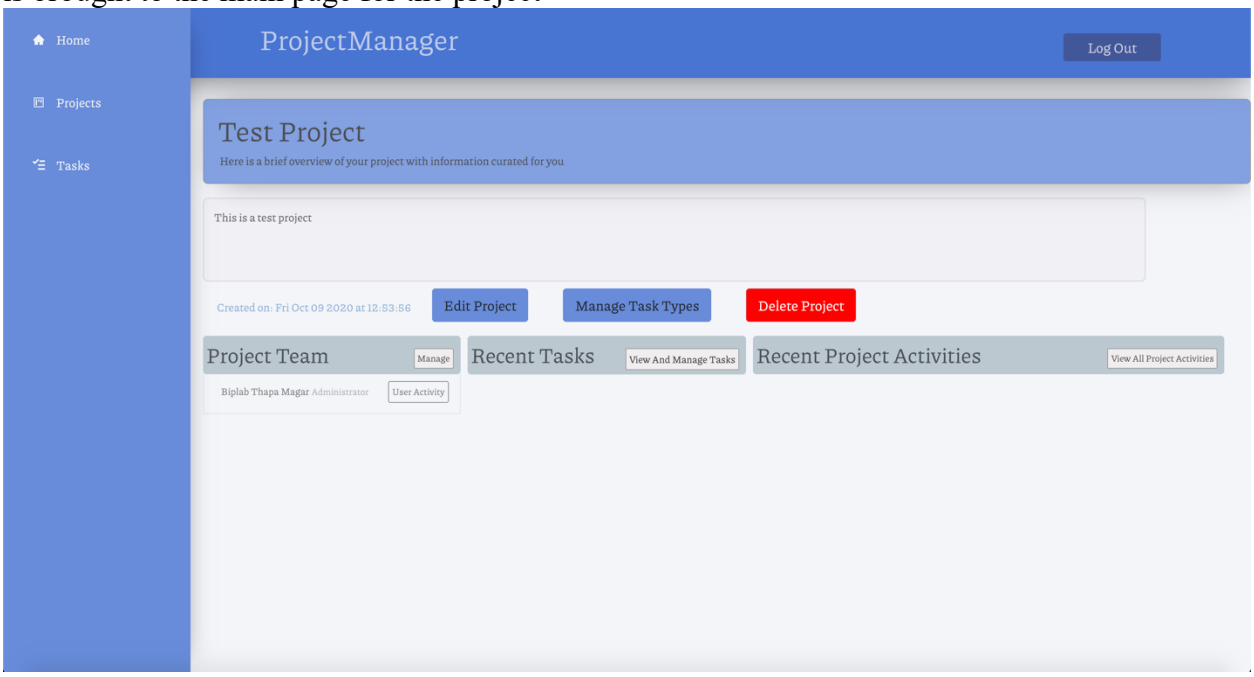
To get started with a project, the user goes to the Projects section. The Projects section lists out all the Projects that the user is a part of



On the Projects page, the user can click on the + Create New Project button to create a new project. He will be asked to enter basic information about the project

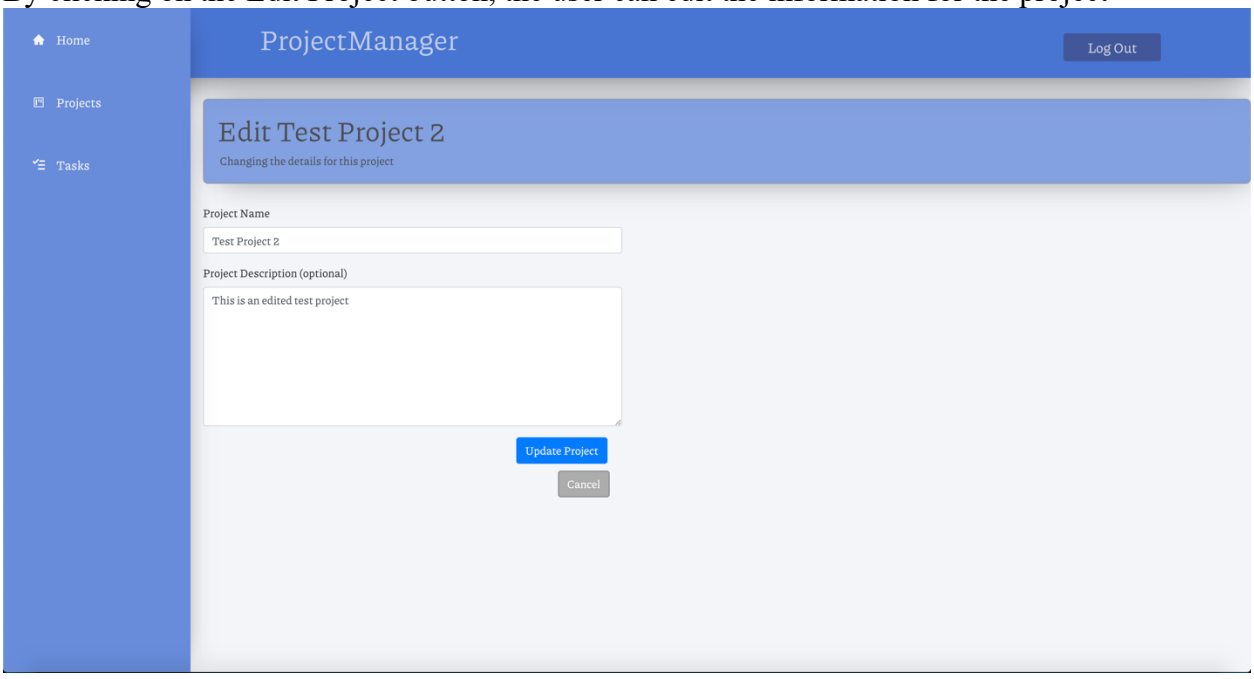


After filling out basic information for the project, the user clicks on the Create Project button and is brought to the main page for the project



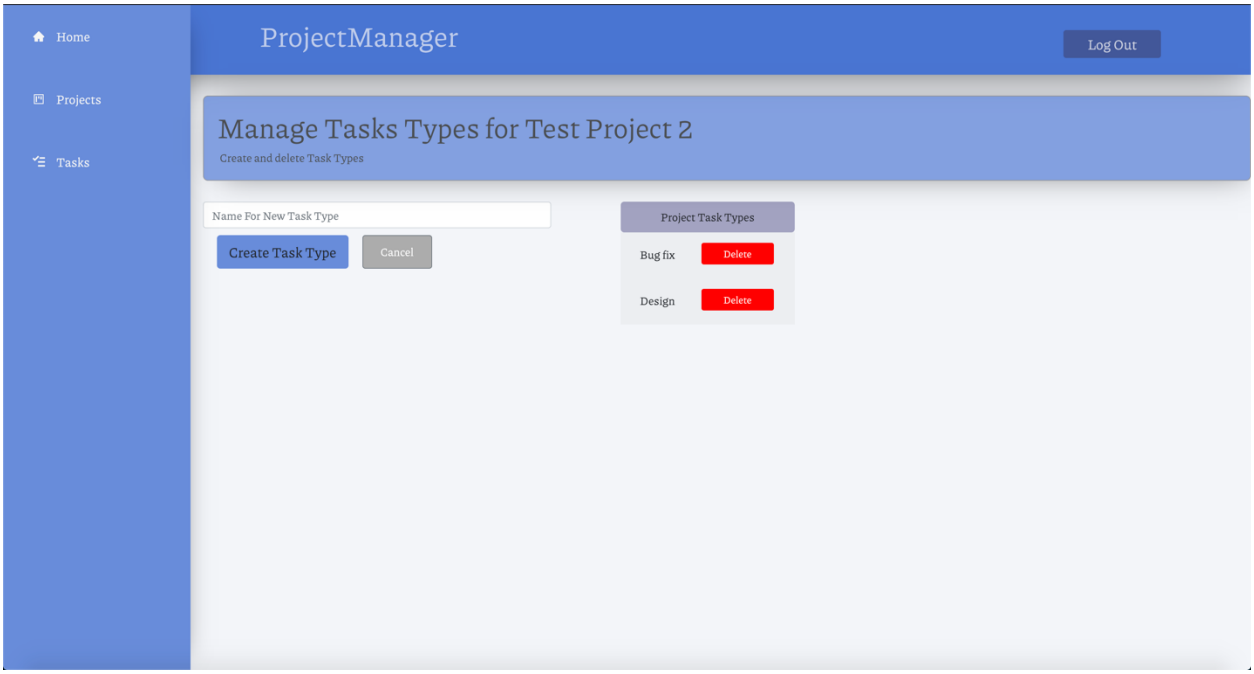
This page lists out all the basic information for the project. Here, users of the project can view their project team members, all the recent tasks in the project, as well as the recent activities by all the users in the project.

By clicking on the Edit Project button, the user can edit the information for the project

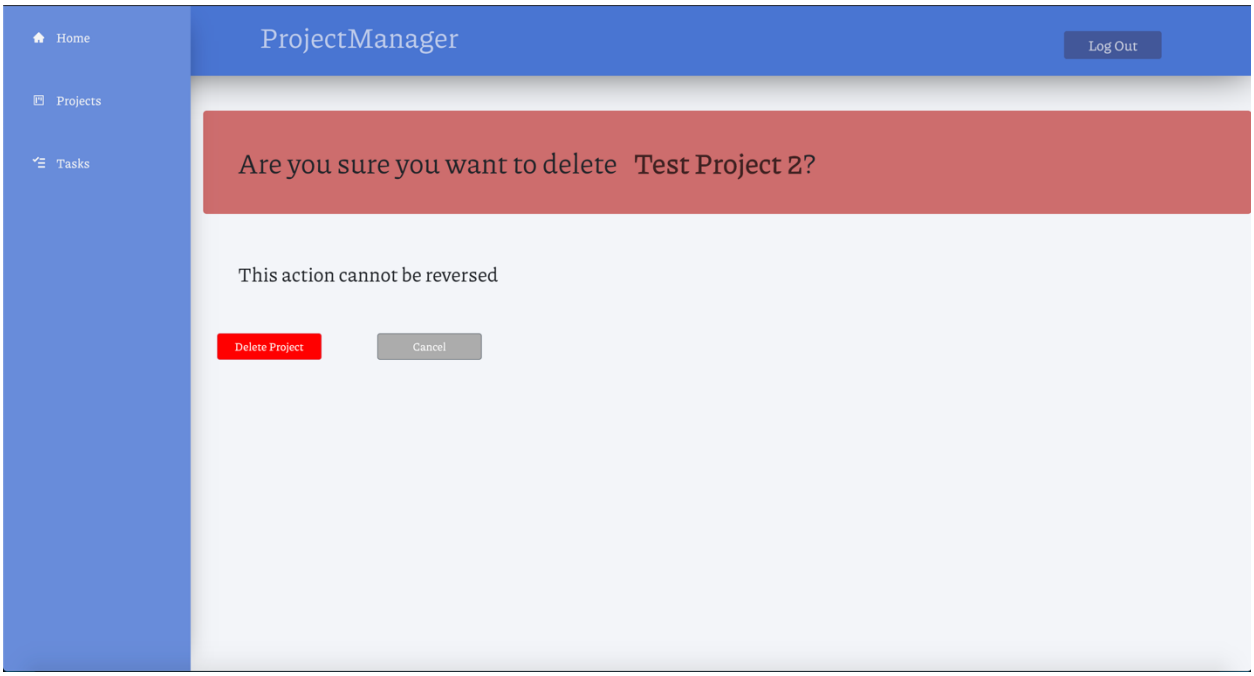


Clicking on the Update Project button brings the user back to the main project page

The user can click on the Manage Task Types button to bring up the section where he can define the task types for the project.
Here, the user created two task types for the project. He can add more task types or remove unneeded task types

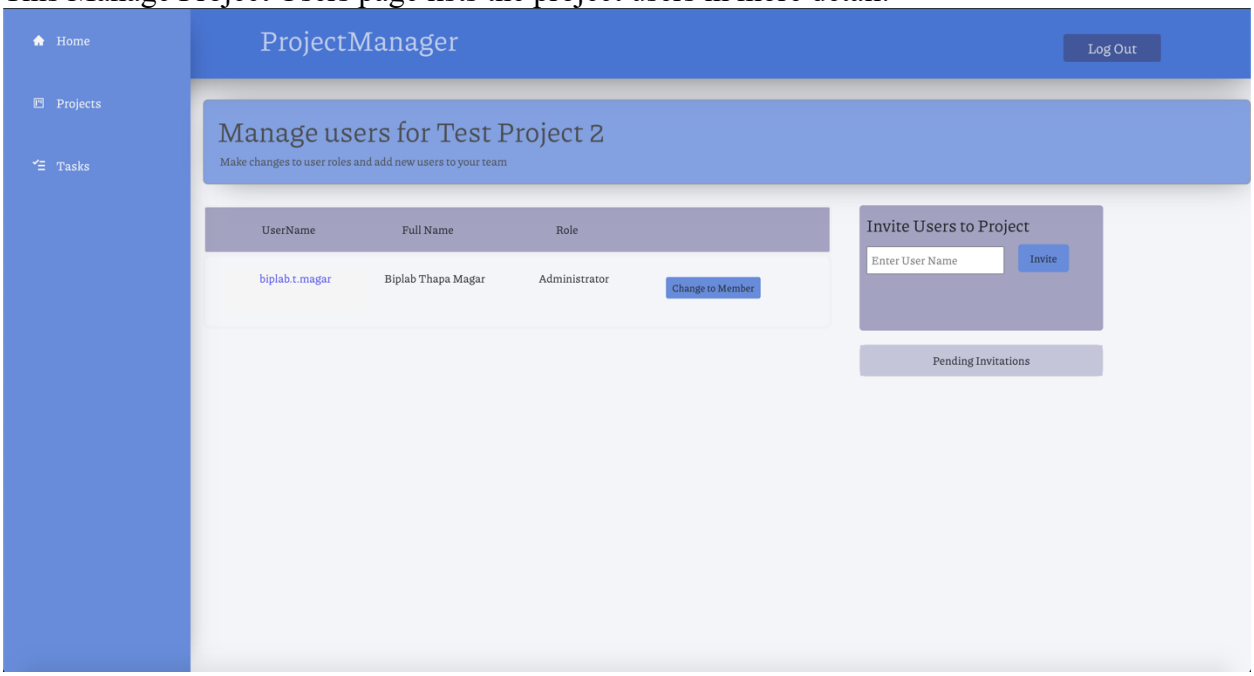


The user can click on the Delete Project button on the main projects page if he wants to delete the project.

