School of Electronic Engineering and Computer Science

Final Report

Programme of study:
BSc Computer Science

Project Title: Minimal Manager

Supervisor:
Matthew Huntbach

Student Name:
Al-Noornabi Al-Qausayen Al-Makki

Final Year
Undergraduate Project 2022/23

Queen Mary
University of London

Date: 2/05/2023

Abstract

One of the cornerstones of human progress is the ability to work together to build projects that solve problems. Projects may be built individually or as a team. Some are simple that require only a few steps while others are complex and may require an astronomical amount for successful completion. Whatever, the size or the number of participants involved it is imperative when undertaking a task of this nature that we can plan and record progress to find an optimal way to meet the requirements for project, as resources such as budget and/or time are finite. Hence, optimal utilisation of resources is vital. This is known as project management and brilliant minds have come together to formulate principles that increase the rate of success. This report takes a deep dive into these principles and describes an attempt to produce the implementation of a system that utilises them. Particularly, details about requirements development, stakeholders, a final prototype that may have met the requirements and plenty more.

Contents

Chapter 1: **Introduction**

1.1 Background

The difference in technological progress that exists between humans and other species is evident all around us. The key to this progress is our drive to build tools or projects that solve a problem. Upon completion, the success of a project is determined by whether it has appropriately addressed the issues for which it was built. As society progressed, the complexity of projects has grown rapidly. This is particularly true for software development. Due to this rapid growth, a challenge has arisen. It is to ensure the optimal allocation and usage of resources to achieve successful completion of a project. Resources are time, capital, labour, or any commodity that contribute to building a project. This has led to the development of project management techniques that improve how resources might be allocated.

One such technique is the agile manifesto.[1] It is simply 4 values and 12 principles by 17 software development practitioners who sought to simplify and optimise the software development process. Inspired by these principles, many project-management tools such as Asana, Microsoft project management, Smartsheet and Basecamp. Studies have shown them to increase productivity with varying degrees for specific projects.[2] Hence, educators are now implementing Project Based Learning to teach software engineering to students trying to break into the field.[3]

1.2 Problem Statement

Although, great progress has been made in Project Based Learning for software engineers, improvements still need to be made.[3] Project-management is one. Existing project-management tools do address this issue. The target audience of these tools is quite diverse too which has its advantages however, it means that either certain niche requirements of students and educators may be unmet or have too many features that prevent learners with short attention spans to pick up the technology. Furthermore, these tools are only free for certain groups of students and the price puts it out of reach for others.

1.3 Aim

It is imperative that there is a system that addresses the requirements of students and educators while at the same time being simple enough to learn to use for those with short attention spans. The system must implement a user interface that students and educators may interact with intuitively without the need for much training. It should encourage iterative development and allow educators to supply feedback on features that have been implemented. Students should then make changes to their project accordingly and seek further feedback from educators. Finally, the system should enable teams to divide project workload equally and update tasks and responsibilities assigned to members.

1.4 Objectives

The following list of objectives are crucial for the successful implementation of the proposed system.

- Study related products and identified features that may be useful.
- Take inspiration from agile project management to develop functional requirements.
- Maintain a log of problems faced while developing the project so which could be used as a source for potential requirements.
- Identify the various actors and primary stakeholders for the project
- Build the user interface and ask the stakeholders to test it and provide feedback.
- Learn and employ a test-driven development strategy.
- Build a backend Api.
- Deploy the application.

1.5 Research Questions

This research is based on the following questions.

1. Why is the system based on agile project management instead of other approaches?
2. What are the key features of a project management system for students and educators?
3. How to implement such a system?
4. Can a minimalist design improve learning for students with short attention spans

Chapter 2: **Literature Review**

2.1 Primary Stakeholders

The primary stakeholders are the main actors that interact with the system. This system is designed for students and educators therefore they are the primary stakeholders. Worldwide the number of students studying computer science is larger when compared to other engineering fields. In South Korea, 145,718 (fig, 2.1.1) and India 935,270 (Figure 1), it is by far the most popular one. Therefore, it makes sense to have educators and students as our primary stakeholders.

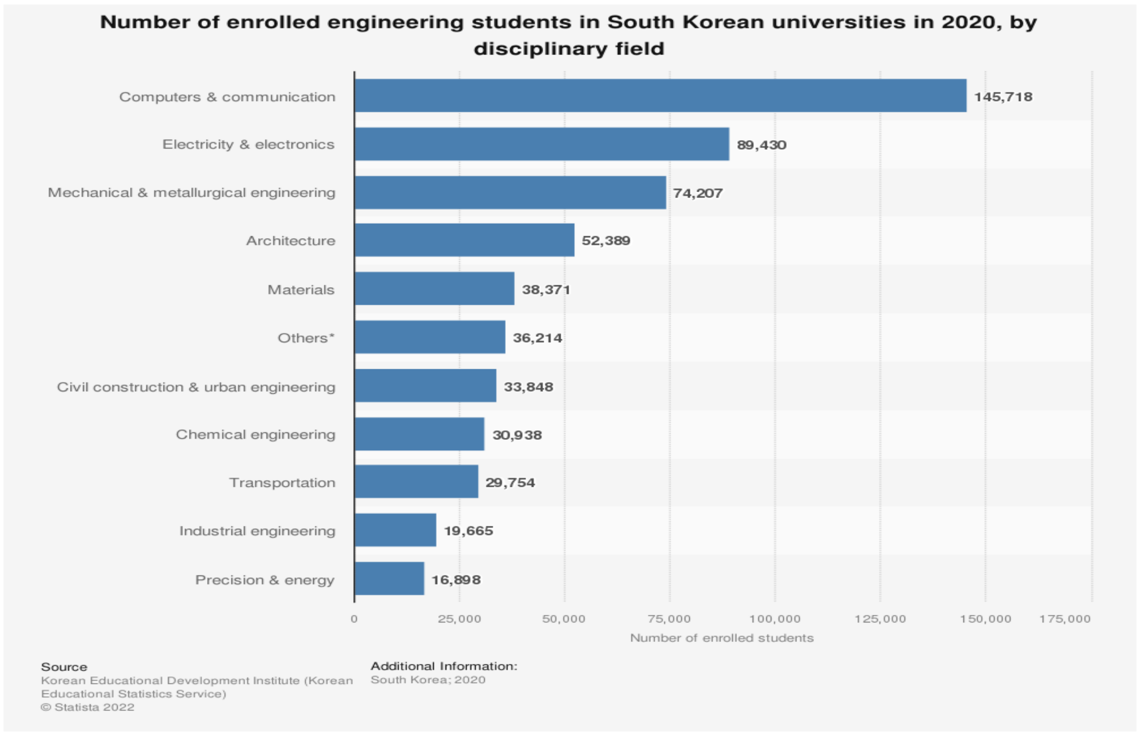


Figure 1. Number of enrolled engineering students in South Korean universities in 2020 by disciplinary field

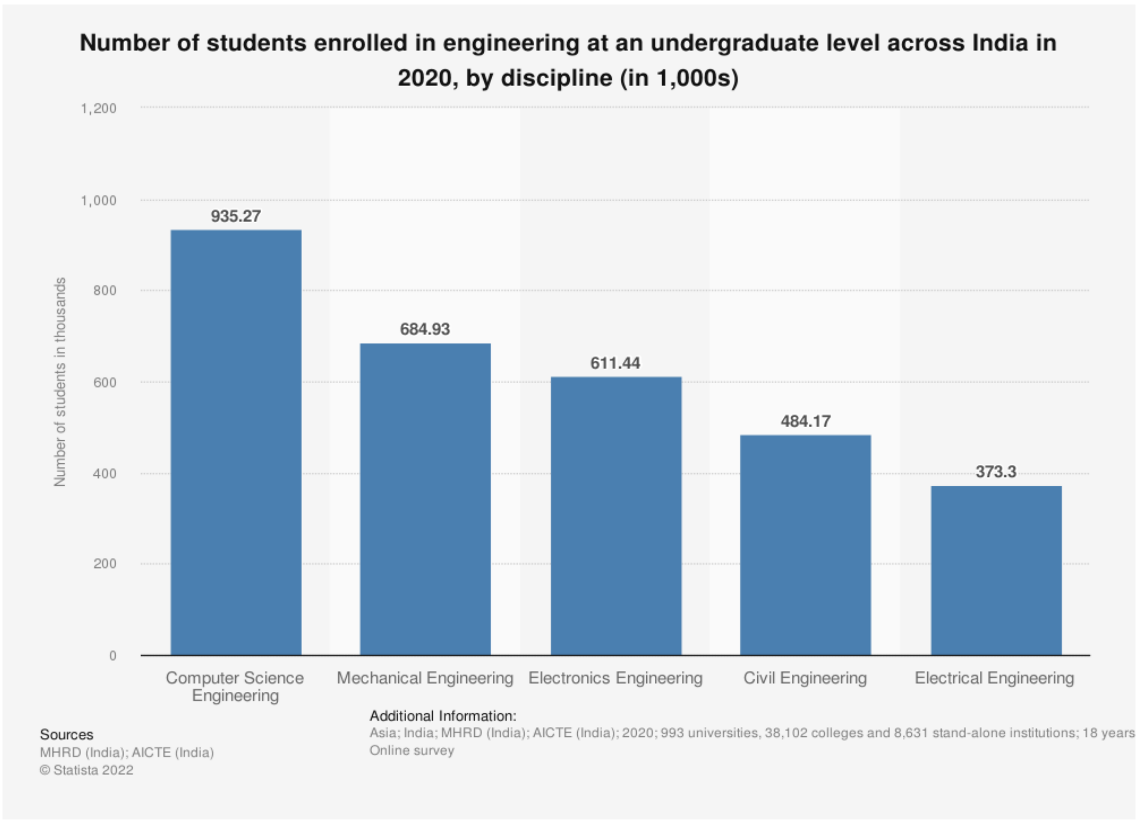


Figure 2. Number of students enrolled in engineering at undergraduate level across India in 2020, by discipline (in 1000s)

2.2 Agile Project Management

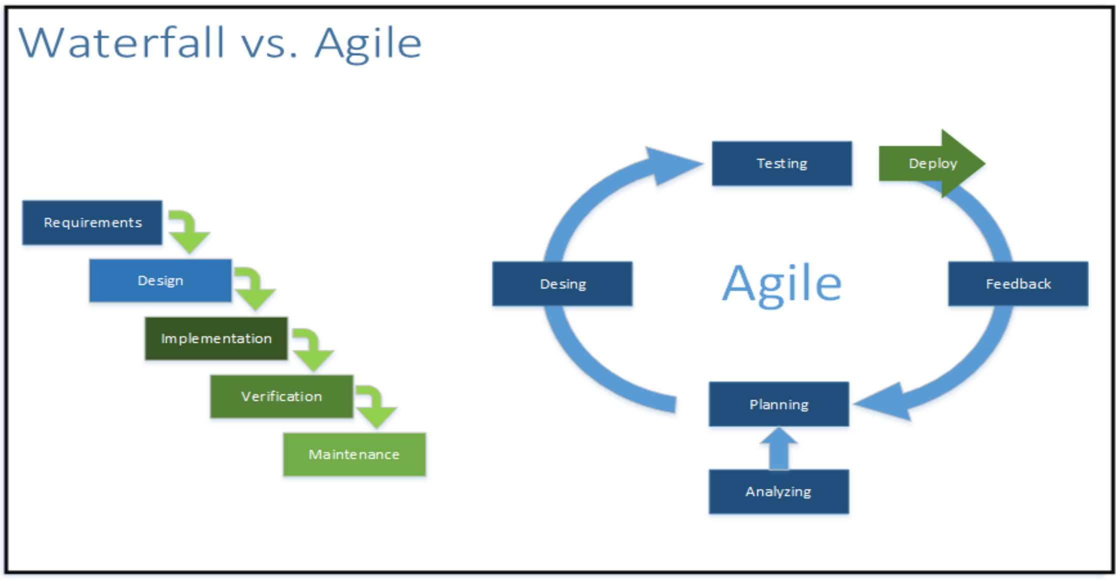


Fig 2.2

Figure 3. Waterfall vs Agile Project management.

We will now look at 2 popular software management techniques, agile and waterfall. In Figure 2 we can observe that although the two methodologies go through similar stages agile is iterative while waterfall is sequential. Both have their pros and cons. In waterfall requirements are clearly defined in the first stage followed by design, implementation, verification, and maintenance. Each step is completed before moving on to the next giving it a linear rigid structure making the process easy to implement while at the same time reducing the quantity of resources required.[5]

This rigidity means that if we are at the later of the development process and we want to change for example requirements that a customer wants added it is impossible to do so. This process is solved by the agile method. After quickly analysing and planning a prototype is designed, tested, deployed, sent for feedback and back to planning until the customer is satisfied.[5] Contrary to waterfall agile is more difficult implement, however the flexibility it offers gives it an edge. To make agile easier to implement our system must have the ability for students to record, update and delete requirements, while also allowing the customer (in this case the educator) to be able to provide feedback on regular basis. More on this in the proposed solution section.

2.3 Similar Systems

2.3.1 Asana

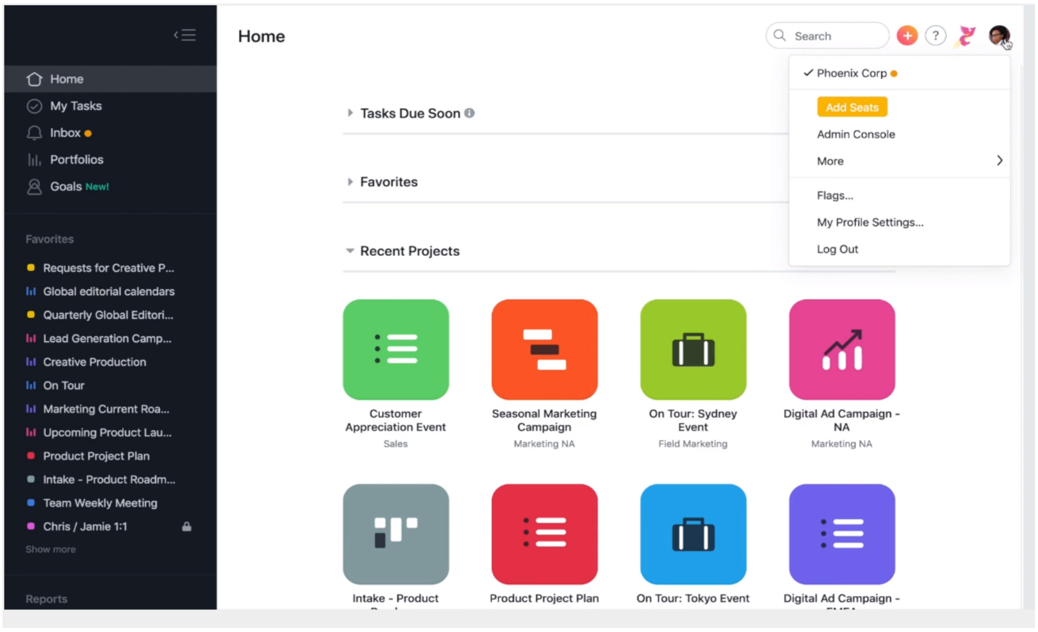


Figure 4. View of asana from tutorial video.

Asana (Figure 4) is a versatile project management tool available both online and mobile. It has built in components that such as list view, timeline, views, and pre-built templates which make it possible to be used for a diverse range of projects. Some useful features include allowing users to assign priorities to the different created projects, manage permissions, for projects and finally the interface is intuitive.[6] However, the sheer number of features it implements might be daunting for students who are new to learning agile development. There is a task functionality that is like a requirements list, that allow to add lots of information about each requirement, which makes it suitable for planning personal tasks however a better implementation of a requirements list should be minimal. Details such as a short one-line description, priority, completions status and name or email of the team member assigned to are enough.

There are 4 price tiers Basic, Premium, Business, and Enterprise. Basic is the most viable for students as it is free. It consists of unlimited tasks, projects, activity log, file storage lots of other features, to justify its usage.[7] Again, this makes it a good personal task planner system but not fit for the problem we are trying to solve.

2.3.2 Other Similar Systems

Other viable project management systems, such as Microsoft project planner, Planner Suite, SmartSheet, etc are like asana with few differences.[8] All existing solutions put an emphasis on personal task management and are not suitable for the problem discussed in this paper. The ideal solution would be a system that allows students to enter requirements as discussed in 2.2.1. The User Interface can be based on Asana. Additional features like a page to provide feedback from educators/customers should be added. This feedback needs to be sorted by time. There also needs to be a feature for reporting bugs, assign resolution to a teammate and track resolution status. The bug resolution and requirements can be broken down into tasks and sub-tasks in a personal daily task manager page. Further details about how this can be achieved can be found in the Chapter 3 Proposed Solution.

2.4 Proposed Solution

2.4.1 Platform

It is vital that our system be able to run on multiple operating systems, as the market is shared by multiple operating systems (Figure 5).[9] Designing and maintaining a system that that is compatible with all these operating systems offline is an extremely time-consuming process making it impossible to achieve with the resources available for this project. Thus, a web-based application makes the most sense to achieve platform independence as it will make it possible to access the system by a browser. Therefore, making we must make sure that we choose languages and frameworks that are supported by a wide range of browsers.

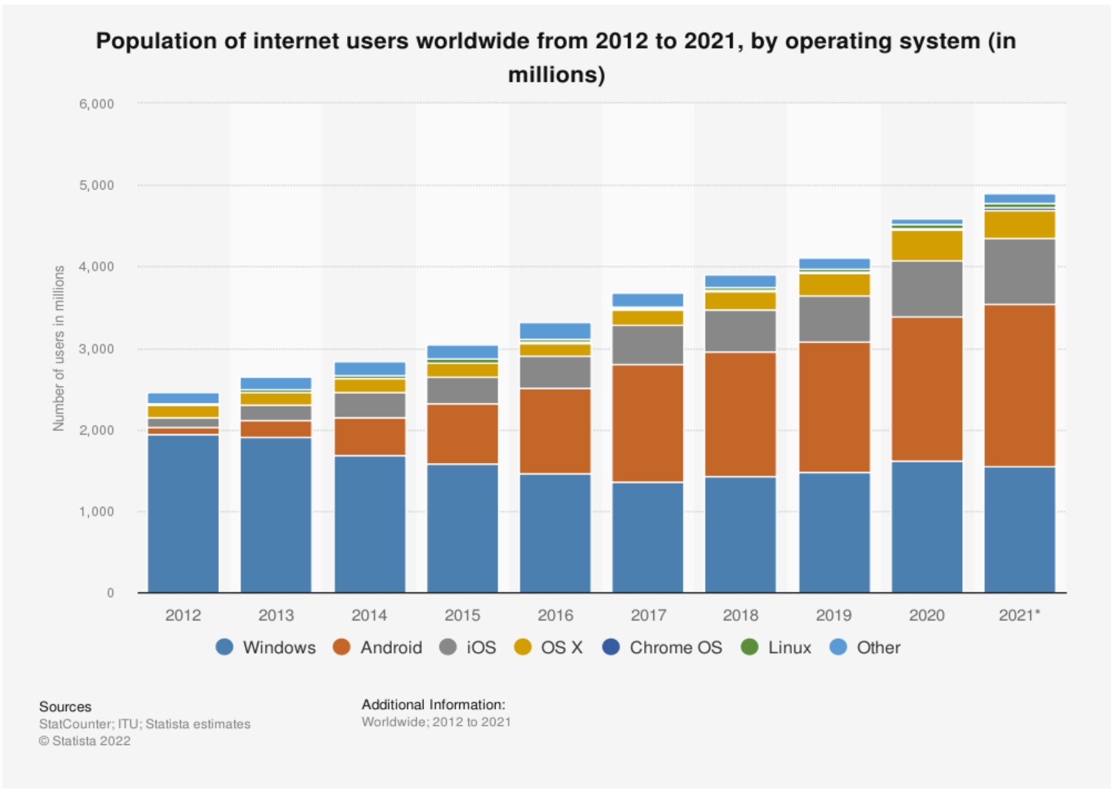


Figure 5. Fig 2.3.1.1 Population of internet users in from 2012 to 2021.9

2.4.2 User Interface

As mentioned, before our system is a web application. The core of the user interface will be made using Html, CSS, and JavaScript. Initially, the framework of choice was React but after learning Vue from the ECS639U – Web Programming module the decision to switch to Vue and bootstrap was made. This is because Vue is more beginner friendly, with its smaller size of the new runtime library, better documentation while also providing a modular structure and core library enabling flexibility [10]. Bootstrap is a CSS framework that allows developers to produce responsive websites using built-in components, grid-system, and JavaScript plugins. These styles can be further customised using Sass.

While there are several viable frameworks and libraries, it is more important to recognise users' attention as a limited resource and design the interface with components that they are already familiar with, that is their pre-existing knowledge through interacting with systems that have a similar interface.[11] Although the latter is achieved by the systems explained in section 2.3, they fail to satisfy the intended goals of the system discussed in this paper. This is due to the sheer volume of features that these systems are cluttered with, which make them versatile but demand a huge amount of user attention a resource that is already in short supply to users with low attention spans.[12] Therefore, the solution must be system that utilises these components but displays the minimal necessary features to accomplish the goals of the system. Chapter 3 will describe the how to use the user interface.

2.4.3 Data

The database will consist of a Django Api with PostgreSQL[14] be deployed on OpenShift. Django provides an MTV (Model-Template-View) Architecture where a model stores the data in appropriate fields and has a one-to-one mapping with our PostgreSQL[14] database, the template interacts with the user and the view simply connects the model layer to the template.[13] On the other hand PostgreSQL[14] is an object-relational database that has MVCC (Multi-Version Concurrency Control) which resolves concurrency issues while reading and writing data, is highly secure[15] and comes with lots of other useful features such scalability etc.

2.4.4 REST (Representational State Transfer) Architecture

Our design of the system will be based on Roy Fielding's conception of REST architecture [19]. It is a style for building distributed systems, based on the use of standard HTTP verbs (GET, PUT POST and DELETE) and resources. These resources are identified by URIs, and operations on those resources are performed using the HTTP protocol. An application designed using these protocols is called a RESTful web application.

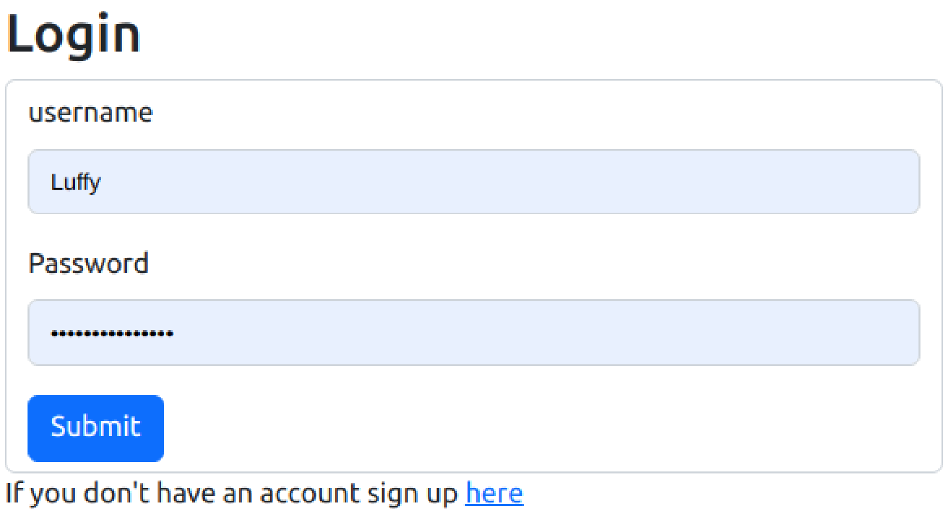
We are going to use the verbs in tandem with Vue and Django Rest Framework (DRF) two popular technologies to build minimal manager. This will be examined further in Chapter 4, where the development process and the key aspects of the protocols, namely CRUD which is short for create, read, update, and delete, will be explained. We will also look at how the client and the server interact with each other when the user performs these actions.

Chapter 3: **Guide For Users**

To explain the features of the system we shall use a simple project scenario. The name of the user is Luffy, and he wants to collaborate with Zorro and Sanji to make an omelette. They are all students of wilderness survival expert Zeff who is acting as an educator. They are in a forest and the project assigned to them is to make an omelette. They divide up tasks for the project and will use minimal manager to track progress and division of labour. It is highly unlikely that the system will be used for a project as simple as this, but this simplistic scenario may help users to understand how to use minimal manager for projects with varying degrees of complexity.

3.1 Login and Signup

The app is accessible by most browsers. If a user is not logged in, they are presented with a login form (Figure 6).



Login

username

Luffy

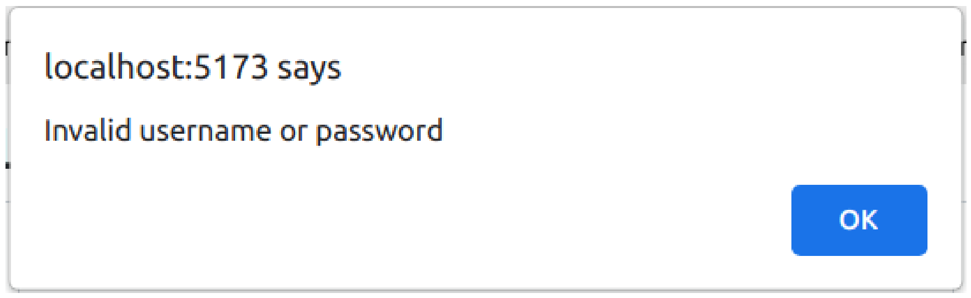
Password

•••••••••••••••

If you don't have an account sign up here

Figure 6. Login Page

The correct username and password must be provided to access the app. An incorrect username or password will prompt this error message in a pop-up box. This box will be used throughout the system and appear when the user makes an error or confirmation is required for sensitive tasks (Figure 7).



localhost:5173 says

Invalid username or password

Figure 7. Error message for incorrect username and password

Additionally, a new user will be able to create an account clicking the link in the form below. Here (Figure 8) the user must provide a valid username, password, and type of account. The password must be entered correctly twice, and the email must include an @ symbol. As this is a prototype, an account can be created for instance for a user named Harry and email Harry@user.com. The username must be unique otherwise, it will prompt an error message.

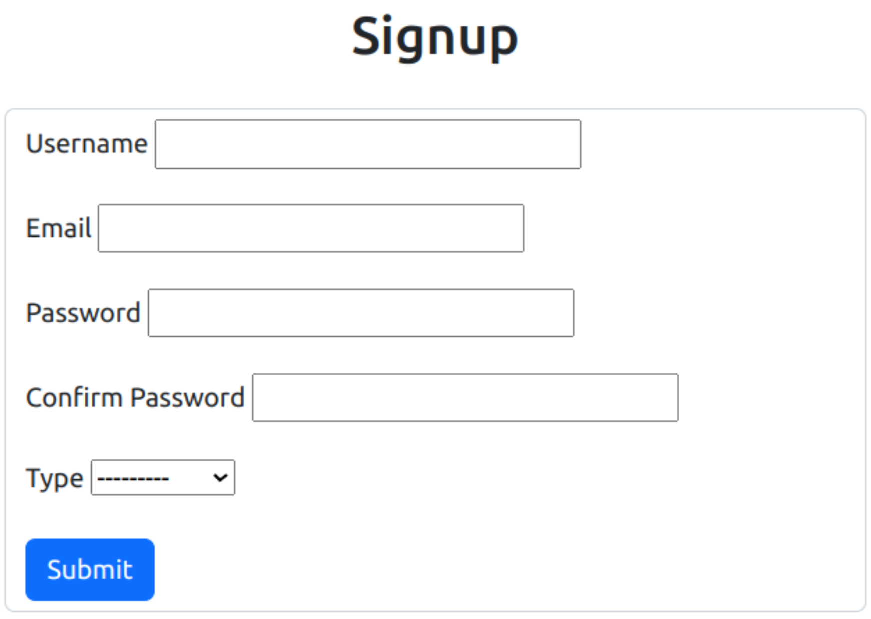


Figure 8. Signup form

3.2 Projects and Navbar

Assuming a user has properly logged in, they shall be greeted with instructions on how to use the system on the home page. The first component we shall look it is the project (Figure 9) located on the left-hand side of the screen. This component has two parts a form and a list with some text and two buttons. Every component in minimal manger follows this exact structure.

3.2.1 The Form

The form (Figure 9) is where a user can enter the name of a project and submit the form pressing the submit button. It is not possible to submit a blank form. The project name must also be within 50 characters spaces included.