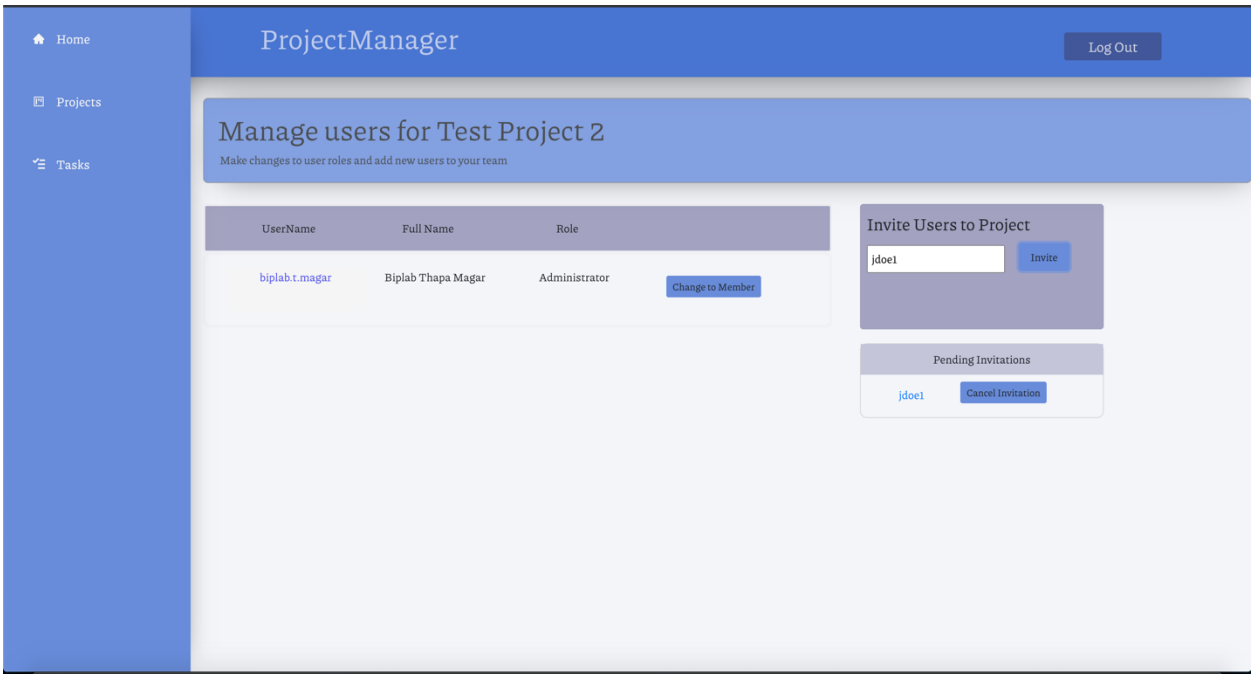


The Project Team subsection in the main project page lists out all the users that are part of the project. By clicking on the Manage button, the user can manage the users in the project

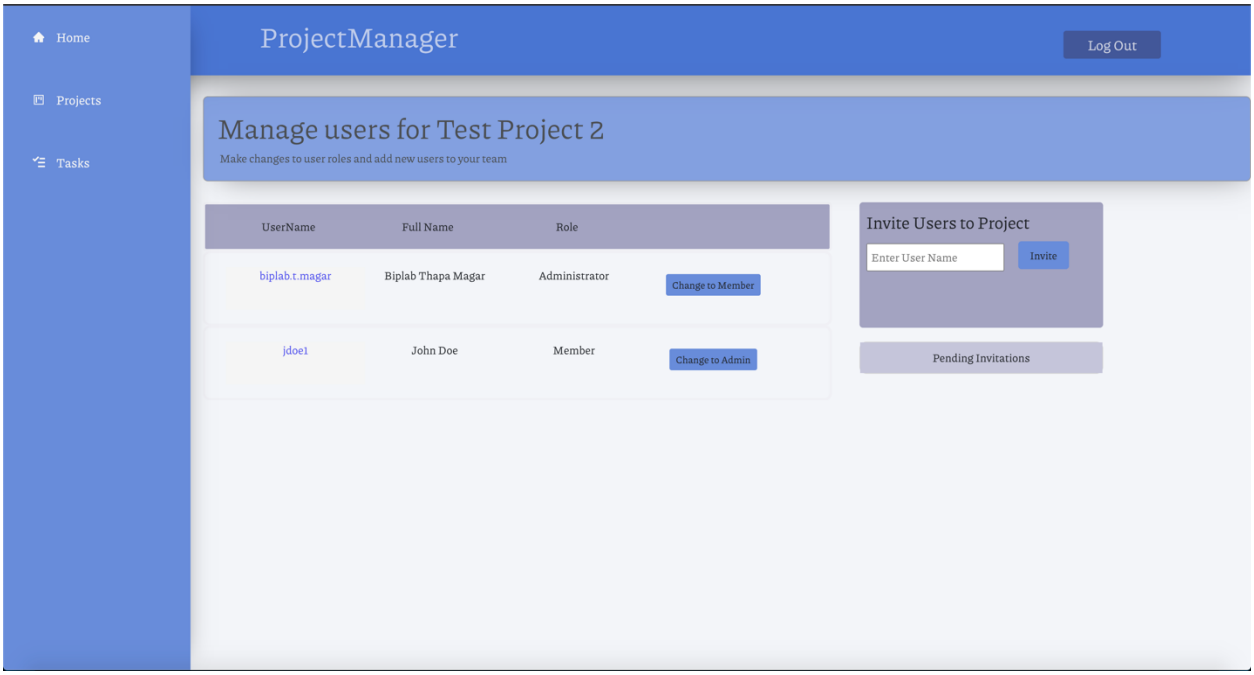
This Manage Project Users page lists the project users in more detail.



Here the user can invite other users to the project by typing out their username



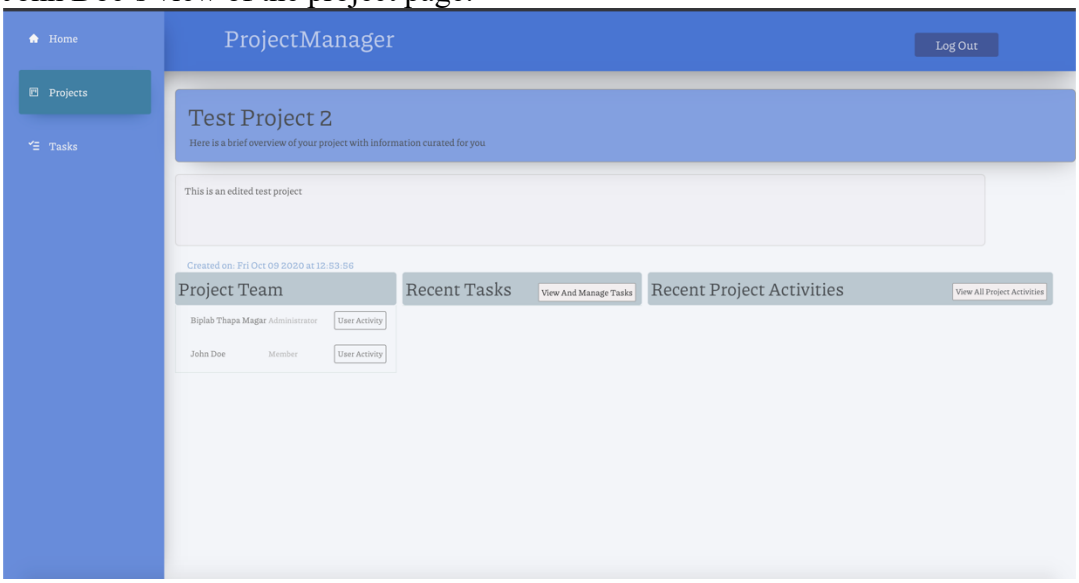
Once the user, jdoe1 in this case, accepts the invitation, he will be included in the project users list



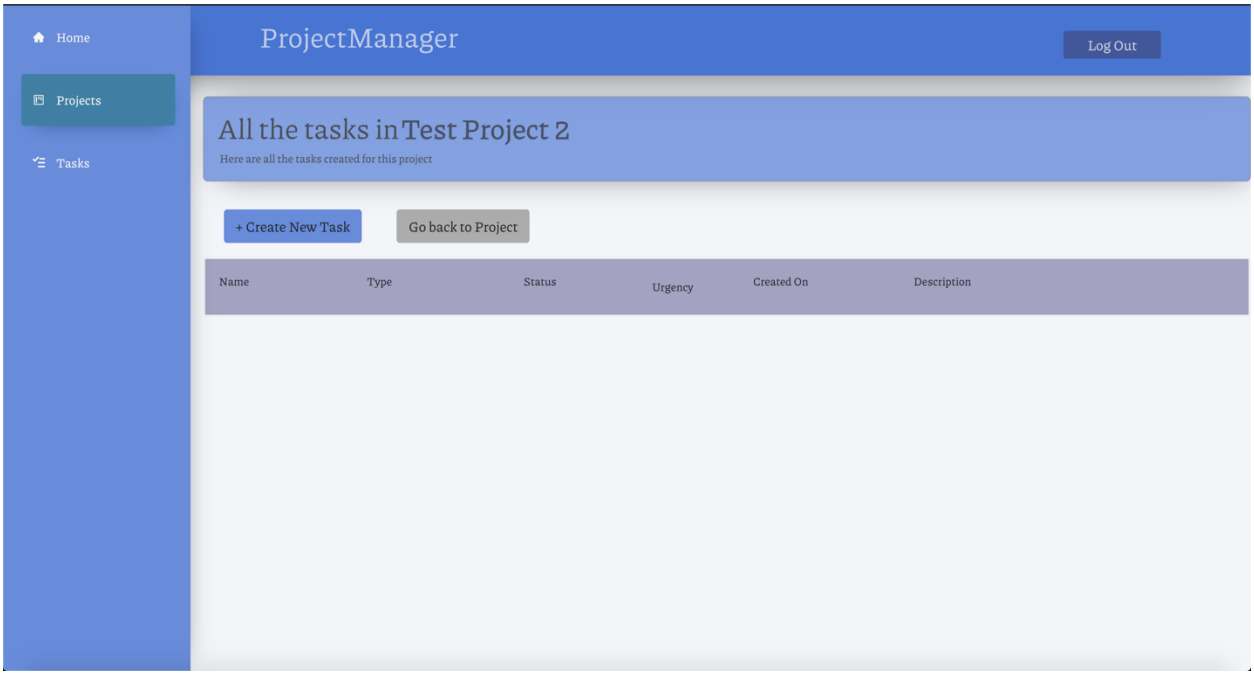
Here, John Doe, who has accepted the invitation is automatically assigned the role of Member. By clicking on the Change to Admin button, he will be assigned as an administrator for the project too.