3.2.2 The List

The list displays the name of the project, and 3 buttons select, *edit* and *del*. (Figure 9)

3.2.2.1 Select

The first step is to select a project to work on or create a new one if no project has been started yet. By clicking on the select button project information will be loaded on to the system and the user can navigate to various parts of the project using the navbar. At any point, the user can change the project they want to view. Selecting a project gives the list a light blue background so the user can understand which project is currently being worked on. In Figure 9 we can see that the project “Making an Omelette” is selected.

3.2.2.2 Edit

Clicking edit displays a form where a user can alter the name of the project, but again as the previously stated an empty form cannot be submitted. If the user changes their mind and feels that altering the name is not necessary anymore, they can just press cancel to dismiss any changes before submitting the form. (Figure 11)

3.2.2.3 Del

Deleting a project is a sensitive operation this is because all other information about the project will also be lost and cannot be retrieved. So, when the user presses the del button a confirmation box pops up and asks the user if they are sure that they want to delete the project. This is done to prevent users from accidentally deleting a project

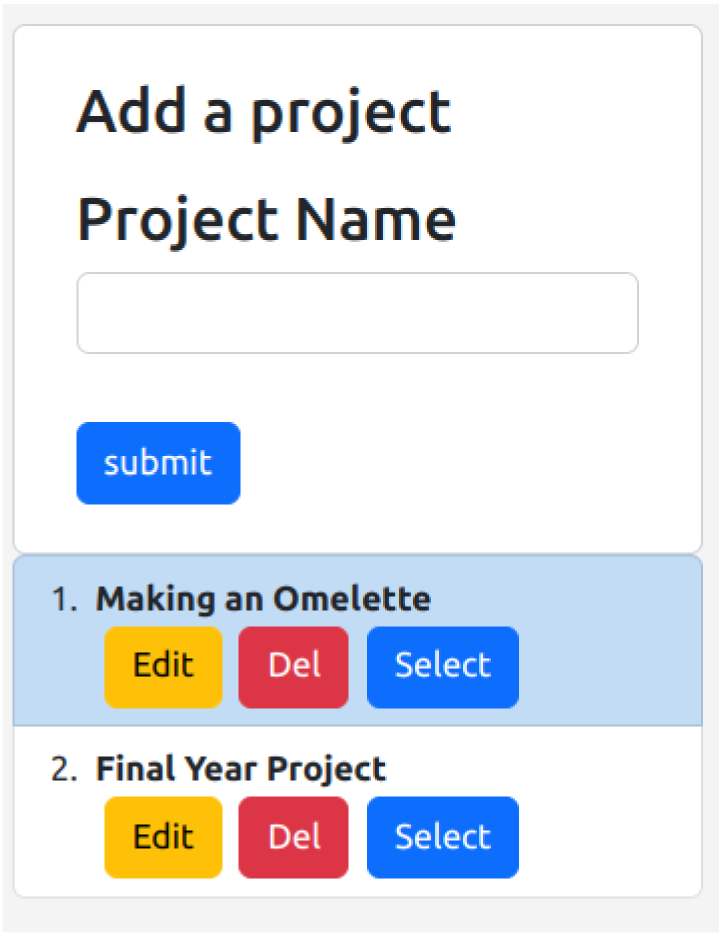


Figure 9. Project

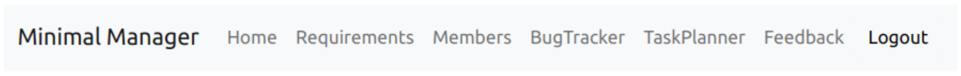


Figure 10. Navbar

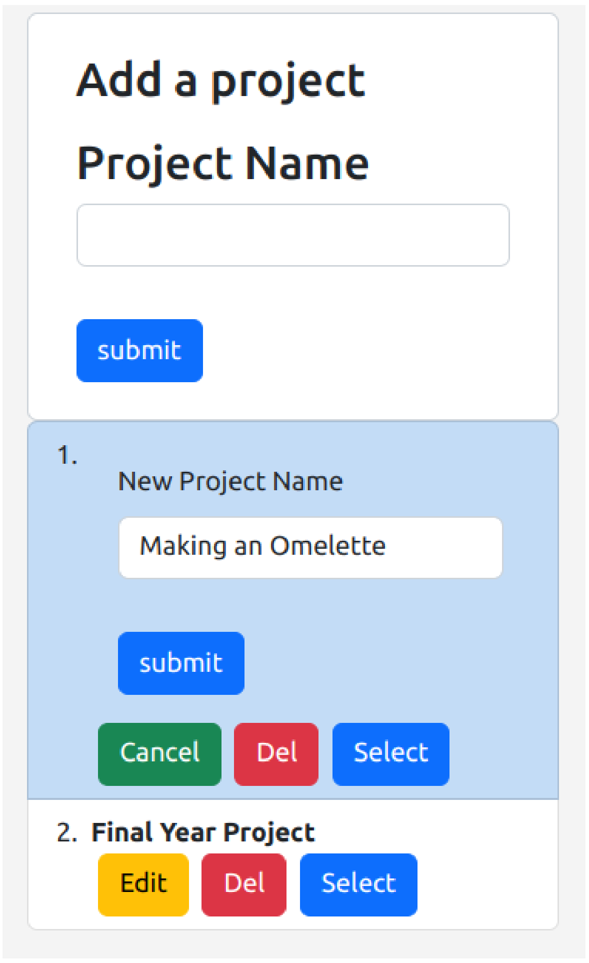


Figure 11. Editing the project name for "Making an Omelette"

3.3 Members

Minimal Manager allows multiple users to collaborate on a project. If the user wants to add more people to the project being worked on, they can click Members where a form is presented to them. Here the user must enter the username of the person they wish to add to the project. The user must exist on the system otherwise a pop-up box will show up on the screen to alert the user that the entered username is not registered in the system. The user that is logged in is already added as a member of the project. In Figure 12 we see 2 users. As previously stated Luffy is the name of the user who is currently using the system. He has added Zorro and Sanji to collaborate with him on the project.

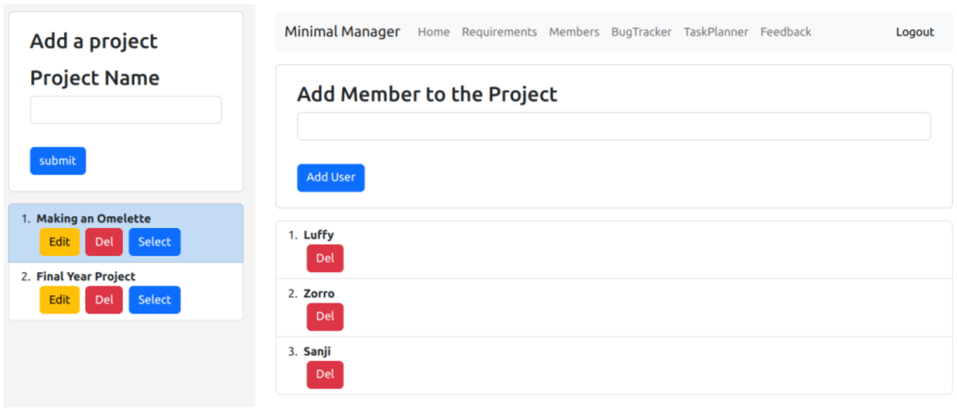


Figure 12. Members

3.4 Requirements

Like projects requirements has 2 parts the form and the component. However, it 4 distinct types of inputs that are displayed on the form and the list. There are also 7 additional buttons the functions of which shall be explained here

3.4.1 The form

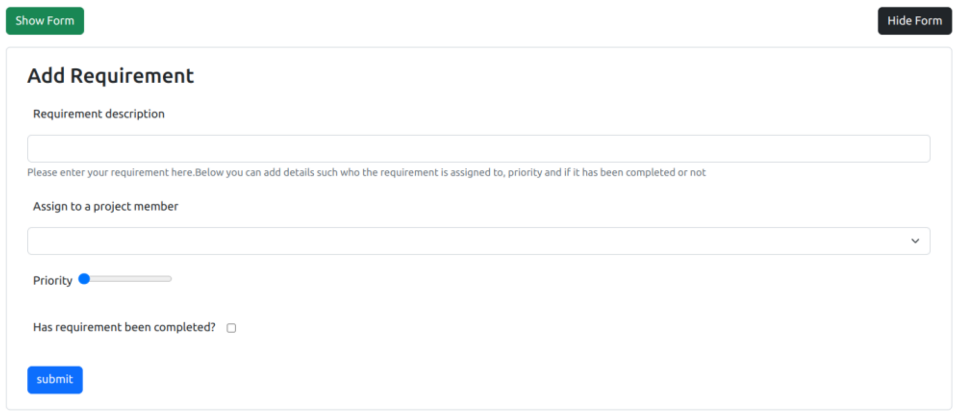
The form takes in four inputs:

Requirement description is a brief description of small tasks that needs to be done by a member to complete the project.

Assign to a project member is a field where only members who are added to the project are shown in a dropdown list when clicked. Clicking on a name assigns them the requirement to them.

Priority sets the value of how important the requirement is from 1 to 5. A higher value indicating a higher priority

Has the requirement been completed field is checkbox that asks the user if the requirement has been completed. Clicking the box colours the list green to indicate that it has been completed.



Add Requirement

Requirement description

Please enter your requirement here.Below you can add details such who the requirement is assigned to, priority and if it has been completed or not

Assign to a project member

Priority

Has requirement been completed?

Figure 13. Requirement form with buttons to show or hide it

3.4.2 The list

The list displays items that were added to the form. These items can be edited by clicking the edit button and deleted by clicking the del button.

3.4.3 The buttons

Show Form and **Hide Form** Buttons are added to either show the form or to hide it, respectively.

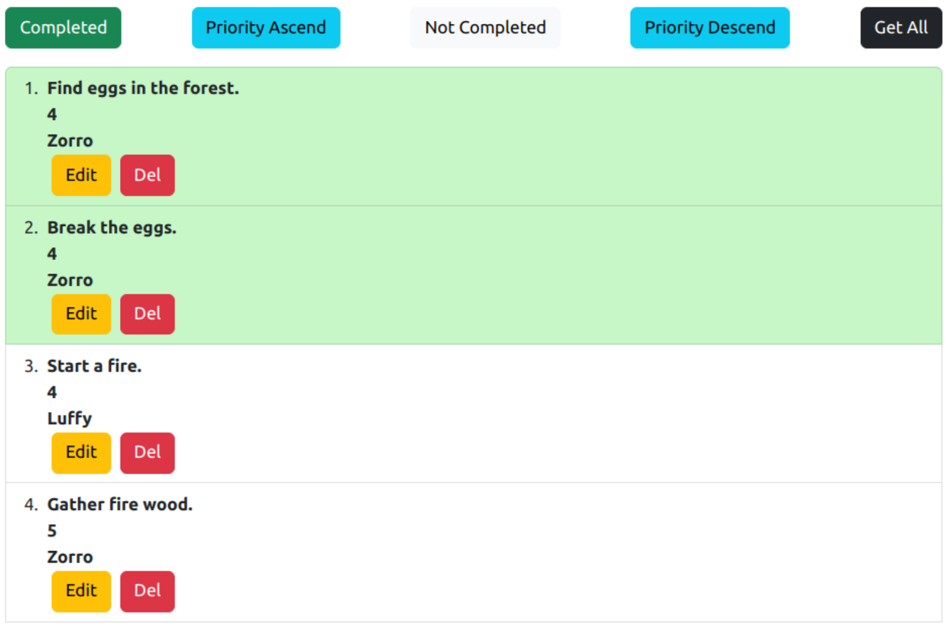
Completed button shows only requirements which have been marked as completed.

Not Completed button shows only requirements which have not been marked as completed.

Priority Ascend Sorts the items in ascending order.

Priority Descend Sorts the items in descending order.

The buttons will do lasting change to the list of items. For example, if completed is pressed followed by not completed then nothing will be shown. To get back the original list we need to press **Get All.**

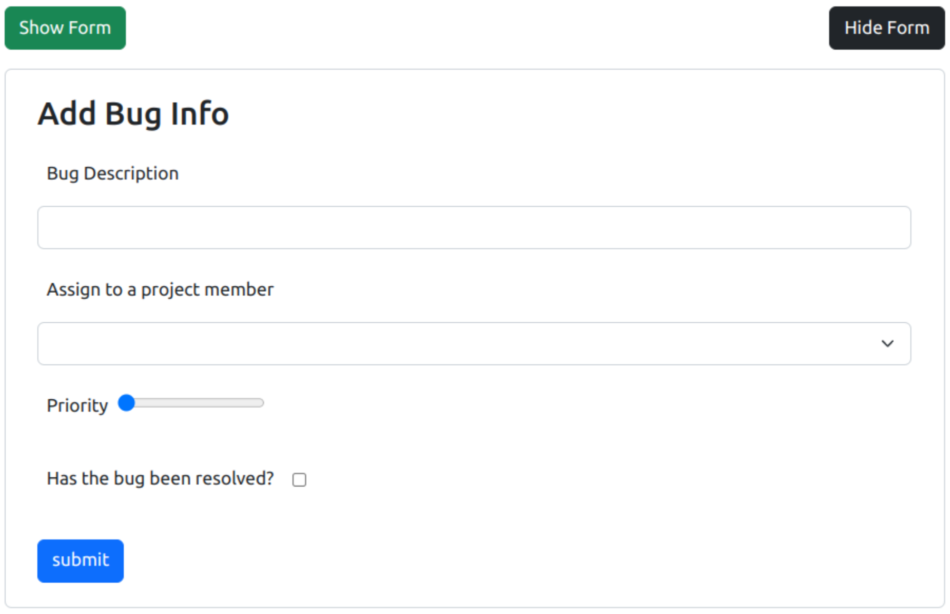


1. Find eggs in the forest.
   4
   Zorro
   Edit Del
2. Break the eggs.
   4
   Zorro
   Edit Del
3. Start a fire.
   4
   Luffy
   Edit Del
4. Gather fire wood.
   5
   Zorro
   Edit Del

Figure 14. Requirement list for project making an omelette with options to sort them

3.5 Bug tracker

This part of the project is like Requirements as it has the same type of form, buttons and list with different names. However, it has been kept separate as it serves different purposes which are to record/report bugs that come up in a system, track who is responsible for resolving them, check their severity and whether it has been solved.



Add Bug Info

Bug Description

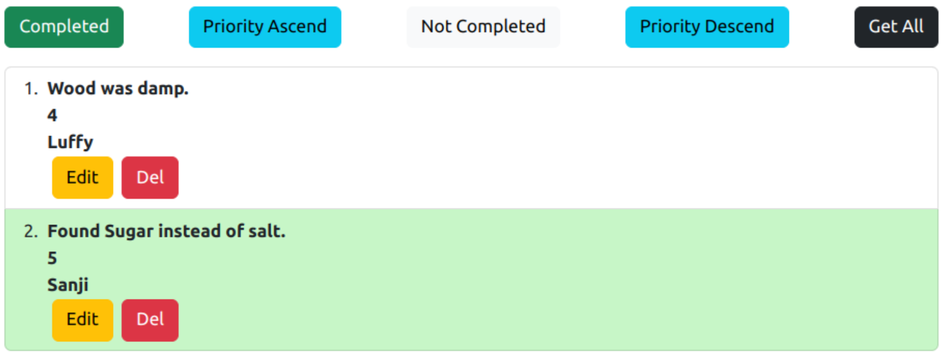
Assign to a project member

Priority

Has the bug been resolved?

submit

Figure 15. Bug tracker form

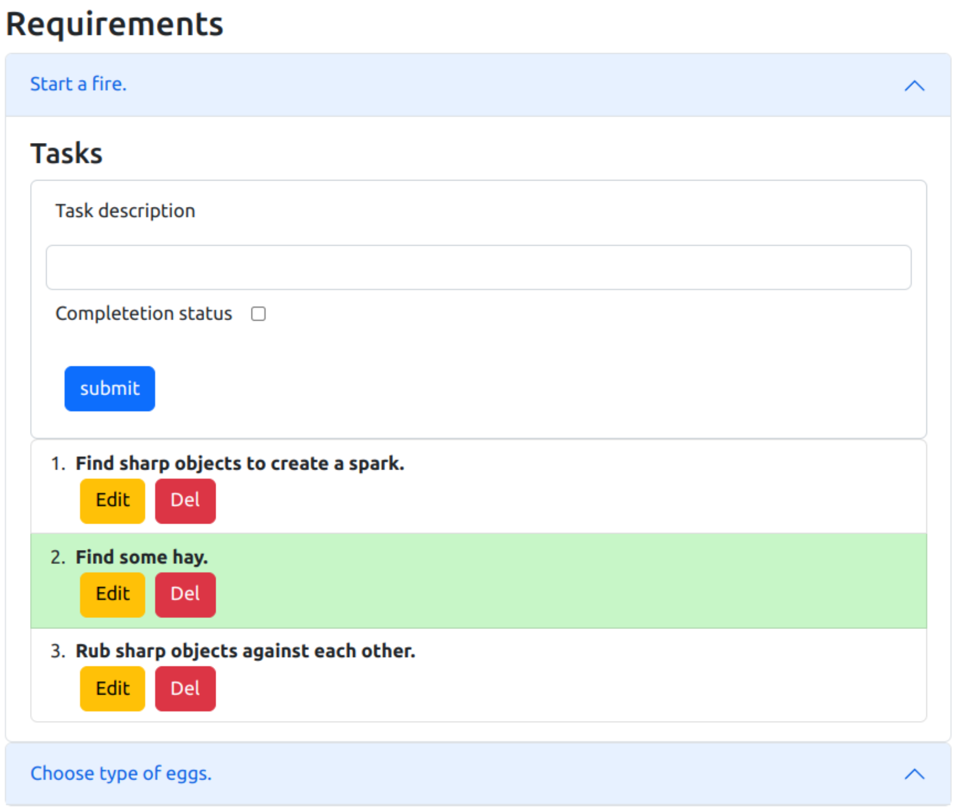


1. Wood was damp.
   4
   Luffy
   Edit Del
2. Found Sugar instead of salt.
   5
   Sanji
   Edit Del

Figure 16. List of all bugs for project making an omelette

3.6 TaskPlanner

In TaskPlanner a user has a list all the requirements that has been assigned to them as an accordion list (a list with items that have a dropdown button). The main purpose behind this feature is so that a requirement that has been assigned to them is broken down into more manageable parts and complete them one by one. Clicking on an accordion item displays a form with 2 fields, one to describe the task and the other to record its current completion status. As with BugTracker and Requirements once a task has been marked as completed it will appear on the list as completed.


Requirements

Start a fire.

Tasks

Task description

Completetion status ☐

submit

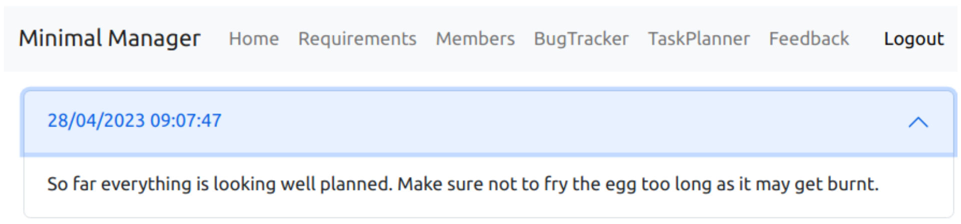
1. Find sharp objects to create a spark.
   Edit Del
2. Find some hay.
   Edit Del
3. Rub sharp objects against each other.
   Edit Del

Choose type of eggs.

Figure 17. TaskPlanner showing only two tasks for user Luffy

3.7 Feedback

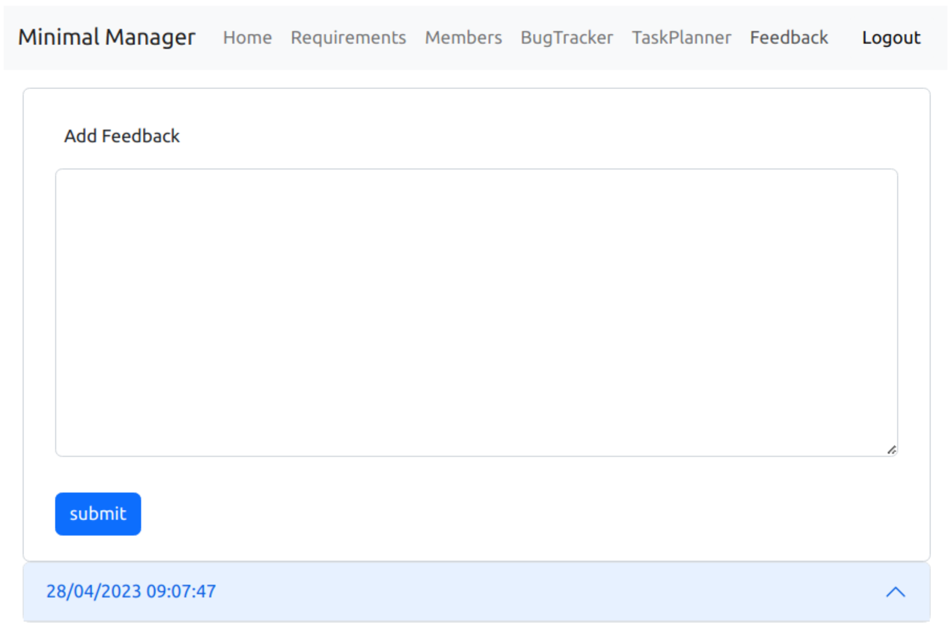
Finally, we look at the last feature of minimal manager, which is feedback. Here, the content is different for the two types of users. Educators have the access to a form where they can provide feedback for the students. Once submitted these are then displayed below as accordion items along with the date and time, they were provided in. The educator will have the option to edit or delete the feedback as well if necessary. The student can only view the feedback. They can then reflect on these and add more requirements to the project to improve it further.



28/04/2023 09:07:47

So far everything is looking well planned. Make sure not to fry the egg too long as it may get burnt.

Figure 18. Feedback for Luffy only shows the feedback



28/04/2023 09:07:47

Figure 19. Feedback for Zeff showing the form, buttons, and feedback

Chapter 4: **Guide for developers**

The code for minimal manager has been separated into 2 parts the client and the server. The client has been built using Vue (Vue documentation) a framework for JavaScript and the server is an application processing interface built using Django (django documentation) which is a framework for python.

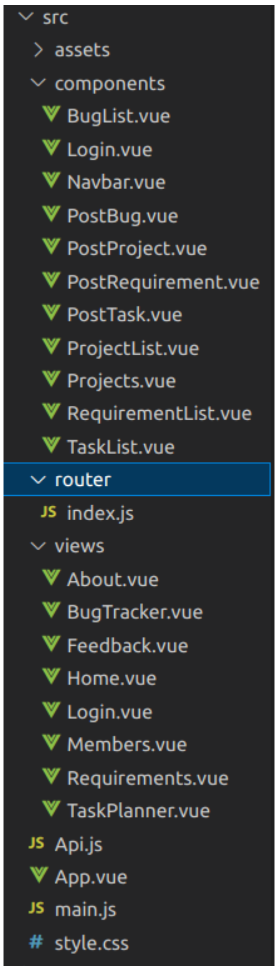
4.1 Client –side: Vue

In minimal manager the client and the server are running on different domains. The client sends requests to the server and receives a response. The four main types of requests that have been made use of to build the client are post, get, put, and delete which allows users to create, read, update, and delete items respectively, CRUD for short. For each of these requests, we must create separate JavaScript functions.

The Vue project was set up using Vite (Vite documentation). It is a build tool that helps set up a Vue project by providing a faster development server and more efficient build process than the traditional setup, which makes the use of web-packs. For this project, the name was frontend which can be found in the path minimal_manager/api/frontend. The 2 most important directories here are src and node_modules. The node_modules directory holds all the dependencies for the project such as e third-party packages, Vue itself and other libraries or plugins that are installed.

4.1.1 src

This is the main source directory (Figure 20). We define our request functions and various components of the client here. This is what we are mostly concerned with in this section. In src we have **index.html** and **main.js which** serve as the entry points of the application. The file index.html is where the main Vue.js file **App.vue** is located.

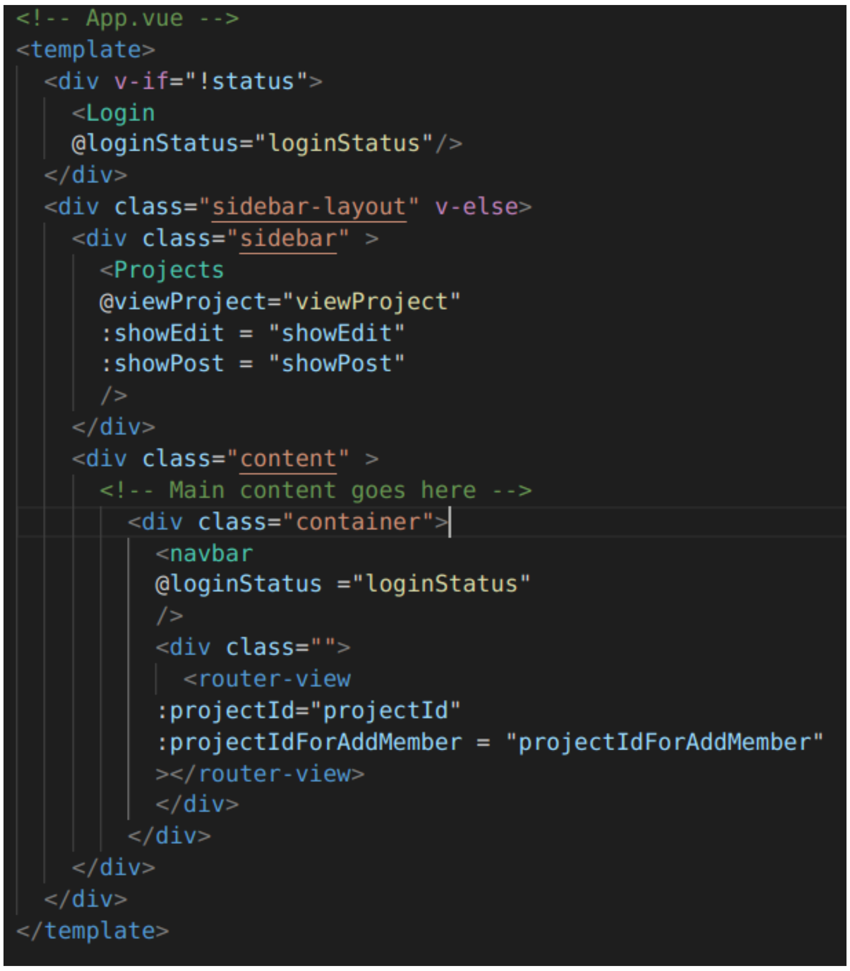


```
src
  assets
  components
    BugList.vue
    Login.vue
    Navbar.vue
    PostBug.vue
    PostProject.vue
    PostRequirement.vue
    PostTask.vue
    ProjectList.vue
    Projects.vue
    RequirementList.vue
    TaskList.vue
  router
    index.js
  views
    About.vue
    BugTracker.vue
    Feedback.vue
    Home.vue
    Login.vue
    Members.vue
    Requirements.vue
    TaskPlanner.vue
  Api.js
  App.vue
  main.js
  style.css
```