From John Doe's view, who is still a member and not an administrator in the project, the main projects page does not include any of the buttons that an administrator can use to make changes to the project and to assign users to the project

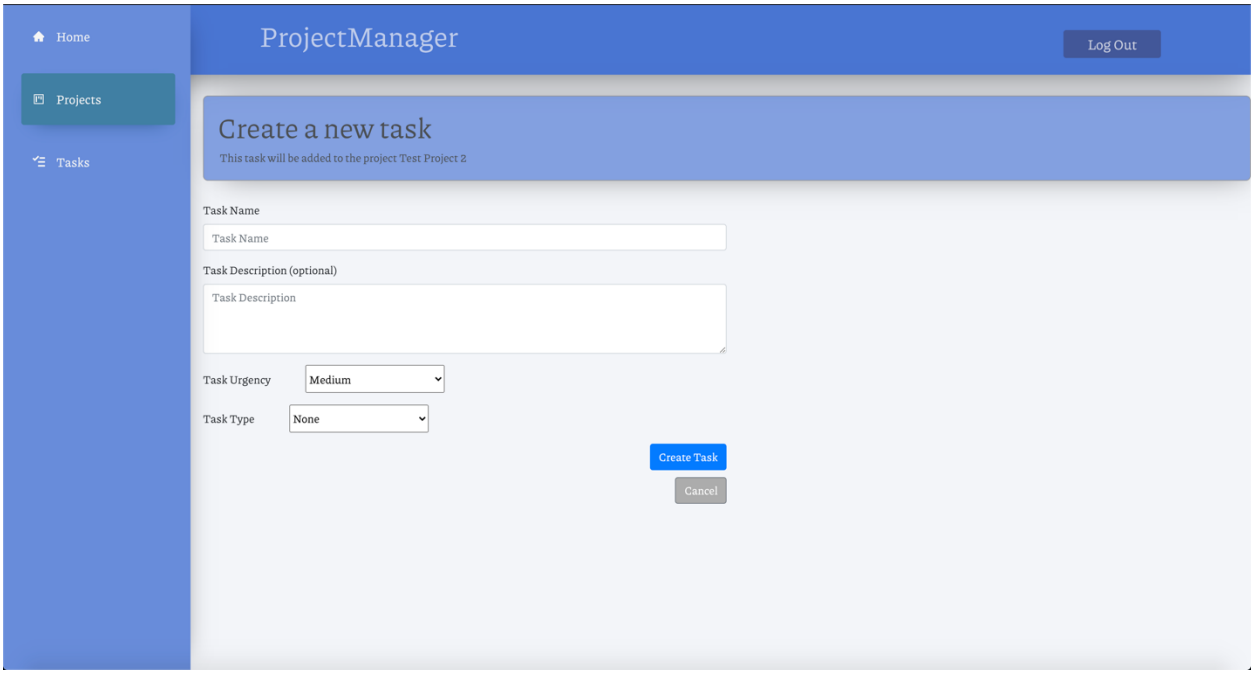
John Doe's view of the project page:



The Recent Tasks subsection in the main project page displays all the recent tasks added to the project, with the newest task at the top. Since there are no tasks in the project right now, no task is displayed. The user can create a task by clicking on the View and Manage tasks button, which takes the user to the Project Tasks page



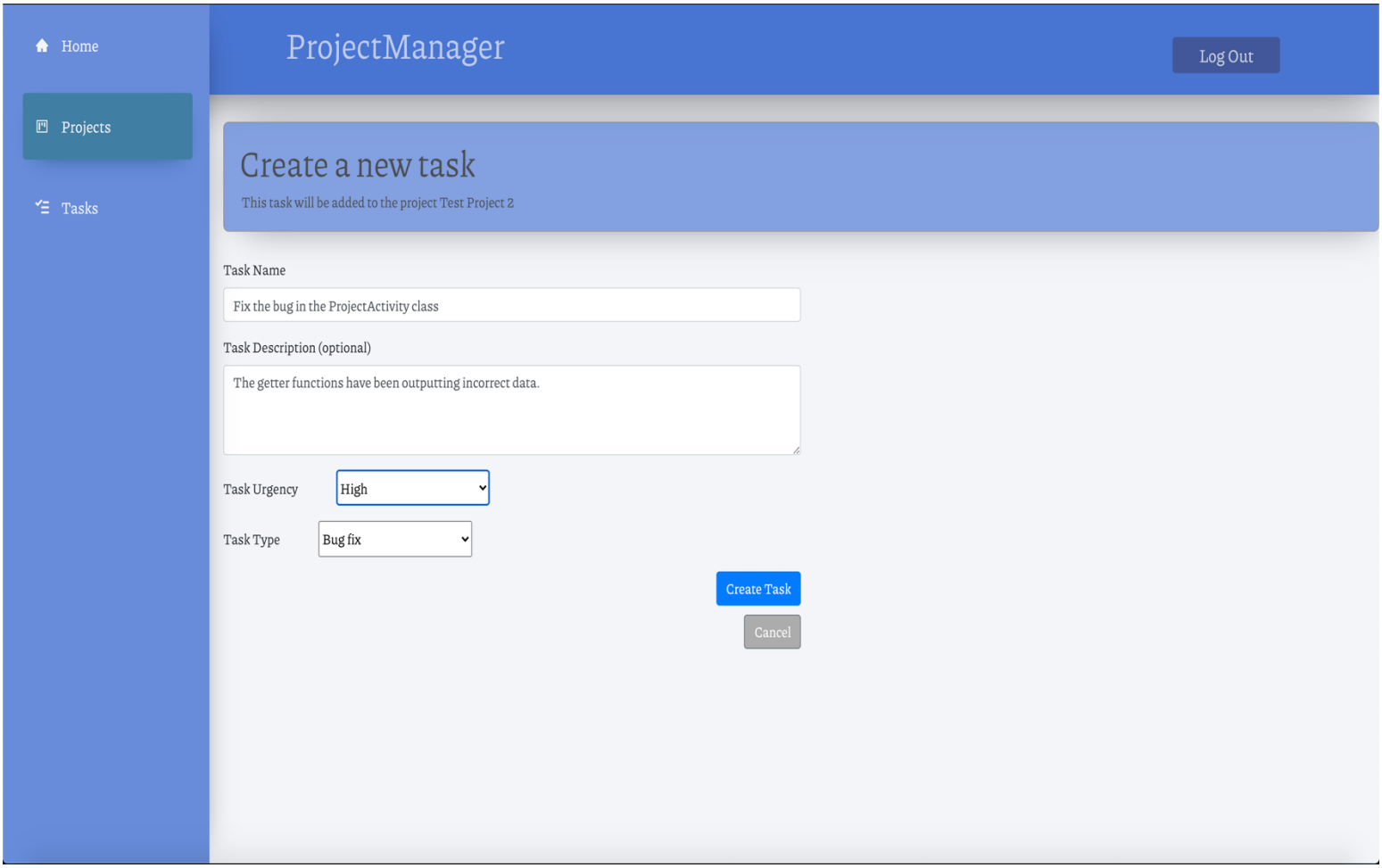
The user clicks on the + Create New Task button to create a new task



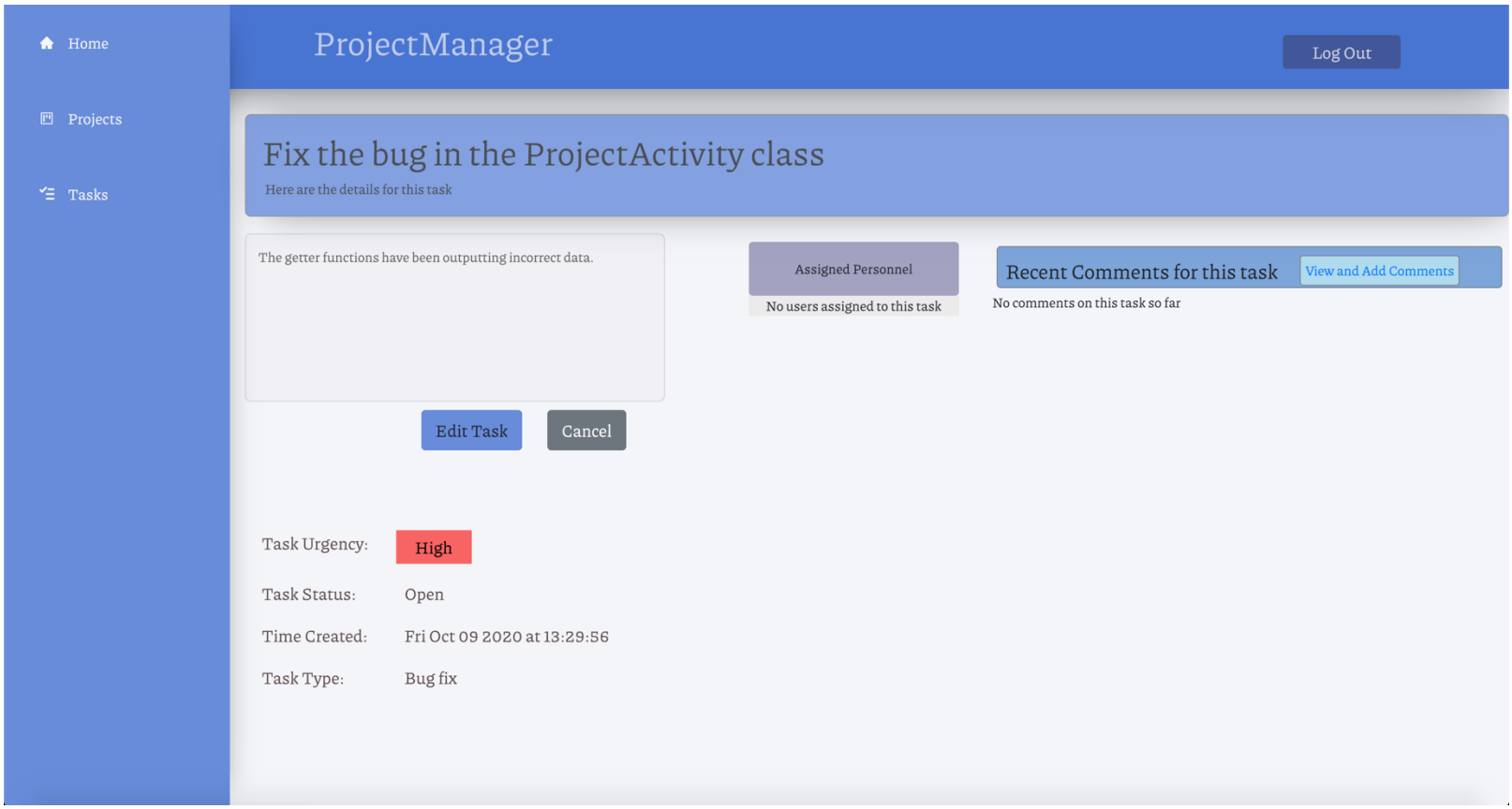
Here the user can fill out information regarding the tasks

Here, the Task Urgency is used to describe how urgent a task is. Task urgency is indicated by different colors, red for High, yellow for Medium, and green for Low.

The Task Type is the categorization of the task. The types available are defined by the user as shown previously



After finishing setting up the task, the user can click on the Create Task button. Once he does so, the task is created and the user is redirected to the page showing the details of the task

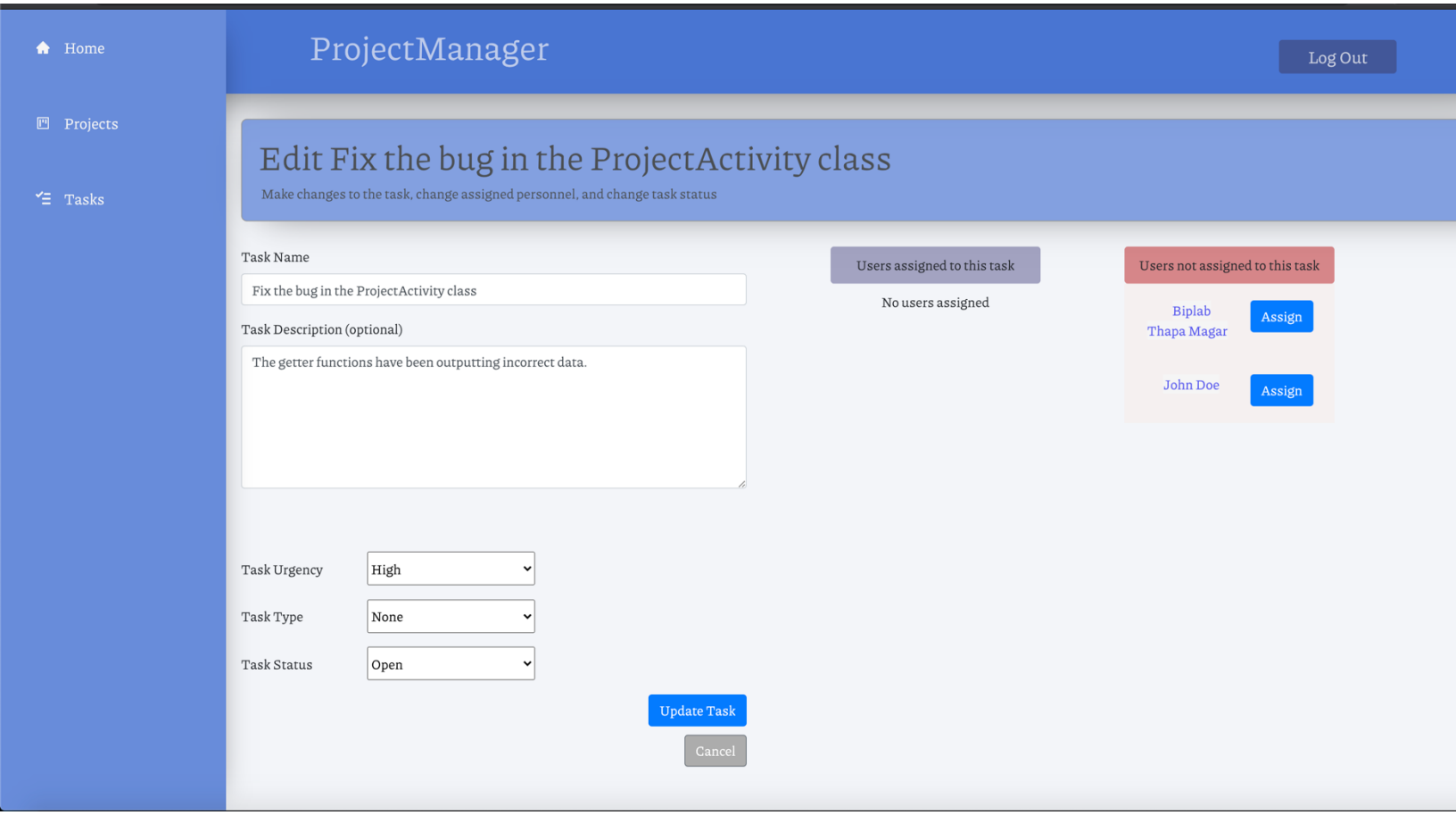


In this page, the details for the task are showed.

The page also shows the users who are currently assigned to the task. Currently, no users have been assigned to this task.