Figure 20. Files within src directory

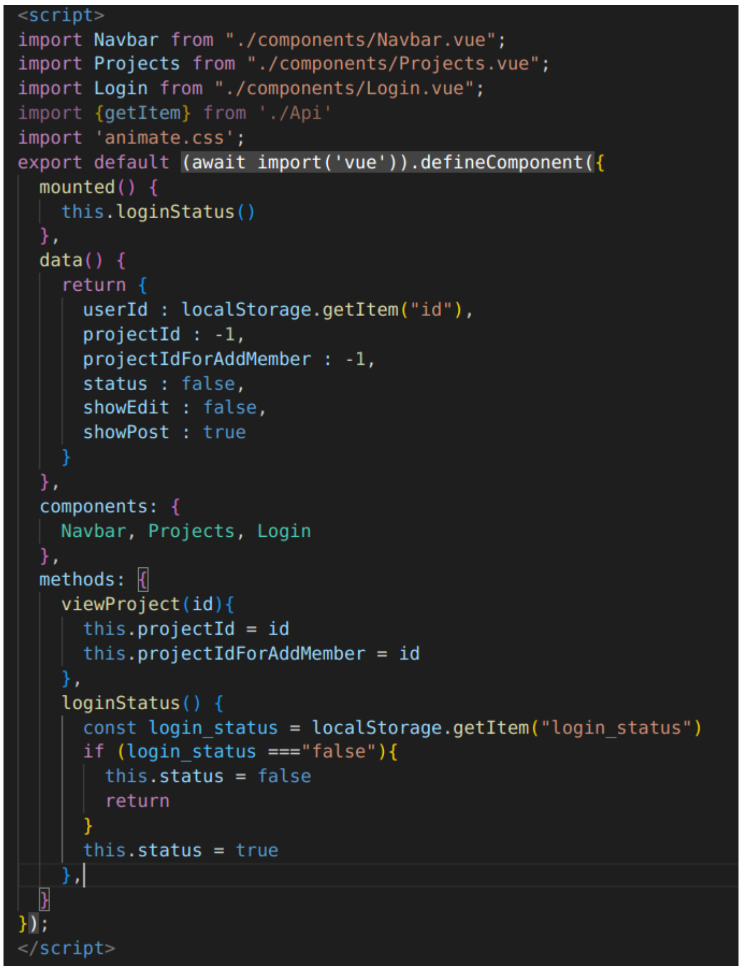
4.1.2 App.vue

The App.vue (Figures 21 and 22) file loads all the components from the components directory that make up minimal manager. Components are reusable and modular pieces of code that encapsulate HTML, CSS, and JavaScript code. A basic Vue component consists of a template tag and a script tag. The html is put within the template tags and the javascript functions are put within the script tag.



```
<!-- App.vue -->
<template>
  <div v-if="!status">
    <Login
    @loginStatus="loginStatus"/>
  </div>
  <div class="sidebar-layout" v-else>
    <div class="sidebar" >
      <Projects
      @viewProject="viewProject"
      :showEdit = "showEdit"
      :showPost = "showPost"
      />
    </div>
    <div class="content" >
      <!-- Main content goes here -->
        <div class="container">
          <navbar
          @loginStatus ="loginStatus"
          />
          <div class="">
            <router-view
          :projectId="projectId"
          :projectIdForAddMember = "projectIdForAddMember"
          ></router-view>
          </div>
        </div>
    </div>
  </div>
</template>
```

Figure 21. Template tags of App.vue



```
<script>
import Navbar from "./components/Navbar.vue";
import Projects from "./components/Projects.vue";
import Login from "./components/Login.vue";
import {getItem} from './Api'
import 'animate.css';
export default (await import('vue')).defineComponent({
  mounted() {
    this.loginStatus()
  },
  data() {
    return {
      userId : localStorage.getItem("id"),
      projectId : -1,
      projectIdForAddMember : -1,
      status : false,
      showEdit : false,
      showPost : true
    }
  },
  components: {
    Navbar, Projects, Login
  },
  methods: {
    viewProject(id){
      this.projectId = id
      this.projectIdForAddMember = id
    },
    loginStatus() {
      const login_status = localStorage.getItem("login_status")
      if (login_status ==="false"){
        this.status = false
        return
      }
      this.status = true
    },
  }
});
</script>
```

Figure 22. Script tags of App.vue

The tags highlighted in green are the components (Figure 21), they are also registered within the components () property. App acts as a container for all these other components. These components can be divided further into more components. The container component shall be referred to as the **parent component** and the contained as the **child component**.

Within the script tags (Figure 22) we first import the components from the file paths they are saved in. We will look at the different components and the Api.js file later but for now we will try to understand the code that follow. A more extensive understanding of some of these terms can be found in the Vue documentation.

The **export default {}** (Figure 22) is the default export of the module. It encapsulates a single object or component.

Although this App.vue does not make use of all options available to be used within export default we will now look at few that have made use of to build minimal manager.

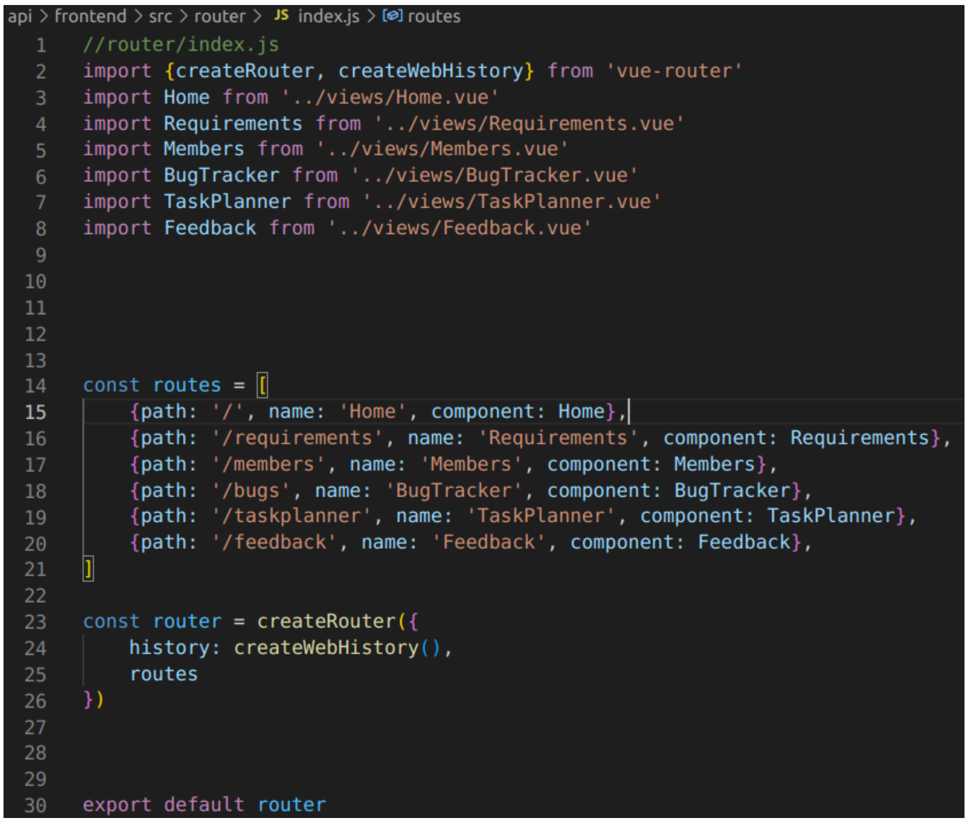
- **mounted ():** Here the functions that have been defined within **methods** are called with the keyword '**this'** so that they are executed when the page is loaded. In Figure 22 displayItems is called.
- **data ():** Here attributes for the object or component to be exported are stored. For example, id attribute gets the value of user id in the local storage of the browser. How the id gets stored in the local storage in the first place will be explained later when we are inspecting the Login component.

- **methods ():** Here the functions for the component are defined
- **watchers ():** Here we can perform custom logic whenever a **data** property changes.
- **created ():** Here the logic that needs to be performed before the component is **mounted** is setup.
- **emits ():** Child components can send back data to or even call a function in the parent component. In the Project tag in Figure 21 we see the line @viewProject=" viewProject". This is a shorthand for binding a custom event named viewProject to a method named viewProject in a Vue component.
- **props ():** Parent components can pass data or a method into a child component. These methods or data are received as props in the child components. Since, App is the main component, it does not have any props passed into it. But it does pass props within the opening tag for router-view, :projectId="projectId". This means that the value of the projectId attribute declared in App within the **data ()** property is passed to the router-view component as projectId.

4.1.3 Router

The router-view tags (Figure 21) in the template tags are especially important. These tags load the files within the 'views' directory (Figure 20). The routes for these files are defined within the index.js file inside the router directory (Figure 23). All the files in views are imported and then assigned a route. This is done by defining the path, name, and component inside an object. This process is repeated for all other views. Each path must have a unique string value. The view home serves as the default path if no other path is mentioned in the browser apart from domain name.

Once the objects have been defined, they are put inside the routes array and then passed into the createRouter object. This is a built-in object for Vue that creates the routes. Once the router has been created, they can be accessed by typing the domain name followed by the route. At this point minimal manager is being run locally so the domain is http://localhost:5173/. This ensures that the router-view tags will load the Home.vue on a user's screen. Adding the path of a route will load the corresponding view.



api > frontend > src > router > JS index.js > routes

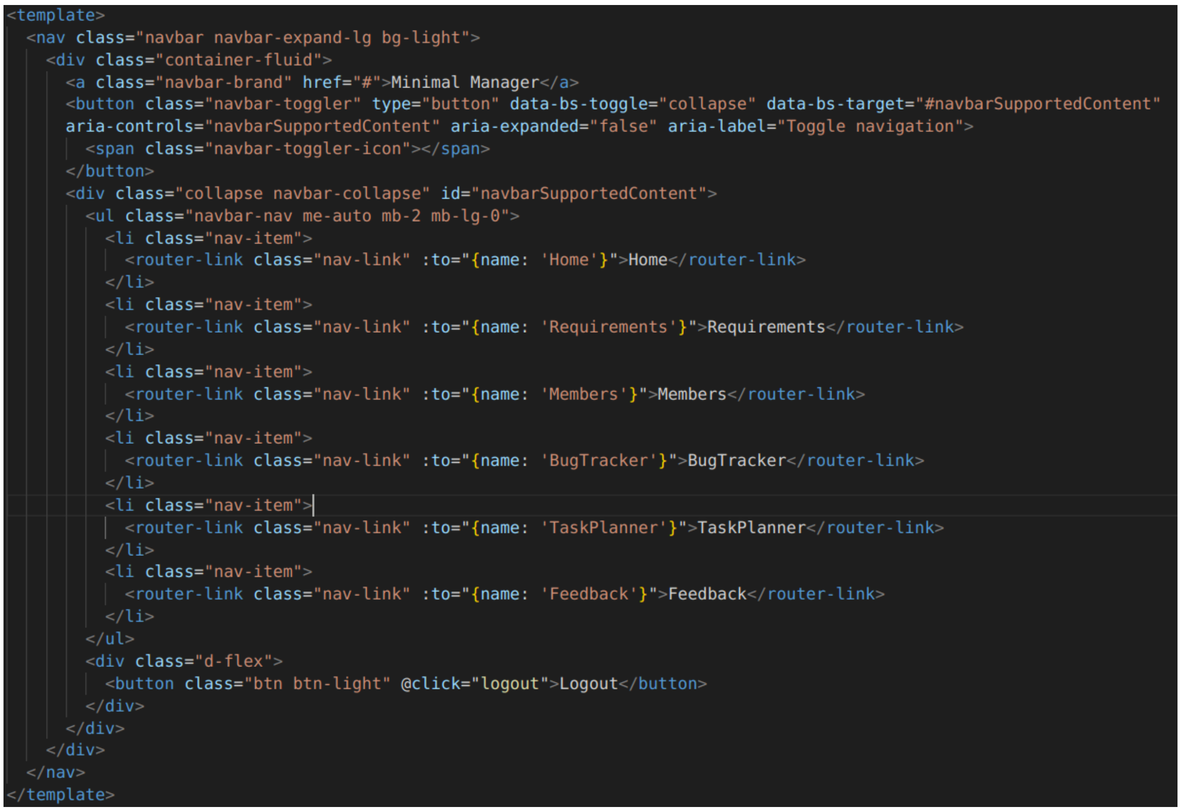
```
//router/index.js
import {createRouter, createWebHistory} from 'vue-router'
import Home from '../views/Home.vue'
import Requirements from '../views/Requirements.vue'
import Members from '../views/Members.vue'
import BugTracker from '../views/BugTracker.vue'
import TaskPlanner from '../views/TaskPlanner.vue'
import Feedback from '../views/Feedback.vue'





const routes = [
    {path: '/', name: 'Home', component: Home},
    {path: '/requirements', name: 'Requirements', component: Requirements},
    {path: '/members', name: 'Members', component: Members},
    {path: '/bugs', name: 'BugTracker', component: BugTracker},
    {path: '/taskplanner', name: 'TaskPlanner', component: TaskPlanner},
    {path: '/feedback', name: 'Feedback', component: Feedback},
]

const router = createRouter({
    history: createWebHistory(),
    routes
})



export default router
```

Figure 23. Path router/index.js

4.1.4 Navbar.vue

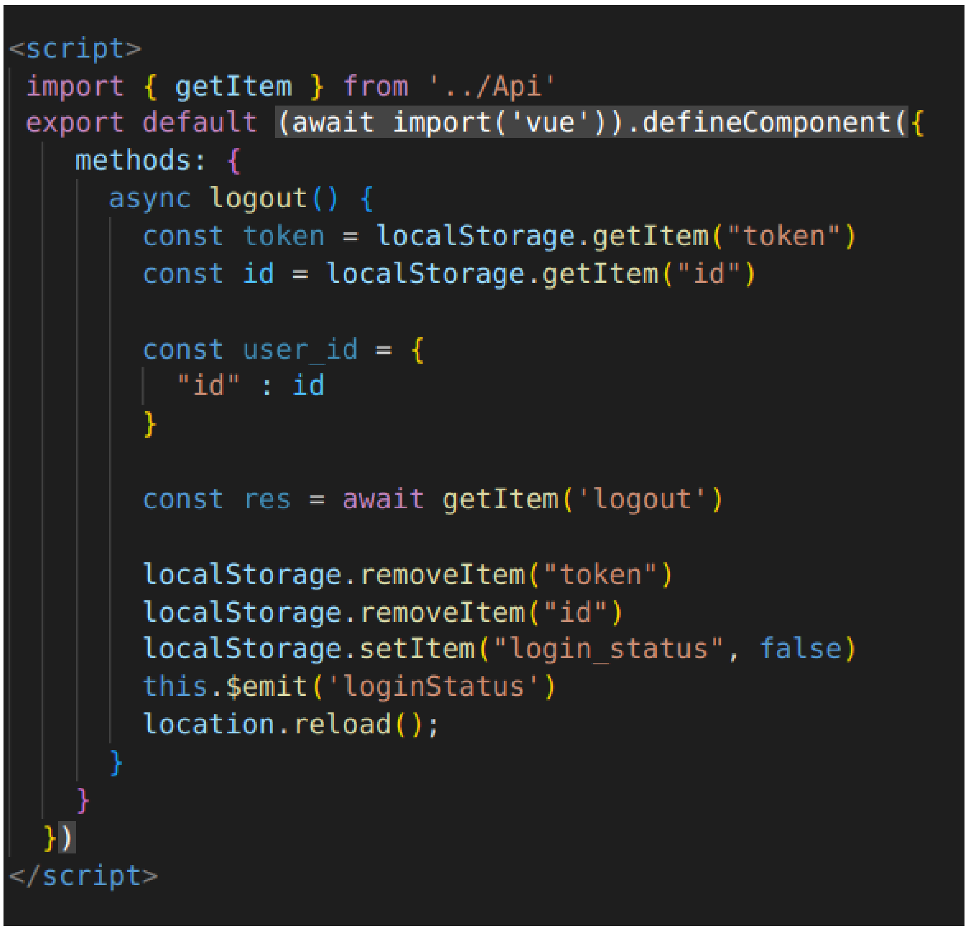
These routes are utilised to create the navigation bar that was shown in Chapter 3. The code for the navigation bar is show in Figure 24. Each list item within the nav tags represent a displayed feature in the navbar shown in Figure.

The logout button which is bound to the logout function is quite important (Figure 25). It sends a get request to the Api and deletes the token required to make requests and the id from the local storage which are set when the user logs in. It also sets the login status to false which tells the App to display the login form as the user has logged out.



```
<template>
  <nav class="navbar navbar-expand-lg bg-light">
    <div class="container-fluid">
      <a class="navbar-brand" href="#">Minimal Manager</a>
      <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent"
      aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
      </button>
      <div class="collapse navbar-collapse" id="navbarSupportedContent">
        <ul class="navbar-nav me-auto mb-2 mb-lg-0">
          <li class="nav-item">
            <router-link class="nav-link" :to="{name: 'Home'}">Home</router-link>
          </li>
          <li class="nav-item">
            <router-link class="nav-link" :to="{name: 'Requirements'}">Requirements</router-link>
          </li>
          <li class="nav-item">
            <router-link class="nav-link" :to="{name: 'Members'}">Members</router-link>
          </li>
          <li class="nav-item">
            <router-link class="nav-link" :to="{name: 'BugTracker'}">BugTracker</router-link>
          </li>
          <li class="nav-item">
            <router-link class="nav-link" :to="{name: 'TaskPlanner'}">TaskPlanner</router-link>
          </li>
          <li class="nav-item">
            <router-link class="nav-link" :to="{name: 'Feedback'}">Feedback</router-link>
          </li>
        </ul>
        <div class="d-flex">
          <button class="btn btn-light" @click="logout">Logout</button>
        </div>
      </div>
    </div>
  </nav>
</template>
```

Figure 24. Navbar that displays the links the views created

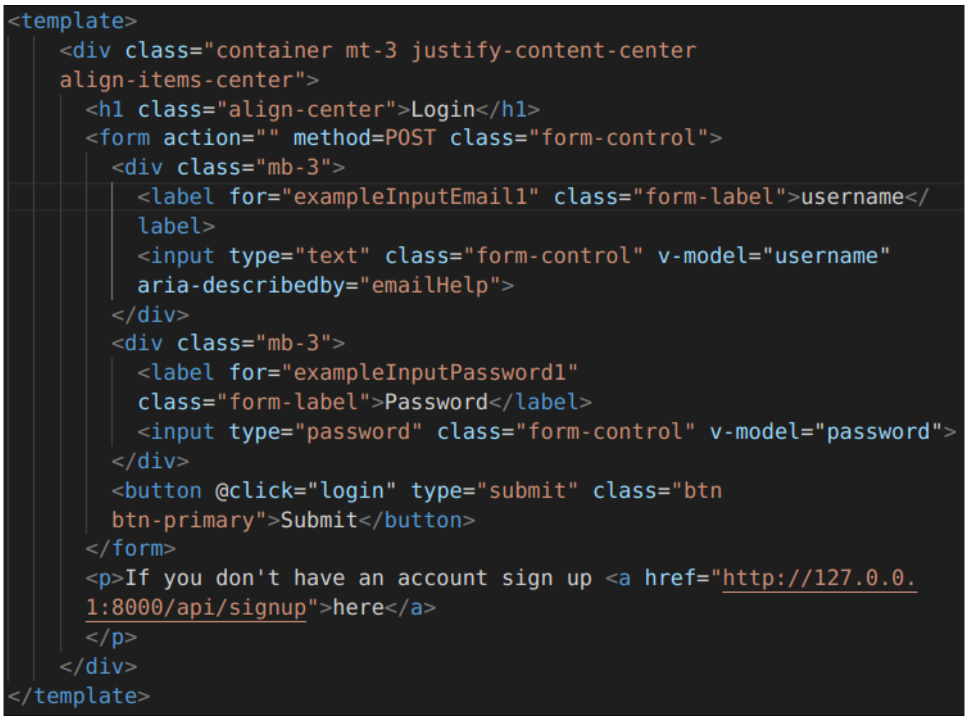


```
<script>
  import { getItem } from '../Api'
  export default (await import('vue')).defineComponent({
    methods: {
      async logout() {
        const token = localStorage.getItem("token")
        const id = localStorage.getItem("id")

        const user_id = {
          "id" : id
        }

        const res = await getItem('logout')

        localStorage.removeItem("token")
        localStorage.removeItem("id")
        localStorage.setItem("login_status", false)
        this.$emit('loginStatus')
        location.reload();
      }
    }
  })
</script>
```

Figure 25. Logout function within the navbar component

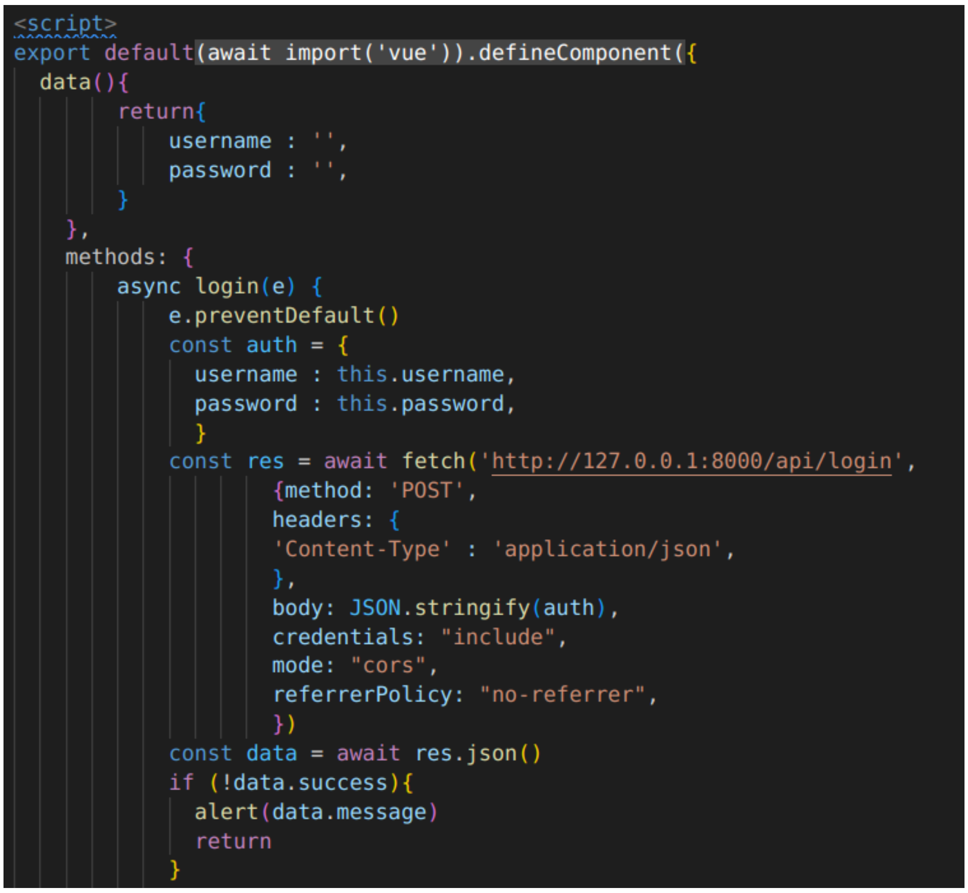
4.1.5 Login

In the Login component we make a post request with the username and password (Figure 26). If they do not match, an alert message (Figure 7) sent by the Api will notify the user that he has not entered the correct username or password. The credentials we saw that were removed in logout in the previous section are set here (Figure 27).

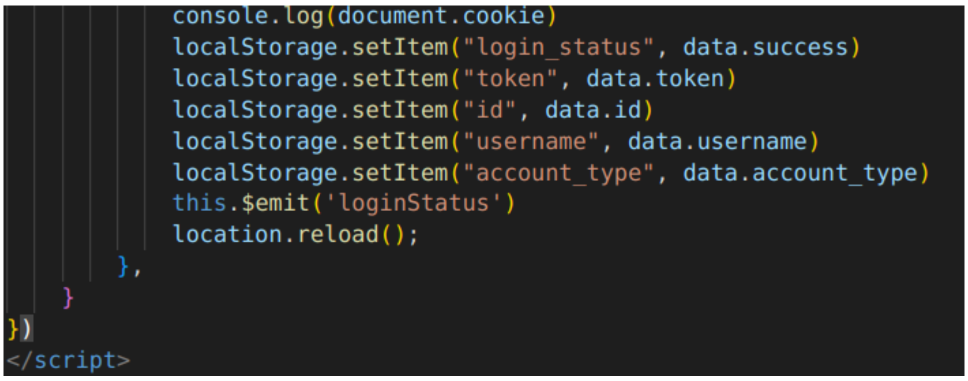


```
<template>
    <div class="container mt-3 justify-content-center
    align-items-center">
      <h1 class="align-center">Login</h1>
      <form action="" method=POST class="form-control">
        <div class="mb-3">
          <label for="exampleInputEmail1" class="form-label">username</
          label>
          <input type="text" class="form-control" v-model="username"
          aria-describedby="emailHelp">
        </div>
        <div class="mb-3">
          <label for="exampleInputPassword1"
          class="form-label">Password</label>
          <input type="password" class="form-control" v-model="password">
        </div>
        <button @click="login" type="submit" class="btn
        btn-primary">Submit</button>
      </form>
      <p>If you don't have an account sign up <a href="http://127.0.0.
      1:8000/api/signup">here</a>
      </p>
    </div>
</template>
```

Figure 26. Login form



```
<script>
export default(await import('vue')).defineComponent({
  data(){
      return{
          username : '',
          password : '',
      }
    },
    methods: {
      async login(e) {
          e.preventDefault()
          const auth = {
            username : this.username,
            password : this.password,
            }
          const res = await fetch('http://127.0.0.1:8000/api/login',
                  {method: 'POST',
                  headers: {
                  'Content-Type' : 'application/json',
                  },
                  body: JSON.stringify(auth),
                  credentials: "include",
                  mode: "cors",
                  referrerPolicy: "no-referrer",
                  })
          const data = await res.json()
          if (!data.success){
            alert(data.message)
            return
          }
```



```
          console.log(document.cookie)
          localStorage.setItem("login_status", data.success)
          localStorage.setItem("token", data.token)
          localStorage.setItem("id", data.id)
          localStorage.setItem("username", data.username)
          localStorage.setItem("account_type", data.account_type)
          this.$emit('loginStatus')
          location.reload();
      },
    }
})
</script>
```

Figure 27. Login Function

4.1.6 Views

Next, we can discuss about the view files. These are components that for which we defined routes for (Figure 23) and load them using the router-view tag (Figure 21). We saw router-view in the App component (Figure 21). In Chapter 3 we saw the various features that work on the Project component. The names of these features correspond to the files in the view directory in Figure. We shall refer to them as views. In minimal manager they all share common functionalities namely:

- Collect data from the user by a form
- Send data back to the API
- Receive data to from the API
- Filter the data and display the desired content such as list information.

We shall explain these four functionalities by investigating Requirements and Api.js file.

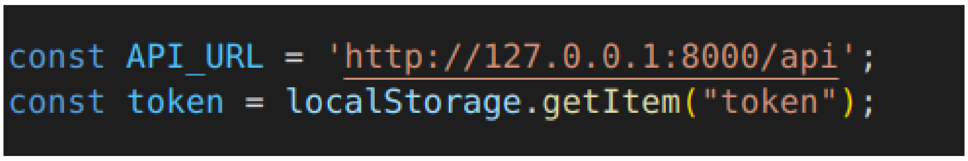
4.1.7 Api.js

The Api.js file contains the functions that send requests to and receives responses from with the API. The API is currently being run on http://127.0.0.1:8000/ locally. The path **api** is added because the name of the Django app is api and all other endpoints are added after this. When the user successfully logs, in a token is returned as a response which is immediately saved in local storage (Figure 26). We need to retrieve this information to make a request. These functions share the following similarities (Figures 29-32).

- All the functions must have the **export** keyword first so that views and components can make use of them.
- The async keyword tells JavaScript that the function is asynchronous one (asynchronous documentation).
- The await keyword tells JavaScript to pause the execution of the function until a promise is resolved (promise documentation). These are objects that represent the eventual completion (or failure) of an asynchronous operation and allow us to work in a more synchronous way.
- They all make use of the **fetch** (fetch documentation) function which is a built-in asynchronous JavaScript function. There is time lag to send and receive requests that is these functions need to be synchronised. All the fetch requests take in a URL and a **Json object** as arguments. The URL is the concatenated string of the domain of the API and the endpoint route to which the request is being made.
- In the **Json object** of the fetch request the first attribute method defines the type of request being made. These are GET, POST, PUT and DELETE. We explained what these functions do in the beginning of this Chapter.
- All the functions must include the type of content being sent and the token received. Without these the request will not be successful as the API cannot determine if the user is authorized without them.
- The importance of mode will be explained in 4.2.3.