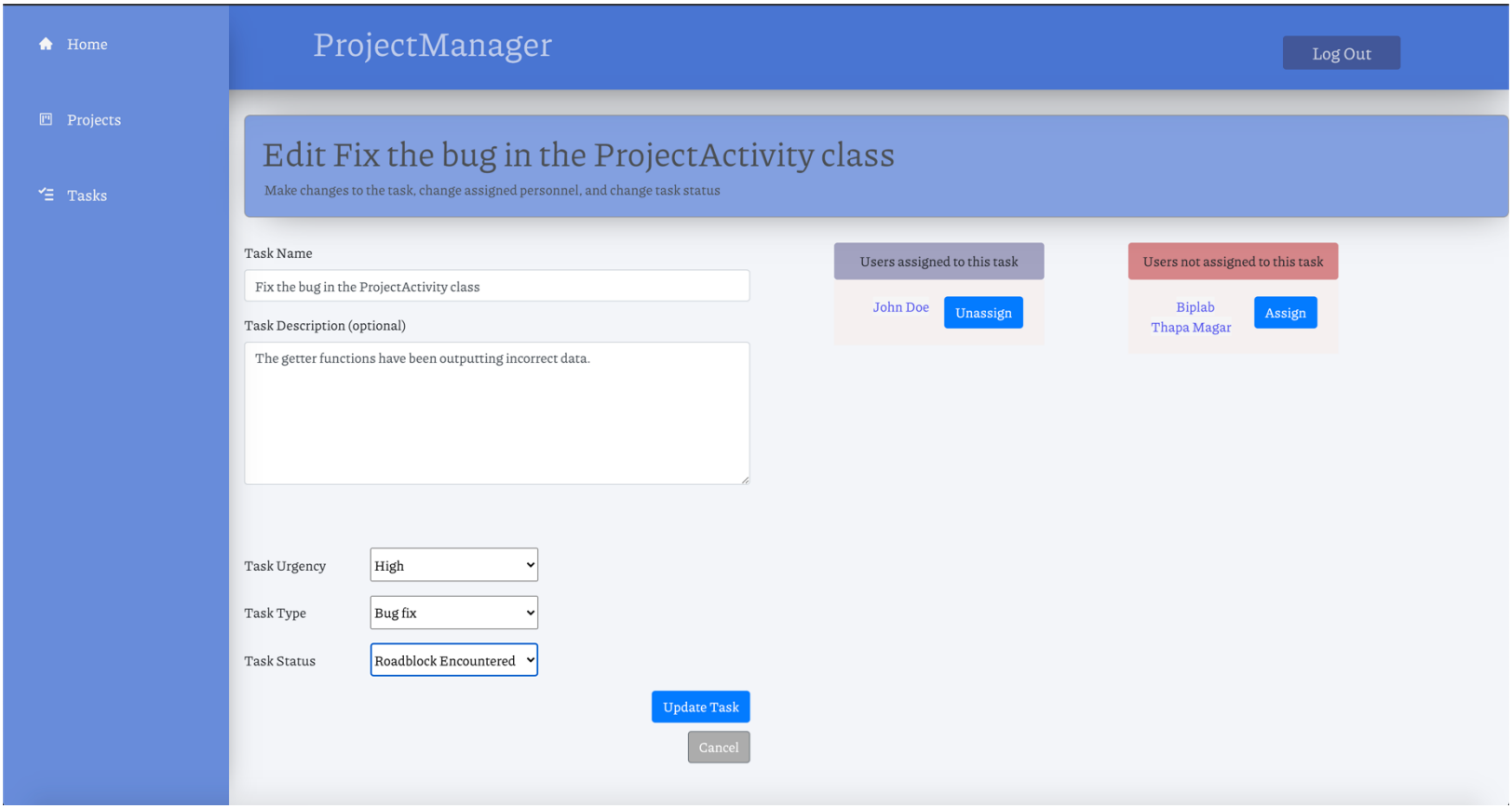
Here, it can be seen that the Task Status is set to Open. Whenever a task is created in a project, the task status is automatically set to Open. Users can change the status of a task too.

To edit the Task, the user clicks on the Edit Task button

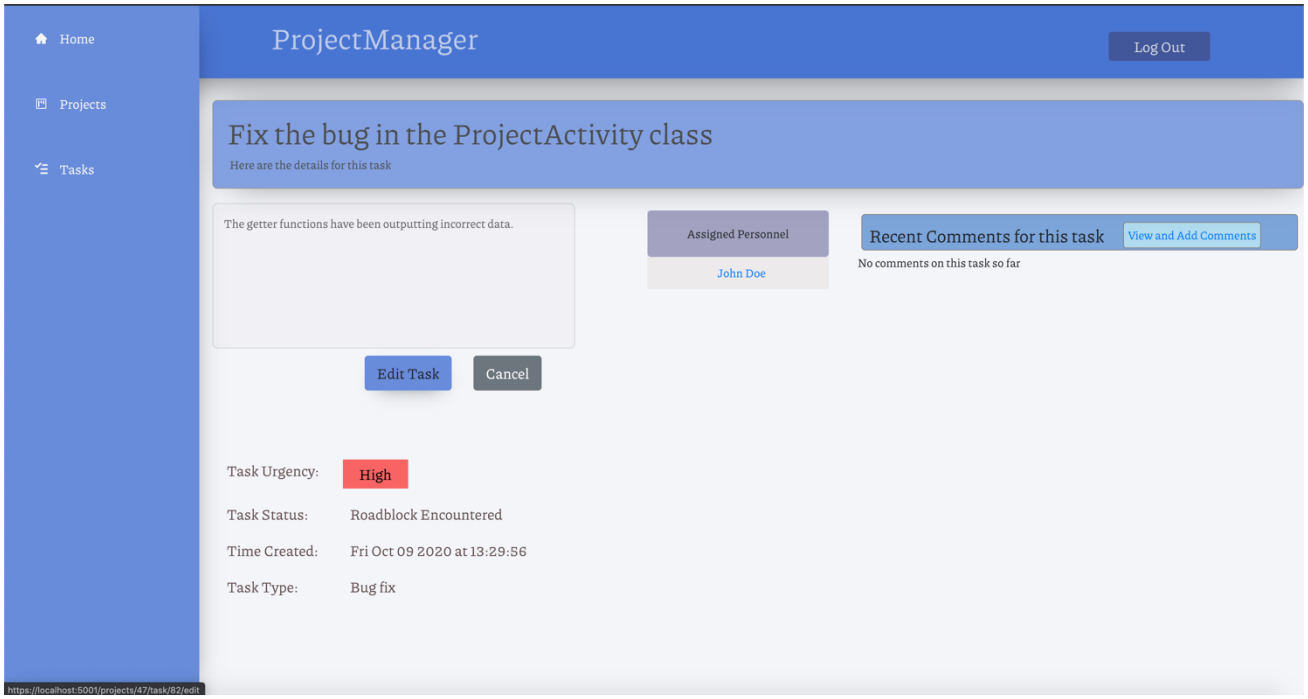


Here, the user can rename the task and adjust the description. He can also change the task urgency and task type. There is a new option now present that did not show up in the Create task page. The Task Status options describes the current status of a task. It can be set to Open, Suspended, Roadblock Encountered, Under Review, and Completed to describe the progress towards completion of the given task.

In the Edit Tasks page, users also get the option to Assign and Unassign users in the project to the task. Assigning a user to a task will make the task appear in the user's Task page. In the following, the user assigned John Doe to the task. The user can also unassign another user from the task if necessary

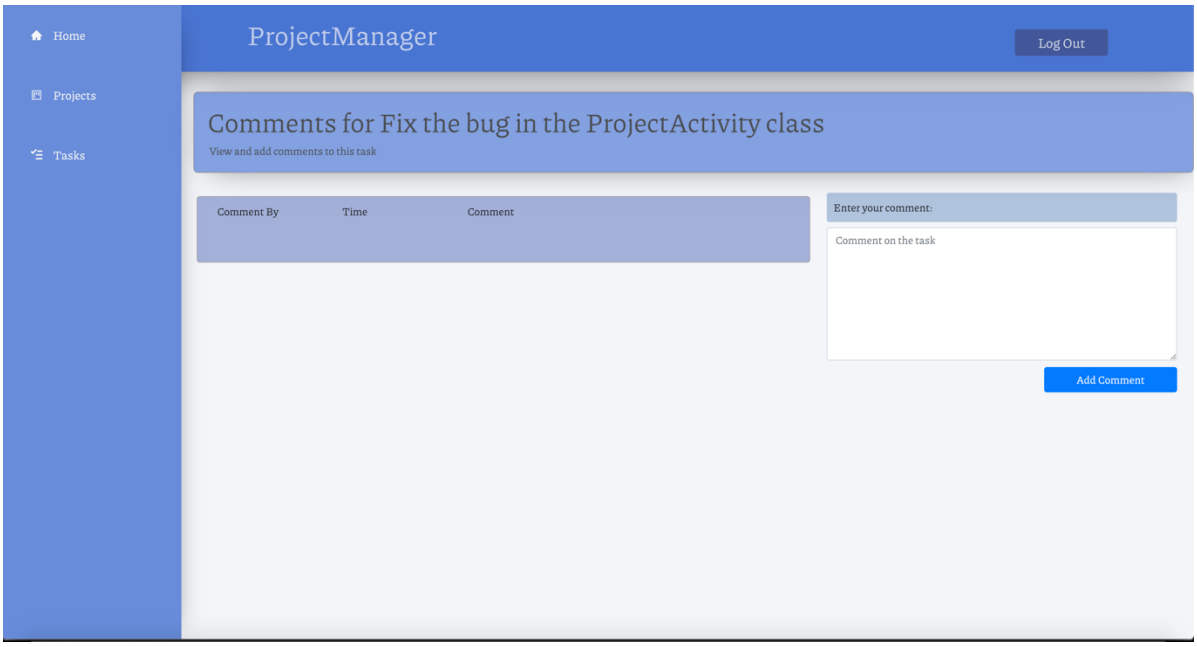


Clicking on the Update Task button will save the changes made to the task and redirect to the page showing task details

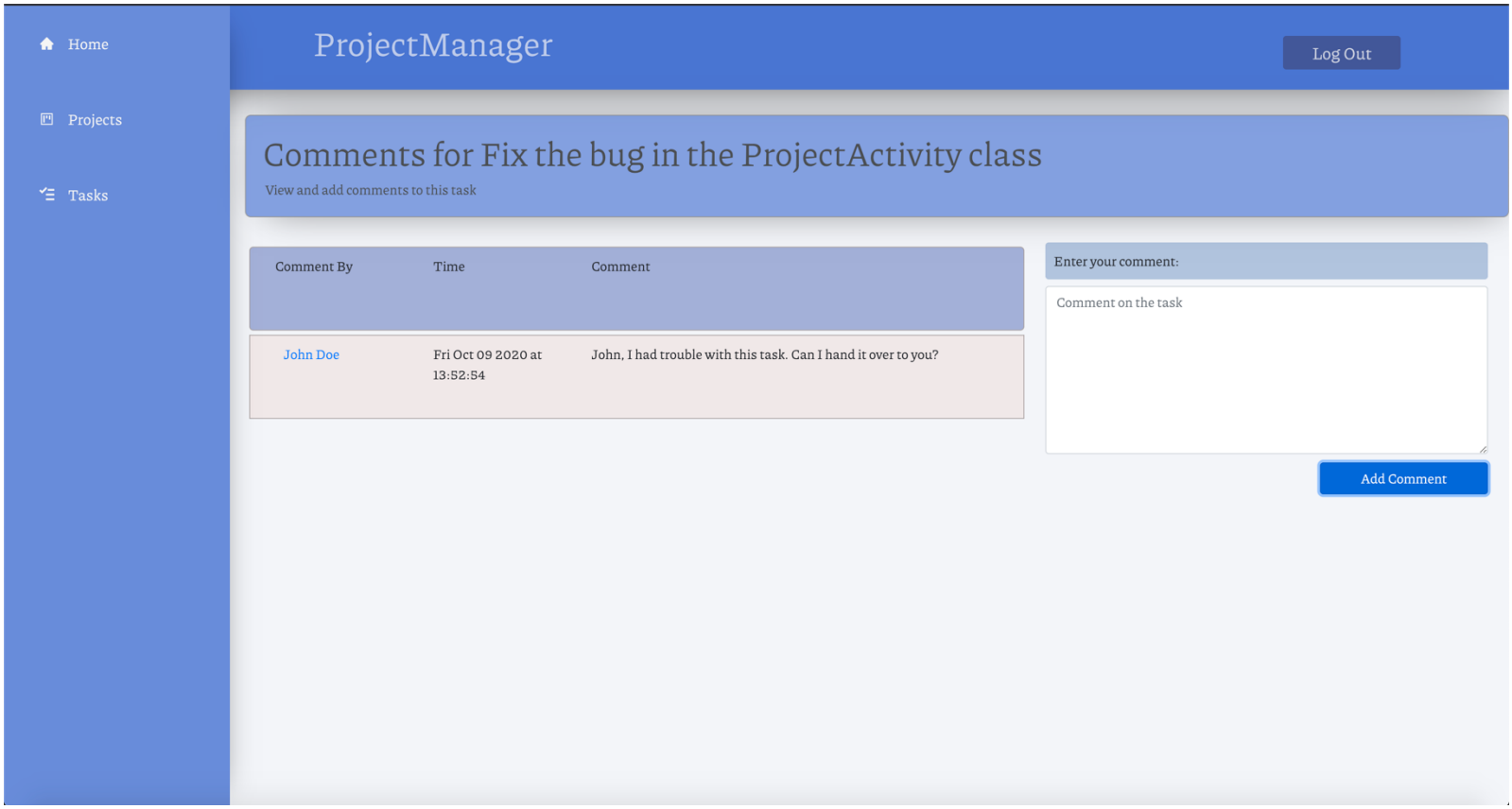


We can see that the task information has been updated and John Doe appears beneath the list of Assigned Personnel.

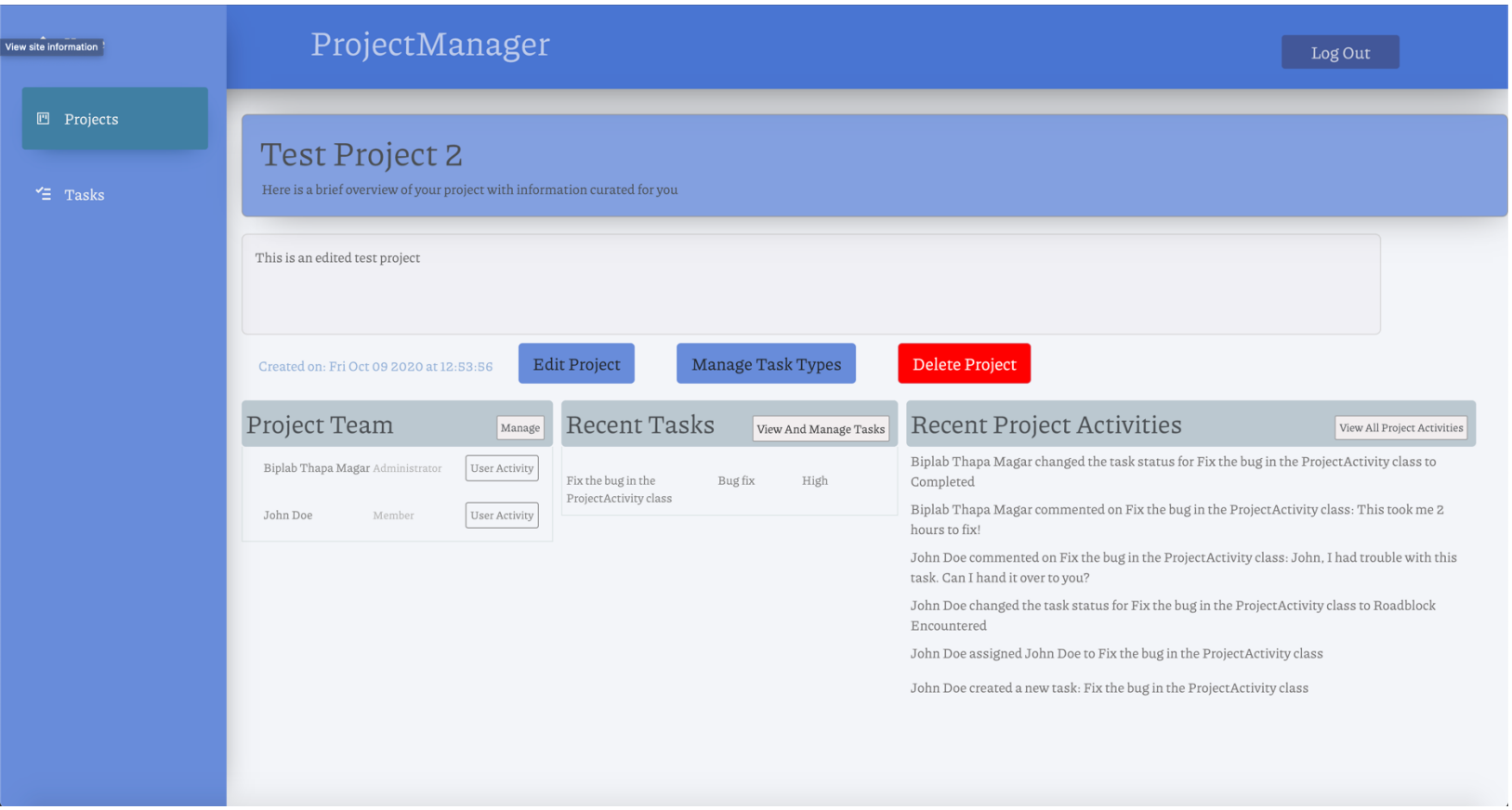
The final section in the task details page is the Recent Comments for this Task section. Here, the most recent comments made on the task would be displayed. Since there are no comments, the user can click on the View and Add Comments button to add comments to the task



The user types out his comment on the text box and puts in Add comment so that the comment appears under the task. The comment entry shows who made the comment and at what time



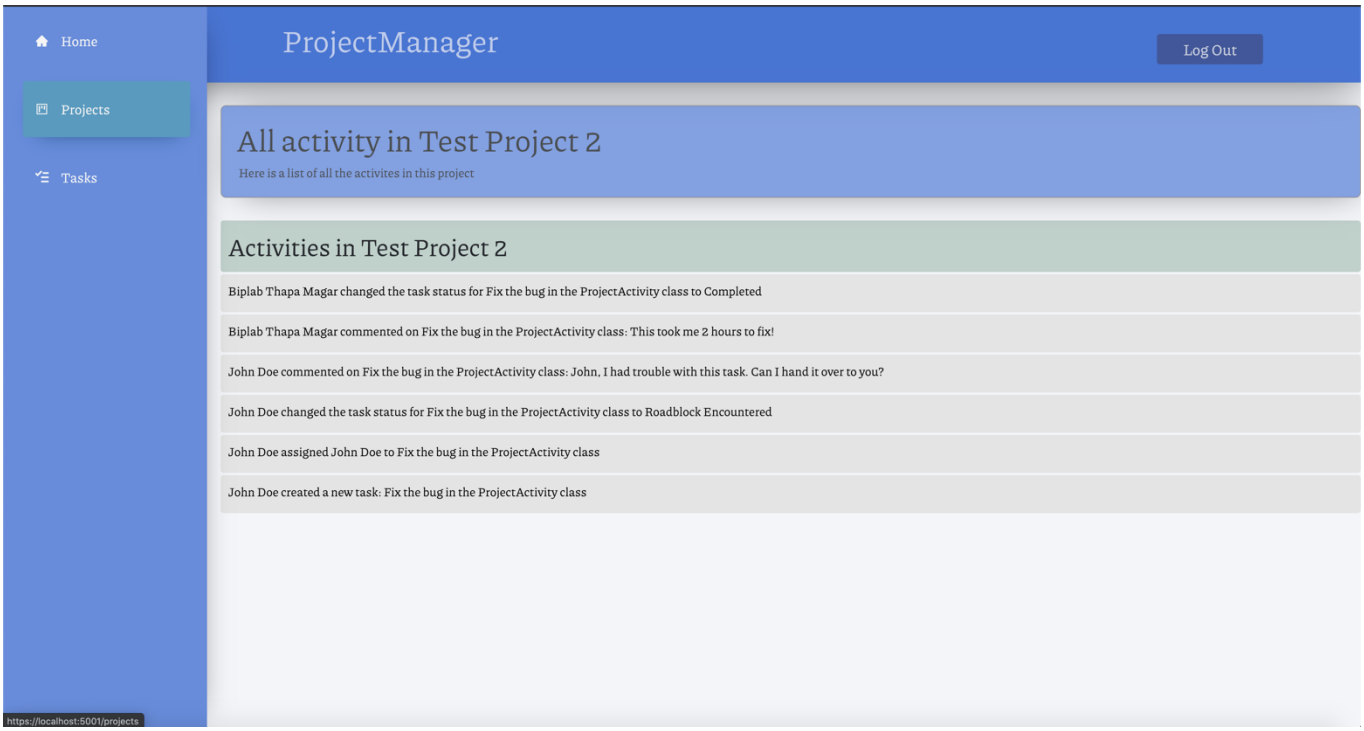
Going back to the main project page, we can see how the Recent Tasks subsection and the recent activity section now have entries under them



The Recent tasks subsection gives a brief overview of the recent tasks in the project.

The Recent Project Activities section lists out the recent activities made by the users of the project. We can see that John created the task and made changes to it. The users he assigned the task to are listed. The changes he made are also listed. The comments he posted are also present. The other user's activities are listed in the same way.

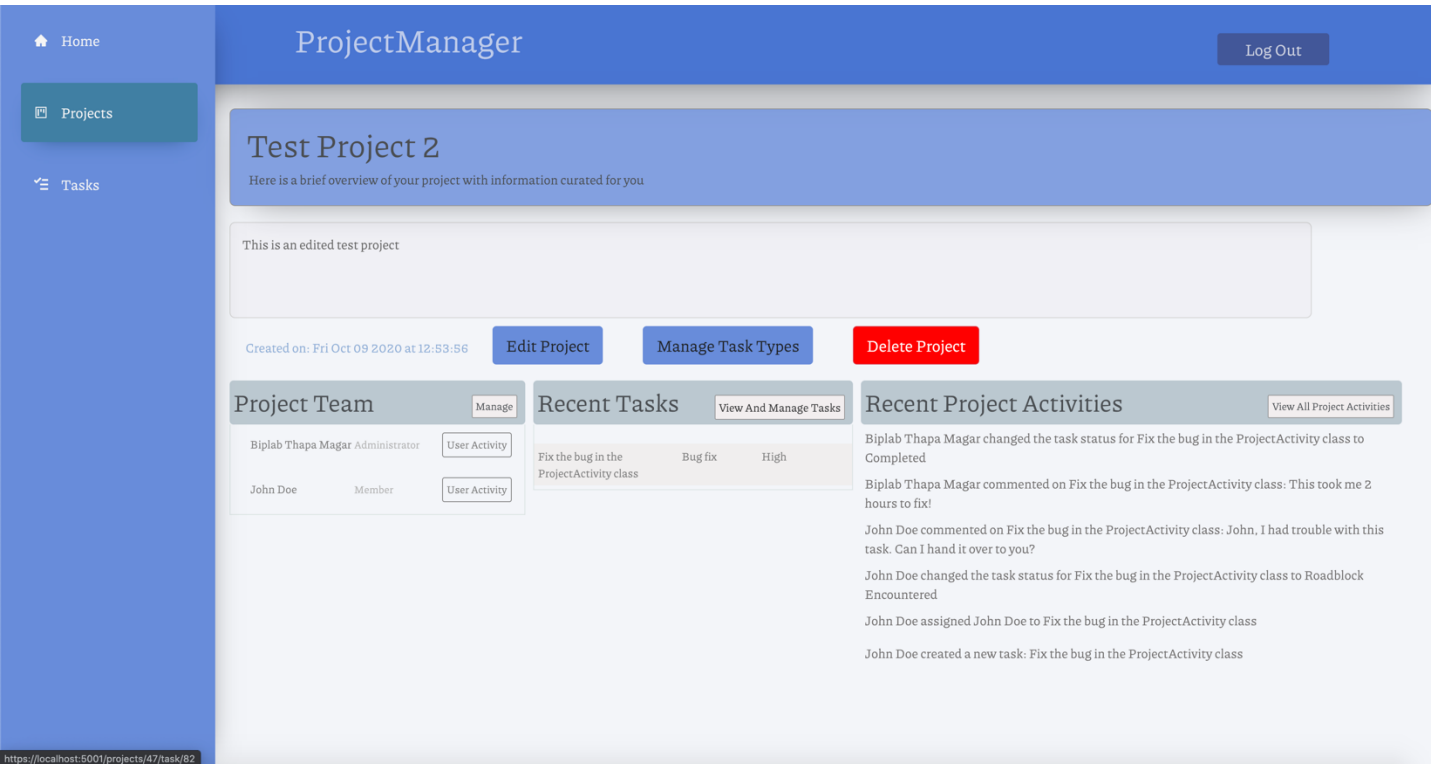
To view the list of all project Activities, the user can click on the View All Project Activities button

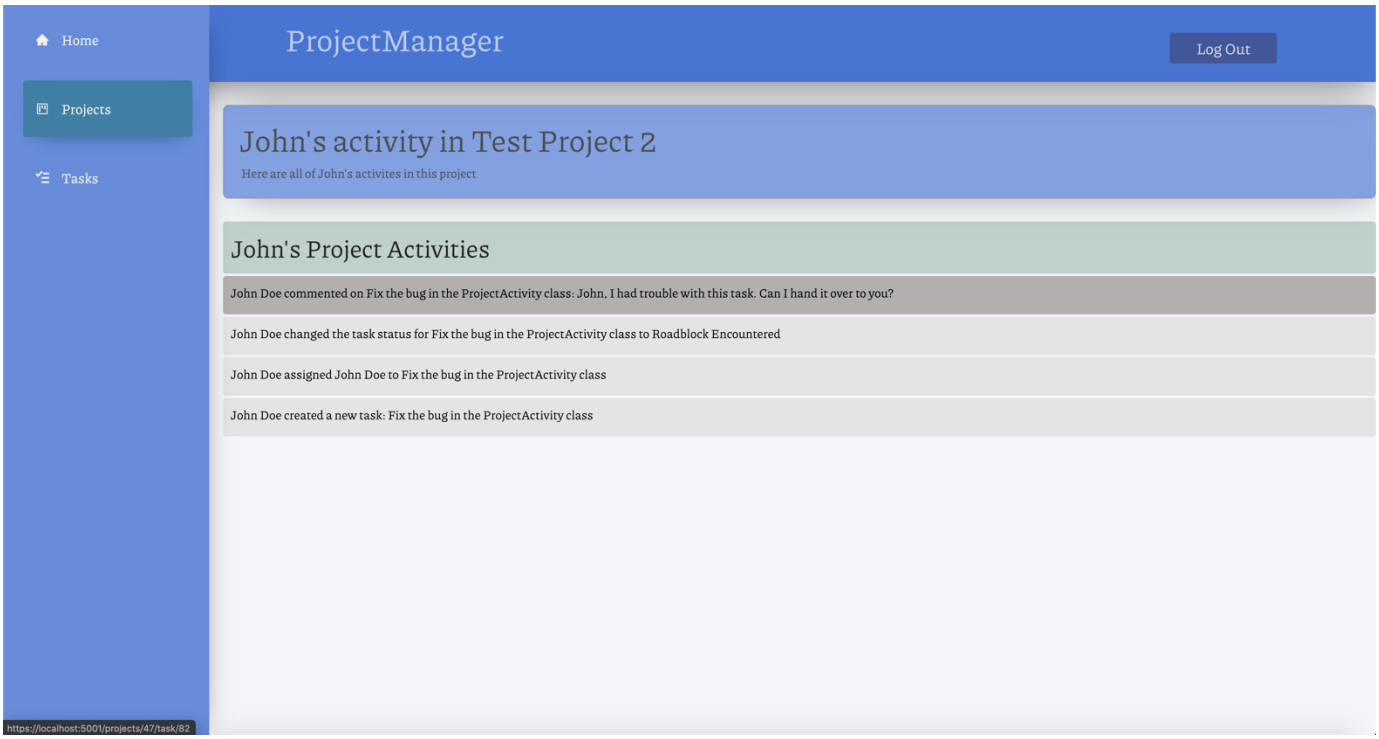


Clicking on one of the listings of the project activity will redirect the user to the task associated with the activity.