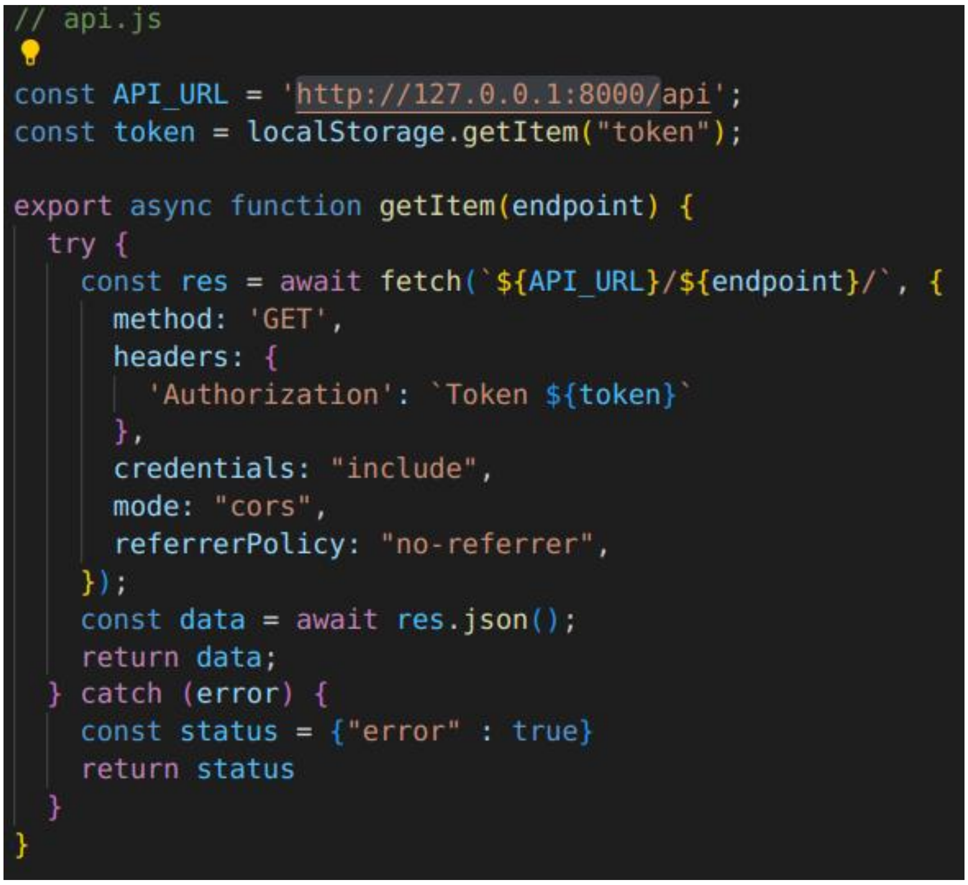
It must also be noted that the functions have differences as well as they serve distinct purposes.

- GET (Figure 29) and DELETE (Figure 31) are like one another. The difference though is that GET method receives data from the API which is filtered and displayed as content on Minimal Manager and DELETE tells the user to destroy a data item on the API and return a response if it was successful.
- PUT (Figure 32) and POST (Figure 30) are like one another but differ from the former pair as they have a body in the **Json object** in the fetch request. These methods send data to be created and updated respectively and this data must be in converted to a string from their Json format so that they can be identified by the API.



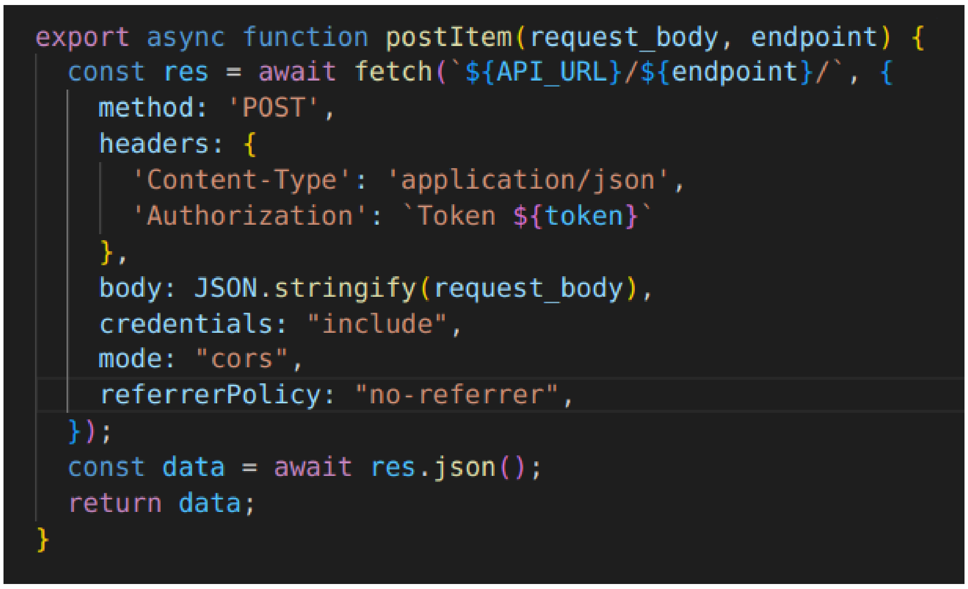
```
const API_URL = 'http://127.0.0.1:8000/api';
const token = localStorage.getItem("token");
```

Figure 28. Setting domain and app name and retrieving token for making requests



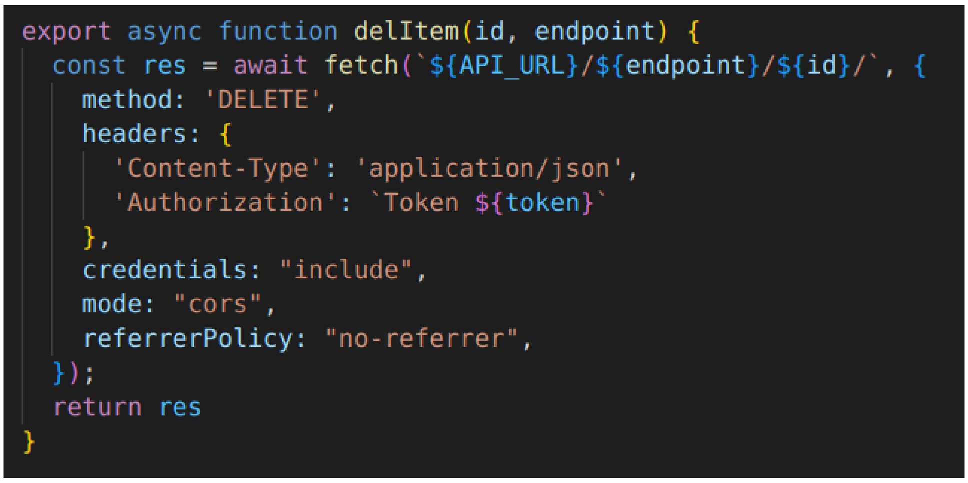
```
// api.js

const API_URL = 'http://127.0.0.1:8000/api';
const token = localStorage.getItem("token");

export async function getItem(endpoint) {
  try {
    const res = await fetch(`${API_URL}/${endpoint}/`, {
      method: 'GET',
      headers: {
        'Authorization': `Token ${token}`
      },
      credentials: "include",
      mode: "cors",
      referrerPolicy: "no-referrer",
    });
    const data = await res.json();
    return data;
  } catch (error) {
    const status = {"error" : true}
    return status
  }
}
```

Figure 29. The function getItem makes get requests



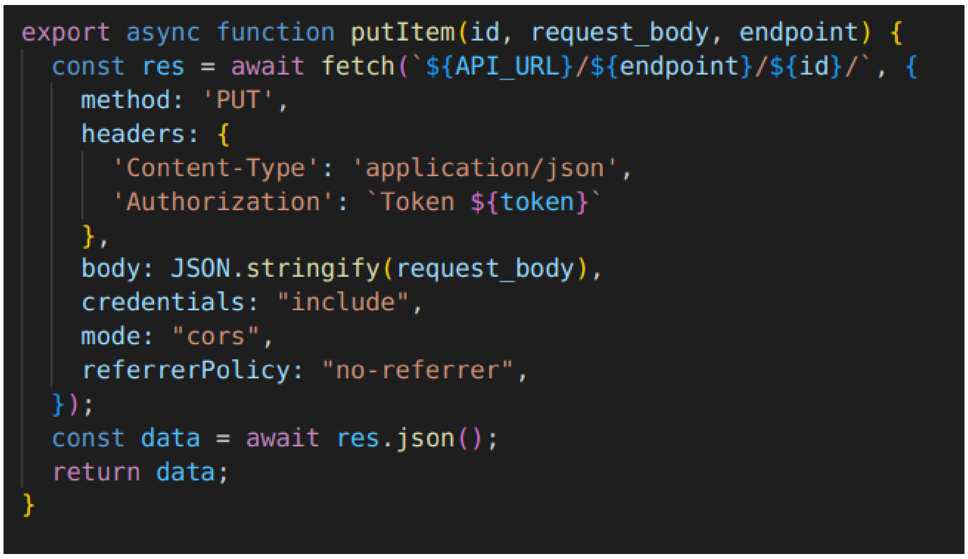
```
export async function postItem(request_body, endpoint) {
  const res = await fetch(`${API_URL}/${endpoint}/`, {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'Authorization': `Token ${token}`
    },
    body: JSON.stringify(request_body),
    credentials: "include",
    mode: "cors",
    referrerPolicy: "no-referrer",
  });
  const data = await res.json();
  return data;
}
```

Figure 30. The function postItem makes post requests



```
export async function delItem(id, endpoint) {
  const res = await fetch(`${API_URL}/${endpoint}/${id}/`, {
    method: 'DELETE',
    headers: {
      'Content-Type': 'application/json',
      'Authorization': `Token ${token}`
    },
    credentials: "include",
    mode: "cors",
    referrerPolicy: "no-referrer",
  });
  return res
}
```

Figure 31. The function delItem makes delete requests



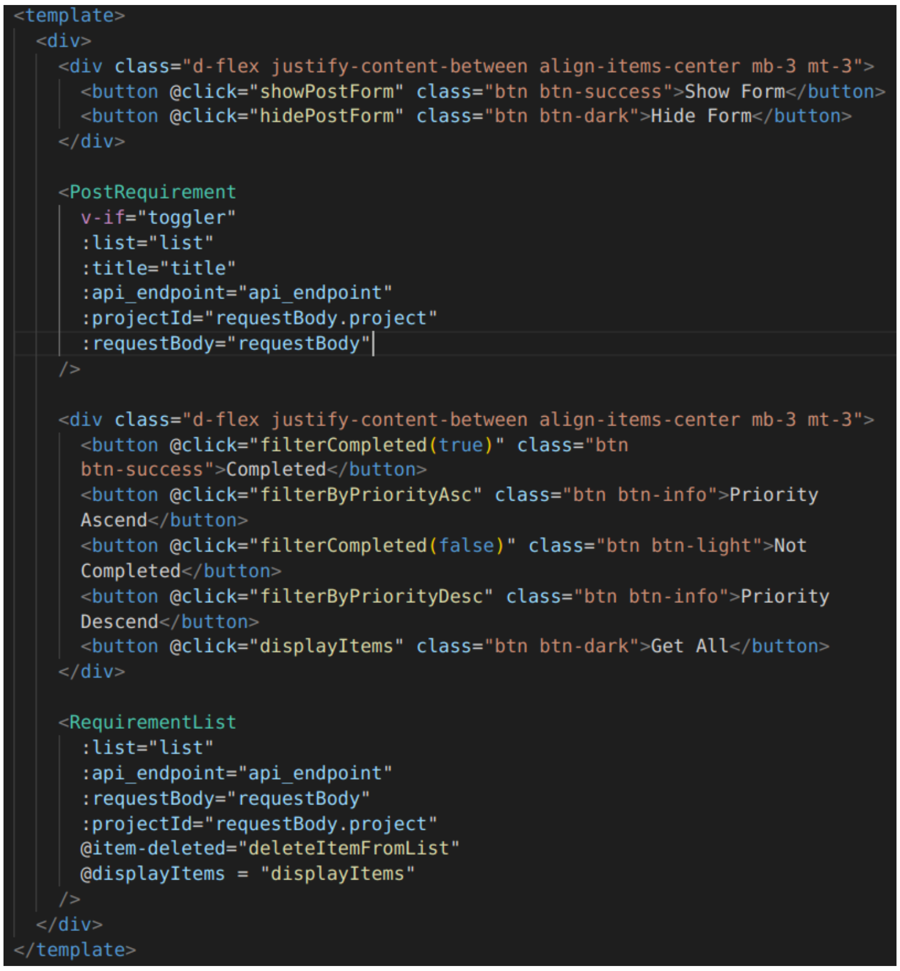
```
export async function putItem(id, request_body, endpoint) {
  const res = await fetch(`${API_URL}/${endpoint}/${id}/`, {
    method: 'PUT',
    headers: {
      'Content-Type': 'application/json',
      'Authorization': `Token ${token}`
    },
    body: JSON.stringify(request_body),
    credentials: "include",
    mode: "cors",
    referrerPolicy: "no-referrer",
  });
  const data = await res.json();
  return data;
}
```

Figure 32. The function putItem makes put requests

4.1.8 Requirements

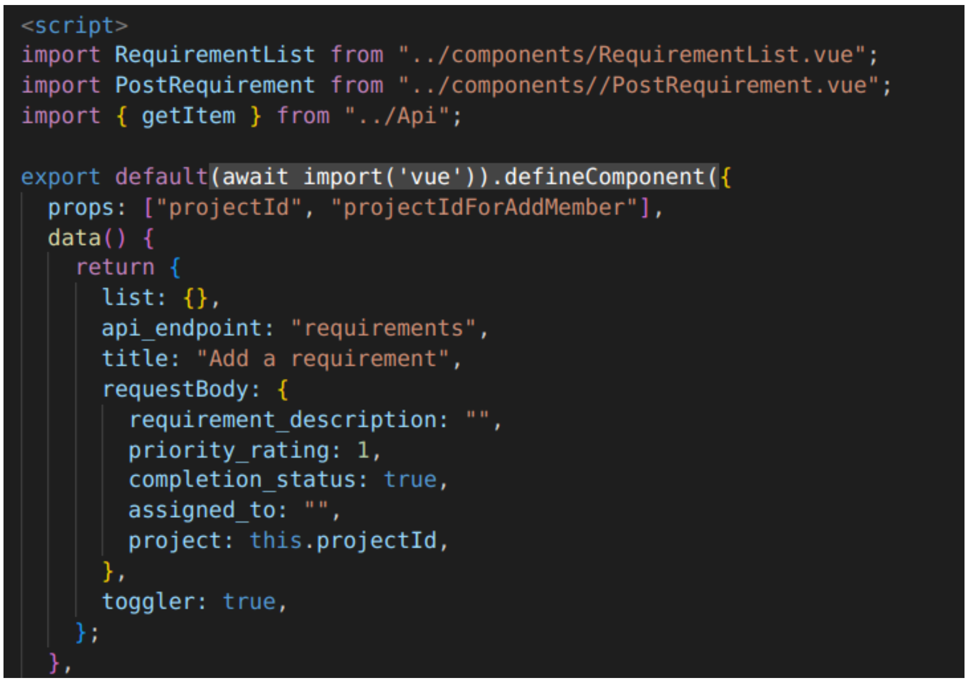
As discussed, earlier Requirements (Figures 33-36) view has two other components RequirementList (Figures 37-42) and PostRequirement (Figures 43-46). Here we first declare the list object to store the Requirement object we shall receive from the API, the API endpoint where it can be found, the title for the form, the request body to be used in the post and put request and a toggler to indicate whether to show or hide the form (Figure 34).