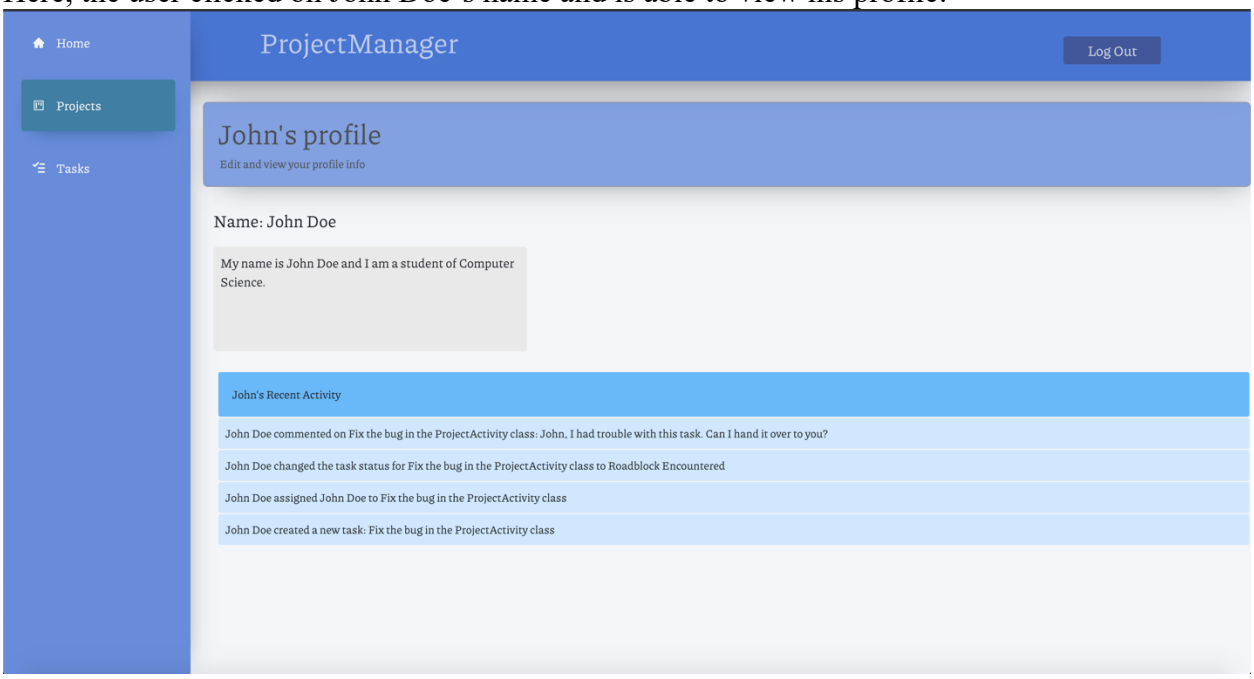
Activities can also be filtered by user, i.e the activities in the project by a specific user can be monitored.

In the main project page, the user can click on the User Activity button next to a project user's name to see his activity in the project


Home
Projects
Tasks
ProjectManager
Log Out
Test Project 2
Here is a brief overview of your project with information curated for you
This is an edited test project
Created on: Fri Oct 09 2020 at 12:53:56
Edit Project
Manage Task Types
Delete Project
Project Team
Manage
Biplab Thapa Magar Administrator
User Activity
John Doe
Member
User Activity
Recent Tasks
View And Manage Tasks
Fix the bug in the ProjectActivity class
Bug fix
High
Recent Project Activities
View All Project Activities
Biplab Thapa Magar changed the task status for Fix the bug in the ProjectActivity class to Completed
Biplab Thapa Magar commented on Fix the bug in the ProjectActivity class: This took me 2 hours to fix!
John Doe commented on Fix the bug in the ProjectActivity class: John, I had trouble with this task. Can I hand it over to you?
John Doe changed the task status for Fix the bug in the ProjectActivity class to Roadblock Encountered
John Doe assigned John Doe to Fix the bug in the ProjectActivity class
John Doe created a new task: Fix the bug in the ProjectActivity class
https://localhost:5001/projects/47/task/82

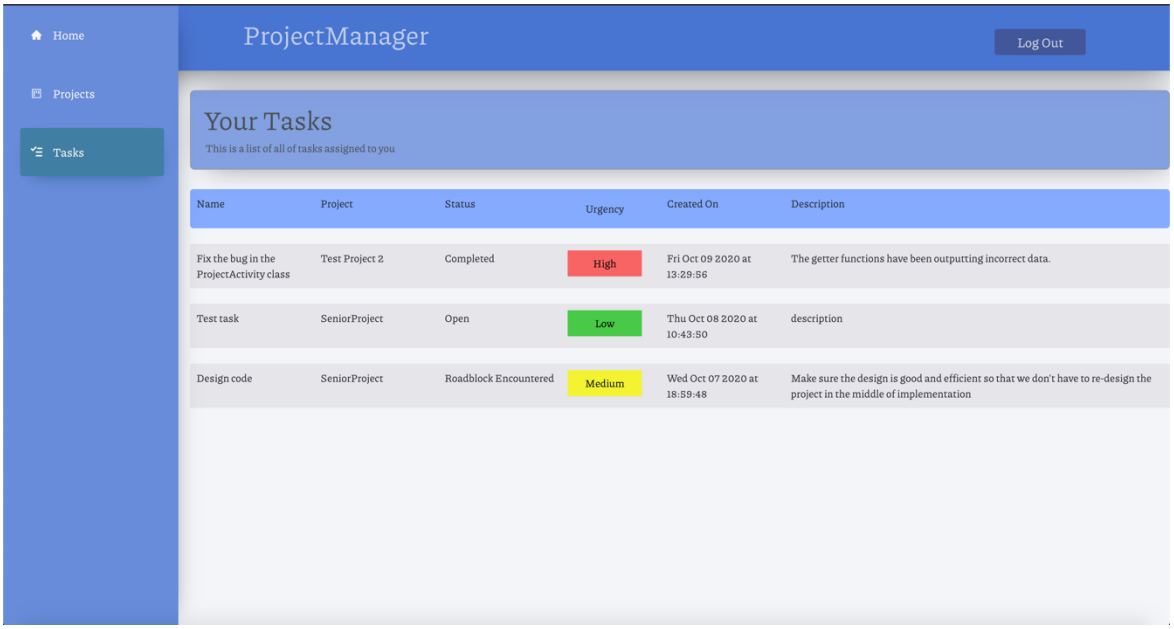

Home
Projects
Tasks
ProjectManager
Log Out
John's activity in Test Project 2
Here are all of John's activites in this project
John's Project Activities
John Doe commented on Fix the bug in the ProjectActivity class: John, I had trouble with this task. Can I hand it over to you?
John Doe changed the task status for Fix the bug in the ProjectActivity class to Roadblock Encountered
John Doe assigned John Doe to Fix the bug in the ProjectActivity class
John Doe created a new task: Fix the bug in the ProjectActivity class
https://localhost:5001/projects/47/task/82

Users can also view each other's user profiles. In the projects space, clicking on the name of the user redirects to the user's profile page.
Here, the user clicked on John Doe's name and is able to view his profile:



The user's profile shows his general information as well as his recent activity. The recent activity is not limited to just one project, but shows his activity throughout all of his projects.

Finally, a user can to the Tasks section in the web application to view all of the tasks, from all of his projects, that have been assigned to him. The latest task is shown first:

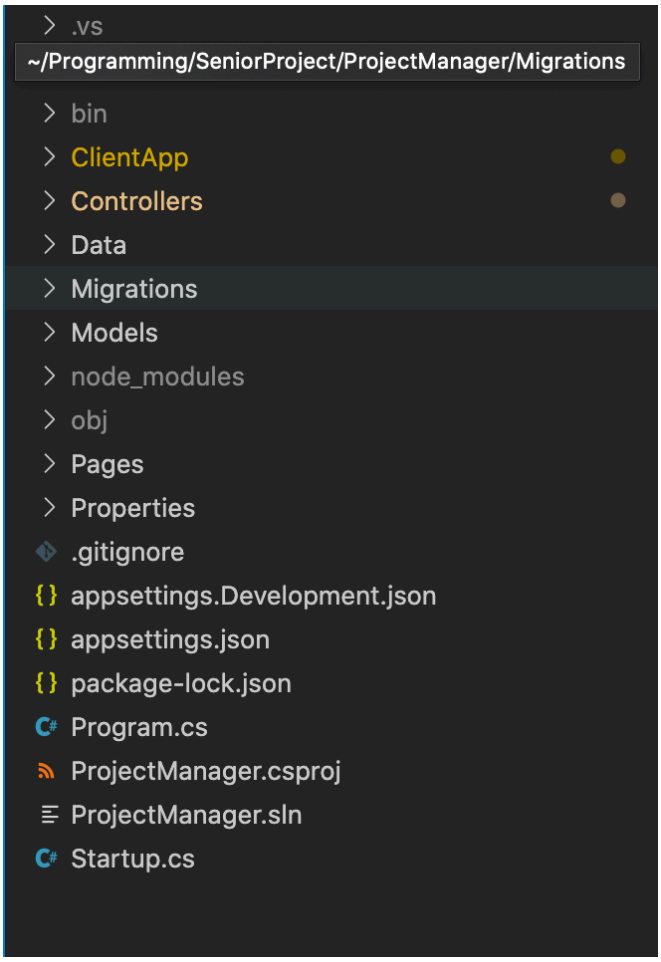


Clicking on a task redirects to the page showing the task's details

Project Design

File Structure:

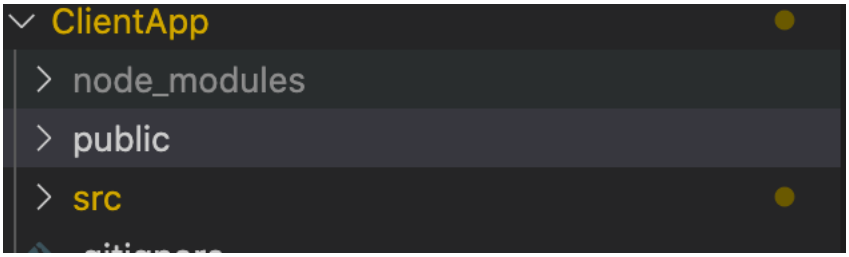
The main directory of the application consists of the following file structure:



Client-Side File Structure:

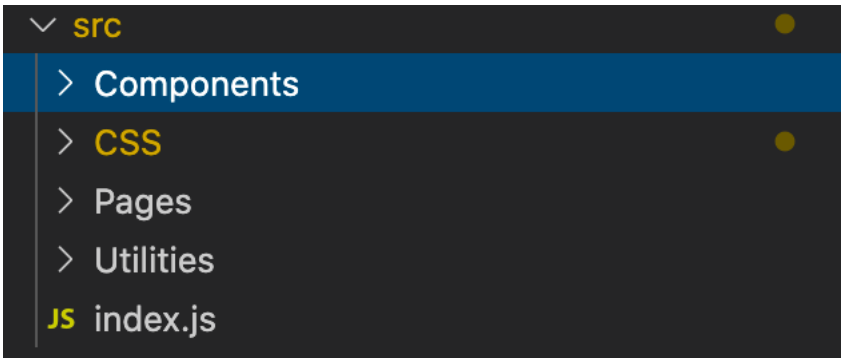
The ClientApp directory contains all the client-side code.

The ClientApp folder, as seen below, contains two folders, the public folder and the src folder.

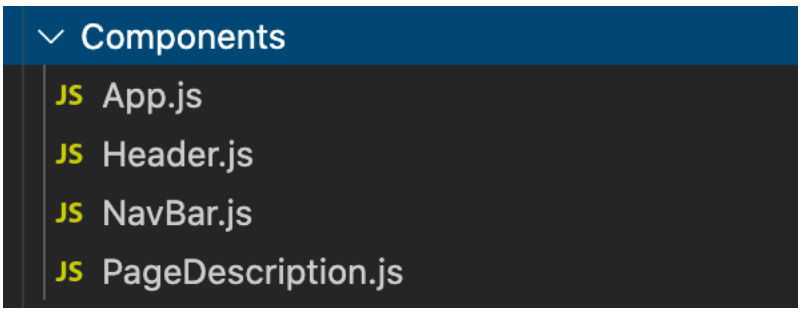


The public folder has the index.html file, which is the page that is rendered to the client. However, this is mostly an empty page, and it into this page that components from the src directory are inserted.

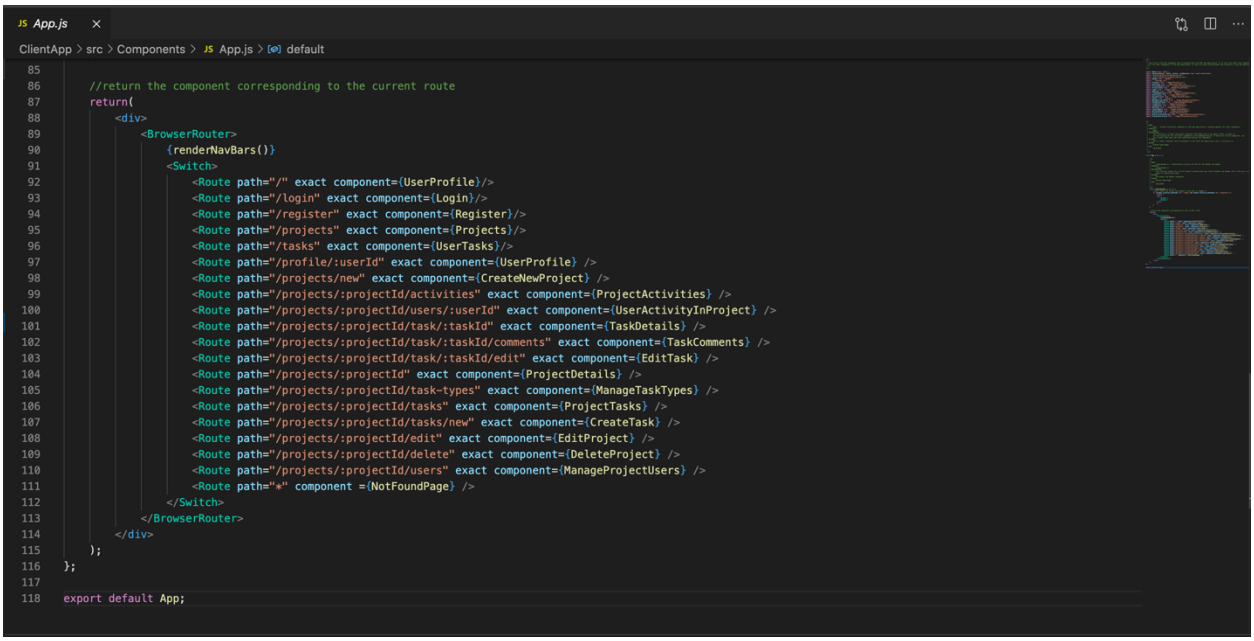
The src folder contains the bulk of the code for the client app. It has four subdirectories:



The **components subdirectory** lists out all the reusable React components used by application. It also contains the App.js file, which defines the routing in the front end application.



App.js: As mentioned earlier, React was the framework used for implementing the client side of the web application. The application uses the React Router component to deal with routing. Using React Router, whenever a user goes to a certain URL, the React Router component decides what component to display based on the URL. In the Project Manager Application, each page is built as a React component. Looking at the App.js page, we can see how the routes and the corresponding page components have been set up.

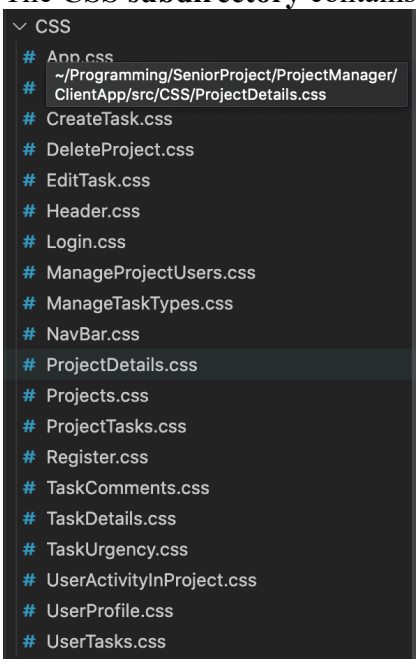


All the components used in the Routing correspond to a page in the application. These pages are in the Pages subdirectory of src folder.

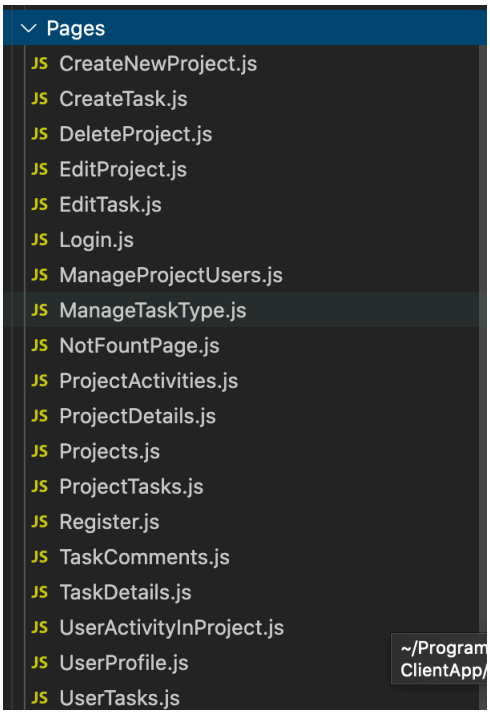
The client-side code separates out frequently used components that are common to many pages into their own files for reuse. These components are also included in the Components folder and they include:

- The Header component, used to display the header in the page
- The Sidebar component, used to display the sidebar in the page
- The PageDescription component, used to display the title and description in some pages

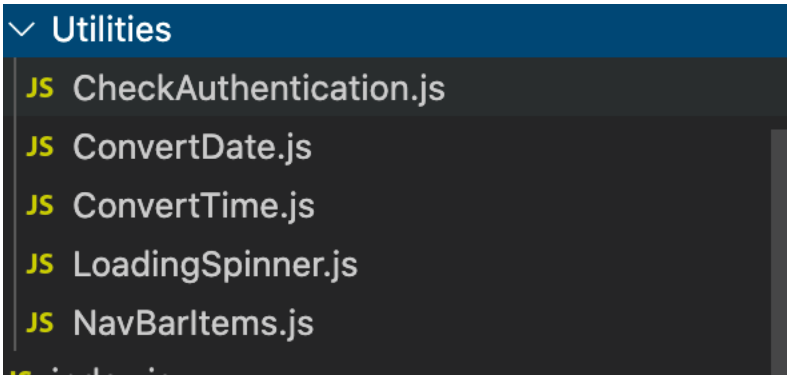
The **CSS subdirectory** contains all the CSS files used for styling



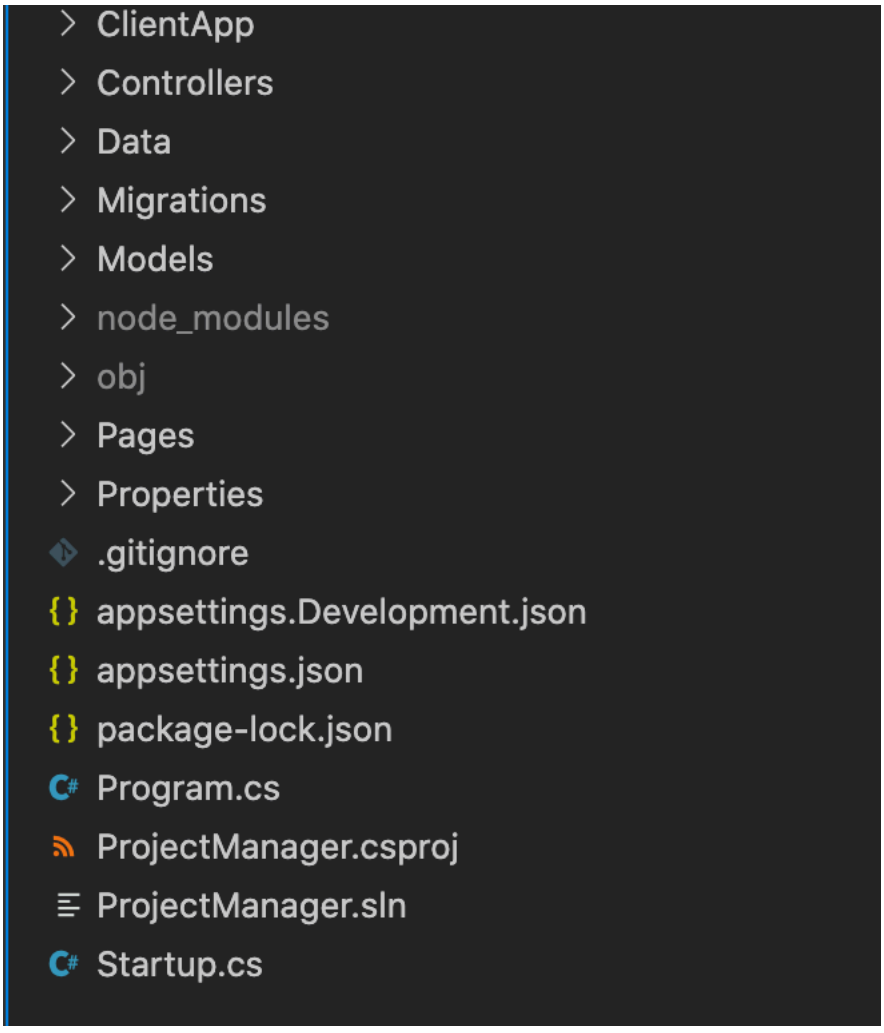
The **Pages subdirectory** contains all the page files. Each page file is a React functional component that corresponds to a page in the web application. Each of these components communicates with the server to fetch necessary data to display to the user.



Finally, the Utilities subdirectory consists of utility functions used by the previous React components.



Server-Side File Structure:
All the other directories and file outside the ClientApp folder contain the server-side code.



The bin, node_modules, obj, Pages, and Properties folders are all generated by ASP.NET Core when setting up the program template.

Likewise, the appsettings.Development.json, package-lock.json, Program.cs and ProjectManager.sln files are all generated by the framework.

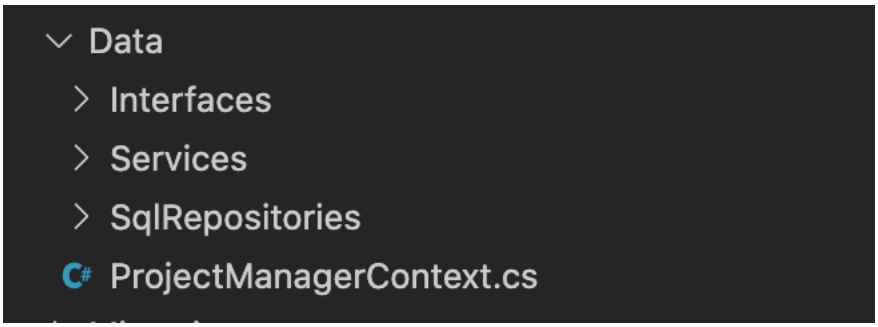
The appsettings.json file is where the connection string to the database is defined.

The Startup.cs file, while generated by the framework, also contains various additions specific to the application. The Startup file is where all the services (whether created within the application or not) that are used by the application are registered. It is also used to define the Middleware Pipeline that HTTP requests to the API has to pass through.

The ProjectManager.csproj file is where the packages, such as EntityFramework Core, are imported into the application.

The **Controllers directory** contains the files representing the three controllers in the application. Controllers are used to define the Web API and to handle requests and responses to the client side.

The **Data** subdirectory contains all the files that have to do with storing, fetching, verifying, and manipulating data stored in the database
It consists of three sub-directories and a file:



The **ProjectManagerContext.cs file** contains the ProjectManagerContext class which defines the database context class for the project manager application. This class is responsible for communicating with the database and setting up the scheme and relations of the tables in the database

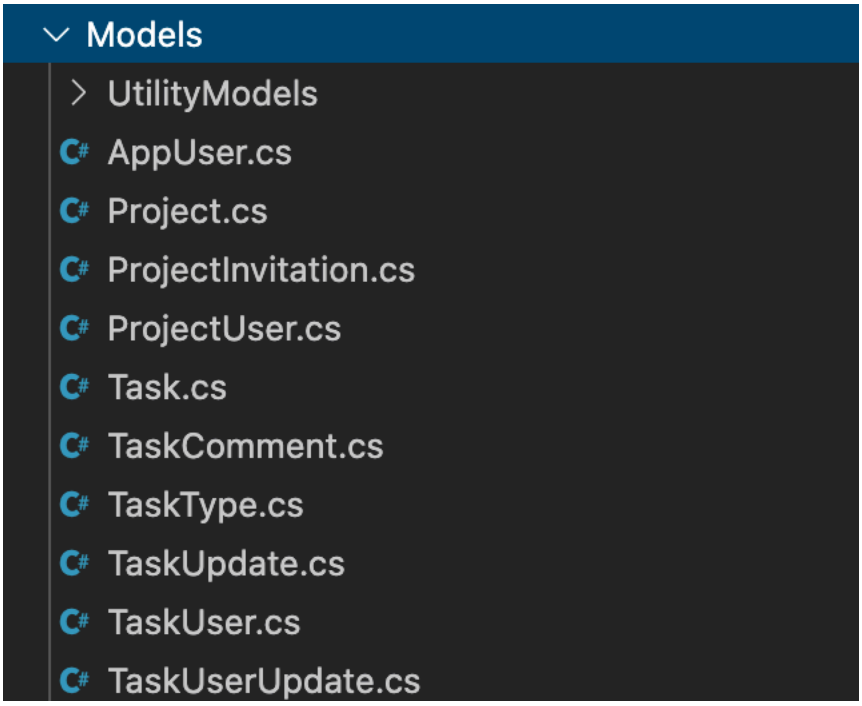
The **Interfaces** subdirectory contains all the interfaces for the data repository. A repository is used an abstraction that provides functions to get data from and manipulate data in the database. It defines functions to communicate with the database that are specific to the ProjectManager application so that other classes, especially the Controller classes do not have to directly work with the database context. Using a repository interface allows programmers to easily switch from one source of data storage to another without much hassle.

The **Services** subdirectory contains all the classes that directly use the data from the database to generate more data or to verify certain information using the data. The services classes, ultimately, provide an elaboration on the data provided by the Repository classes.

The **SqlRepositories** subdirectory contains the SqlRepository classes, which are the implementation of the Repository interfaces. These implementations make use of PostgreSQL as the data store.

The **Migrations** directory is generated automatically by Entity Framework Core and it contains files that represent the process of creating and setting up the database by EF Core.

The **Models** directory lists out all the models that are used by the ProjectManager web application. These models represent all the conceptual entities in the application as well as the tables in the database.



The **UtilityModels** subdirectory consists of all the utility model classes in the application. Utility models are those models that do not correspond to any actual conceptual entity in the ProjectManager web application.

Design

The core of the design of this web application is centered on the backend part of the program, because the client-side is chiefly concerned with UI, and the models and classes that make up the concepts in the ProjectManager web application are not as relevant to the client side part, which uses the backend server part to fetch data to display to the client.

The server side defines all the models, repositories, classes, and interfaces that structure the ProjectManager Web Application. It defines the concepts that the ProjectManager Web Application uses.

Models:

The first and principle aspect of the design of the project is the models used for the project. In C#/ASP.NET Core, models are defined using classes, with or without member functions, but definitely with attributes that define the model. Models, although they are also classes like the other classes defined in the application, play a further special role. Model classes define each entity in the ProjectManager application, and because each entity in the application has to be persisted, the models are used as the basis for the database scheme. Each model generates a table in the database and each attribute in a model corresponds to one column in the database table. The following are all the models used:

AppUser: The AppUser class represents a user in the ProjectManager web application. This class also inherits from the IdentityUser class (provided by the Identity system in ASP.NET Core, which helps handle user accounts and authorization). This class adds additional attributes to the IdentityUser class so that the Model represented by the class fits the needs of the ProjectManager web application

Project: The Project class represents a project in the ProjectManager web application.

ProjectInvitation: The ProjectInvitation class represents a invitation to a project by one AppUser to another in the ProjectManager web application.

ProjectUser: The ProjectUser class represents the inclusion of a user in a project. In the database level, the ProjectUser class represents a join table in a many-to-many relationship between the Project table and the User table.

Task: The Task class represents a task in the ProjectManager web application.

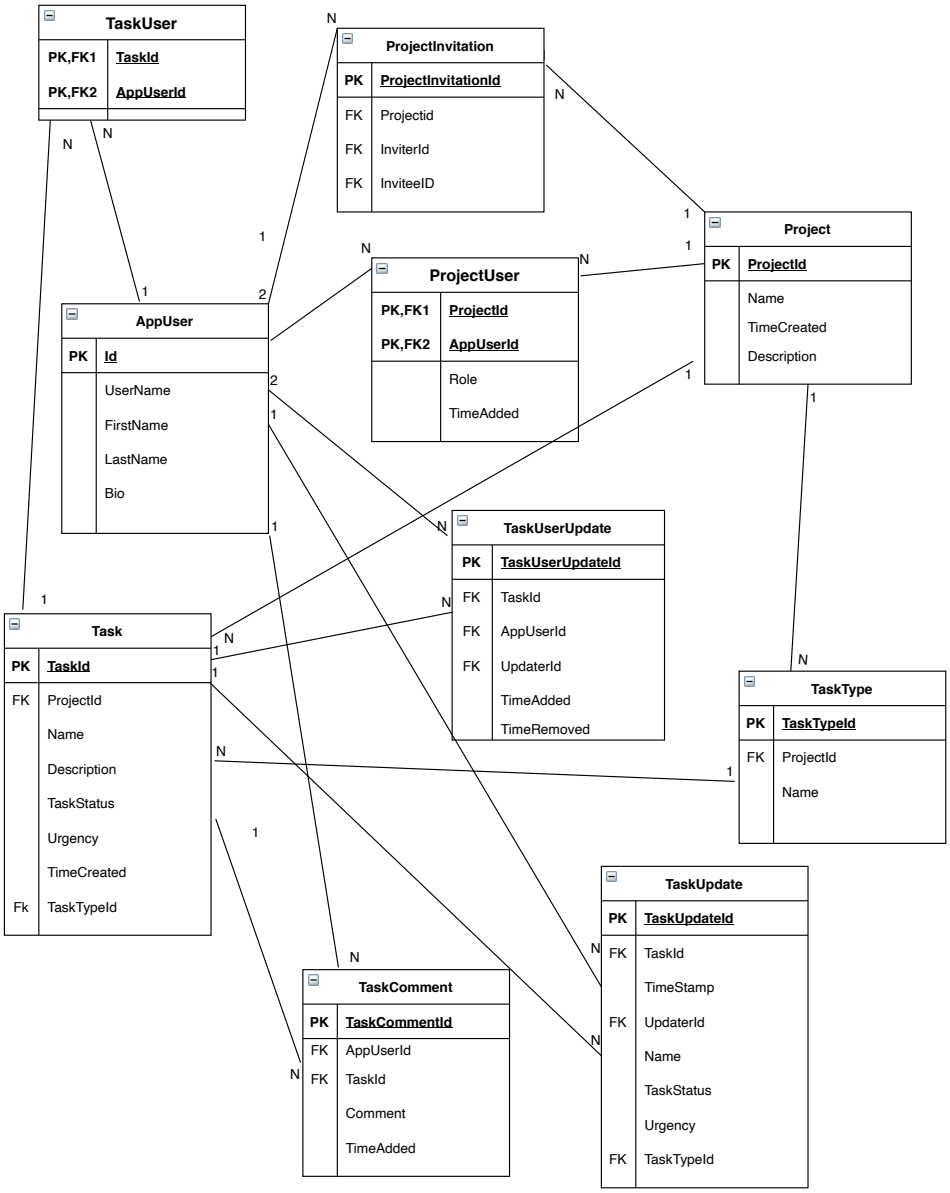
TaskComment: The TaskComment class represents a comment made on a task by a user in the ProjectManager web application.

TaskType: The TaskType class represents a category of tasks defined by an administrator in a project.

TaskUpdate: The TaskUpdate class represents an update made to a task by a user in the ProjectManager web application. In the application and the database, the Name, TaskStatus, Urgency, and TaskType attributes are uninitialized by default. If any of these attributes is initialized to a certain value, the web application interprets that particular attribute as the attribute that was updated in the task

TaskUser: The TaskUser class represents an assignment of a user to a task in the ProjectManager web application. In the database level, the TaskUser class represents a join table in a many-to-many relationship between the Task table and the User table.

TaskUserUpdate: The TaskUserUpdate class represents an update in the users assigned to a task in the ProjectManager Web Application.



Setting up and managing the database: The ProjectManager application makes use of the Entity Framework Core (EF Core) library for database access. The EF Core allows for C# classes to be transformed into models in the database, with all the relationships between the tables being definable by the programmer.

Furthermore, EF Core also provides a DbContext class, which is inherited by the ProjectManagerContext class. This context class allows provides an interface from which to interact with the database, making it very simple to query entries in the database

Repositories: Repository classes are another key aspect of how data is handled in the ProjectManager web application. Repository classes represent another type of abstraction over the database layer. In ProjectManager, a repository class groups a specific type of resource into one "repository", which provides an interface to get, add to, and manipulate data stores in that "repository". In the ProjectManager application, this grouping is set up in terms of four resource categories: Projects, Users, Tasks, and TaskTypes. That is why there are four (SQL) repository classes defined, each class corresponding to each one of these categories. The repositories all make use of the ProjectManagerContext class in order to interact with the database, but each repository class only defines interactions with relevant resources. So, the projects repository does not deal much with the users repository. However, this constraint is complicated, since many times, like for example when we need a list of all the users in a project, one resource is linked inseparably to another. Careful considerations have to be made in these instances to decide which repository should handle such requirements.

Setting up user accounts: User accounts were set up with the help of the Identity system provided by ASP.NET Core. The Identity system also makes use of EF Core to generate tables that correspond to the data stored for the user. In fact, the Identity system provides its own IdentityUser class, which represent a user of the application in the database. The ProjectManager application's AppUser model/class inherits from the IdentityUser class to get all the functionality of Identity while also extending attributes specific to the ProjectManager web application. Identity also allows for authentication. When a user logs in, Identity sends a cookie to the user's browser, representing a signed in state.

Setting up the Web API: The Client side uses the web API provided by the server side in order to communicate with the server through HTTP requests and responses. The mechanism for defining the routes and behavior of each API request in ASP.NET Core are through the Controller classes and the Action methods within these controller classes. Controller classes are a grouping of Action methods that handle HTTP requests to specific URL endpoint requests. What endpoints are handled by what Action methods are defined by the the [HttpGet], [HttpPost], and [HttpDelete] attributes placed on top of the action methods. The [ApiController] attribute applied to the Controller classes declare that these classes are controllers and the [Route] attribute define the URL path for the controller class and its methods.

Classes used (excluding Model classes):

Utility Model classes:

Utility models are models that are differentiated from the regular models in that they are not used to define database tables/schemes. Instead, utility models are used as containers of data within the server-side application, mostly when receiving HTTP POST requests from a client of the Web API or when sending data back to the client. These models are used to populate the body of HTTP requests.

RegisterUserModel: The RegisterUserModel class is used when a user of the application logs in/registers and the login/registration information is sent to the server.

UpdateUserModel: The UpdateUserModel class is used when a user of the application updates user information and the information is sent to the server.

UserActivity: The UserActivity class is used by the ProjectActivity class to store generated project activity information in the ProjectManager web application. This object is then used to send activity information to the client-side.

UtilityInviteModel: The UtilityInviteModel class is used when a user of the application invites another user to a project and the invitation information is sent to the server.

UtilityProjectModel: The UtilityProjectModel class is used when a user of the application creates a project and the project information is sent to the server.

UtilityTaskCommentModel: The UtilityTaskCommentModel class is used when a user of the application posts a comment on the client side and the comment information is sent to the server.

UtilityTaskCreateModel: The UtilityTaskCreateModel class is used when a user of the application creates a new task for a project and the task information is sent to the server.

UtilityTaskEditModel: The UtilityTaskEditModel class is used when a user of the application edits a task and the task information is sent to the server.

UtilityTaskTypeModel: The UtilityTaskTypeModel class is used when a user of the application creates a new Task Type and the task type information is sent to the server.

Controller Classes:

A controller class in ASP.NET Core is responsible for defining and handling HTTP requests to specific routes.

AccountsController: The AccountsController class handles routes and HTTP requests relating to user accounts in the ProjectManager Web Application

ProjectsController: The ProjectsController class handles routes and HTTP requests relating to everything specific to Projects in the Project Manager web application

UsersController: The UsersController class handles routes and HTTP requests relating to everything specific to User in the Project Manager web application

Repository Classes and Interfaces:

A repository interface defines the list of all the public functions for a repository used by the web application.

IAppUsersRepo: The IAppUsersRepo interface represents a repository that contains all information on the users of the ProjectManager web application.

IProjectsRepo: The IProjectsRepo interface represents a repository that contains all information on the projects of the ProjectManager web application.

ITasksRepo: The ITasksRepo interface represents a repository that contains all information on the tasks of the ProjectManager web application.

ITaskTypesRepo: The ITaskTypesRepo interface represents a repository that contains all information on the task types of the ProjectManager web application.

A repository class provides a layer of abstraction over a specific type of data resource and allows application-specific functions to fetch and manipulate the data. Repositories are implementations of the above repository interfaces. In the ProjectManager application implementation of repositories, the repositories communicate with a PostgreSQL database through an instance of the ProjectManagerContext object.

SqlAppUsersRepo: The SqlAppUsersRepo class is an implementation of the IAppUsersRepo interface.

SqlProjectsRepo: The SqlProjectsRepo class is an implementation of the IProjectsRepo interface.

SqlTasksRepo: The **SqlTasksRepo** class is an implementation of the ITasksRepo interface.

SqlTaskTypesRepo: The SqlTaskTypesRepo class is an implementation of the ITaskTypesRepo interface.

Service classes:

These classes represent miscellaneous classes.

ProjectActivity: The ProjectActivity class is used to generate information that defines the activities in a project, activities by a user, or activities by a user in a project. The

ProjectActivity class gathers disparate data from the database and interprets the data in order to generate easily comprehensible information regarding project activities

ProjectMemberValidation: The ProjectMemberValidation class is used to determine whether a user making the API calls is a valid member/administrator of a project. This class is used primarily by the Controller class/Action methods in order to verify that API requests for accessing and manipulating resources are coming from only those users who are allowed to access the given resources.

Summary and Conclusion

This project was a surprise, not only because I had to go through the journey of completing it in an incredibly surprising and bewildering time in history, but also because the way it caught me off guard with its difficulty. The difficulty I faced was not related to any concept that was hard to understand. The difficulty lay in the sheer number of new concepts I had to learn just to get started on the project. Having had no experience working in web development at all, I self-learned concepts on web applications and network communications. Furthermore, I had to learn a new language, C# and needed to better understand JavaScript, of which my knowledge was only rudimentary. After learning the languages, I had to learn the frameworks that use these languages ASP.NET Core and React. Especially with ASP.NET Core, I had to learn about numerous software design techniques and paradigms that I had never heard of before. I also had to learn about Entity Framework Core and Identity, which brought with them whole new concepts I had yet to grasp.

Little did I realize when I started this project how much I had to learn and how much I didn't know. But the difficulty of this project was incredibly rewarding for me. I encountered many frustrations (in the form of bugs) when I did not understand a particular concept properly but still went ahead with coding. These were valuable lessons for me.

The fact that I had to work on this project in the middle of the many vicissitude afflicting the world made this even more of a difficult, but still an incredibly insightful experience

I am very thankful for having done this project because it truly made me grow as a programmer.