The items are received by calling the getItem function from Api.js (Figure 29). Each button is binded with @click (Figure 33) to the corresponding name of the function such as sorting the items in the list by priority or filtering the completed Requirements. When a new project is selected it automatically detects it in watch (Figure 36) and makes a new request to get requirements for the new object.



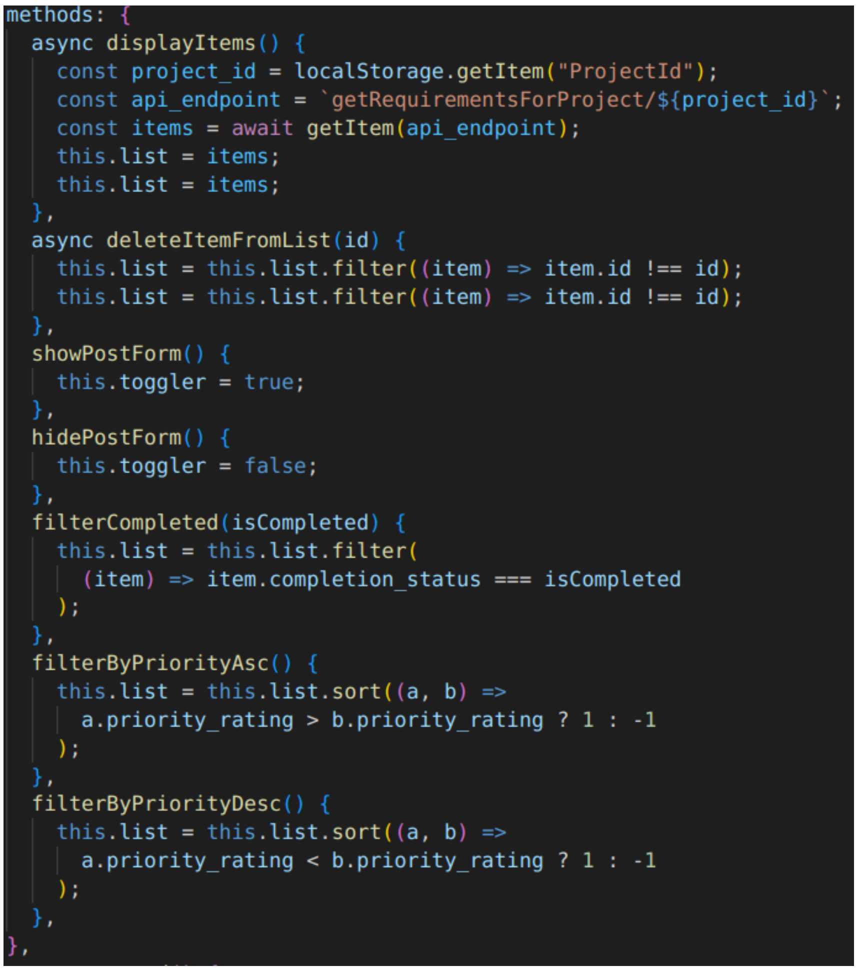
```
<template>
  <div>
    <div class="d-flex justify-content-between align-items-center mb-3 mt-3">
      <button @click="showPostForm" class="btn btn-success">Show Form</button>
      <button @click="hidePostForm" class="btn btn-dark">Hide Form</button>
    </div>

    <PostRequirement
      v-if="toggler"
      :list="list"
      :title="title"
      :api_endpoint="api_endpoint"
      :projectId="requestBody.project"
      :requestBody="requestBody"
    />

    <div class="d-flex justify-content-between align-items-center mb-3 mt-3">
      <button @click="filterCompleted(true)" class="btn
      btn-success">Completed</button>
      <button @click="filterByPriorityAsc" class="btn btn-info">Priority
      Ascend</button>
      <button @click="filterCompleted(false)" class="btn btn-light">Not
      Completed</button>
      <button @click="filterByPriorityDesc" class="btn btn-info">Priority
      Descend</button>
      <button @click="displayItems" class="btn btn-dark">Get All</button>
    </div>

    <RequirementList
      :list="list"
      :api_endpoint="api_endpoint"
      :requestBody="requestBody"
      :projectId="requestBody.project"
      @item-deleted="deleteItemFromList"
      @displayItems = "displayItems"
    />
  </div>
</template>
```

Figure 33. Template of Requirements.vue



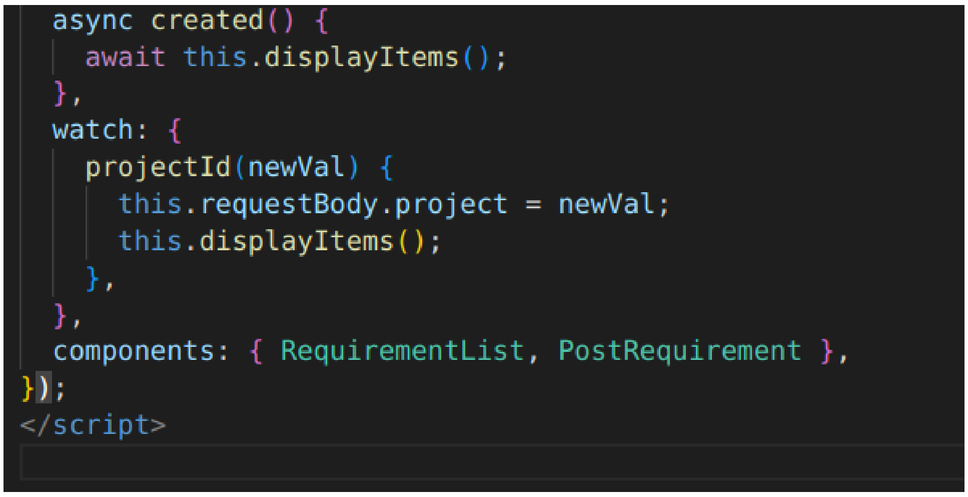
```
<script>
import RequirementList from "../components/RequirementList.vue";
import PostRequirement from "../components//PostRequirement.vue";
import { getItem } from "../Api";

export default(await import('vue')).defineComponent({
  props: ["projectId", "projectIdForAddMember"],
  data() {
    return {
      list: {},
      api_endpoint: "requirements",
      title: "Add a requirement",
      requestBody: {
        requirement_description: "",
        priority_rating: 1,
        completion_status: true,
        assigned_to: "",
        project: this.projectId,
      },
      toggler: true,
    };
  },
```

Figure 34. Script of Requirements.vue



```
methods: {
  async displayItems() {
    const project_id = localStorage.getItem("ProjectId");
    const api_endpoint = `getRequirementsForProject/${project_id}`;
    const items = await getItem(api_endpoint);
    this.list = items;
    this.list = items;
  },
  async deleteItemFromList(id) {
    this.list = this.list.filter((item) => item.id !== id);
    this.list = this.list.filter((item) => item.id !== id);
  },
  showPostForm() {
    this.toggler = true;
  },
  hidePostForm() {
    this.toggler = false;
  },
  filterCompleted(isCompleted) {
    this.list = this.list.filter(
      (item) => item.completion_status === isCompleted
    );
  },
  filterByPriorityAsc() {
    this.list = this.list.sort((a, b) =>
      a.priority_rating > b.priority_rating ? 1 : -1
    );
  },
  filterByPriorityDesc() {
    this.list = this.list.sort((a, b) =>
      a.priority_rating < b.priority_rating ? 1 : -1
    );
  },
},
```

Figure 35. Script of Requirements.vue



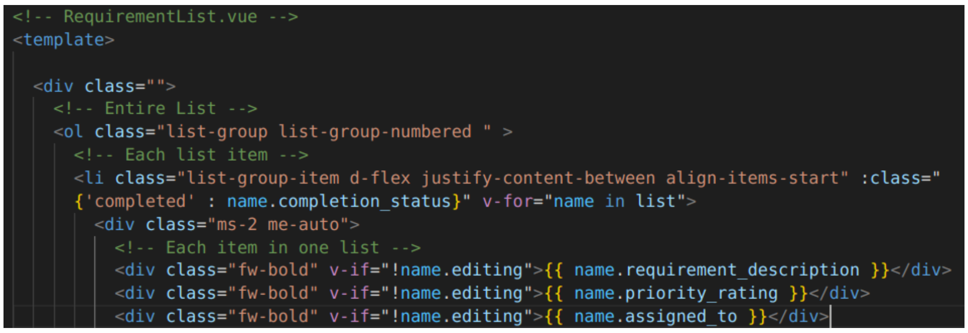
```
  async created() {
    await this.displayItems();
  },
  watch: {
    projectId(newVal) {
      this.requestBody.project = newVal;
      this.displayItems();
    },
  },
  components: { RequirementList, PostRequirement },
});
</script>
```

Figure 36. Script of Requirements.vue

4.1.8.1 RequirementList

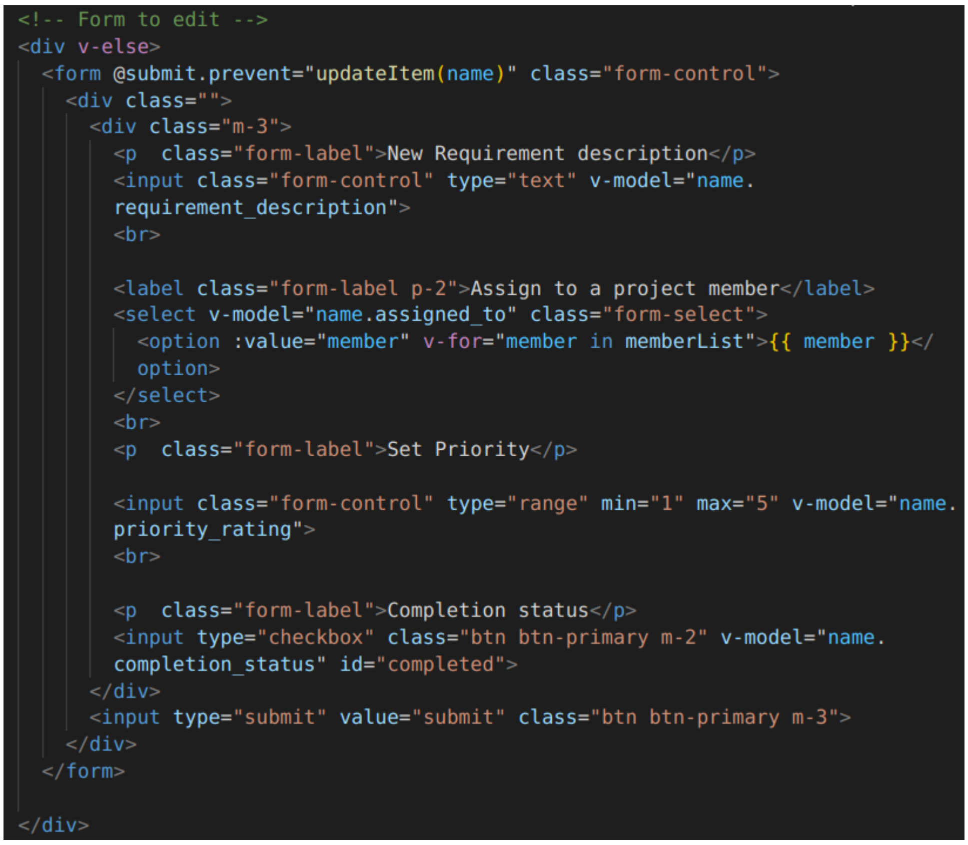
The Requirement received by the client is then feed to the RequirementList component by passing props (Figure 40). There a loop is run through the object and the items to be displayed are extracted from it and displayed in the form of a list (Figure 37). From Api.js the functions getItem (Figure 29), delItem (Figure 31) and putItem (Figure 32) are used here to carry out the features such as showing the assigned users in a project, deleting a requirement, and editing it, respectively (Figure 41).

The other views and the Project component, except Home operate on a similar basis with few intricacies. In summary the items are received after a get request and any changes made by post, put, or delete is adapted to the list and updated to display the results.



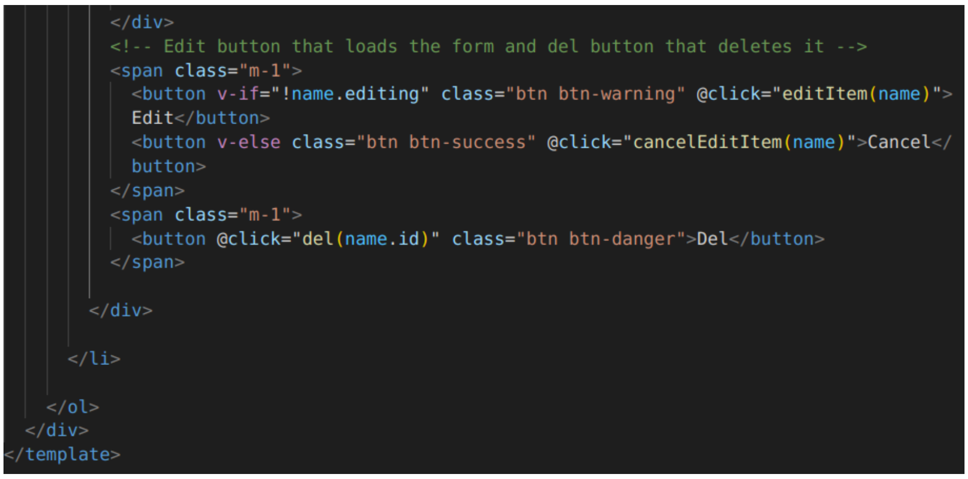
```
<!-- RequirementList.vue -->
<template>

  <div class="">
    <!-- Entire List -->
    <ol class="list-group list-group-numbered " >
      <!-- Each list item -->
      <li class="list-group-item d-flex justify-content-between align-items-start" :class="
      {'completed' : name.completion_status}" v-for="name in list">
        <div class="ms-2 me-auto">
          <!-- Each item in one list -->
          <div class="fw-bold" v-if="!name.editing">{{ name.requirement_description }}</div>
          <div class="fw-bold" v-if="!name.editing">{{ name.priority_rating }}</div>
          <div class="fw-bold" v-if="!name.editing">{{ name.assigned_to }}</div>
```

Figure 37. Template of RequirementList



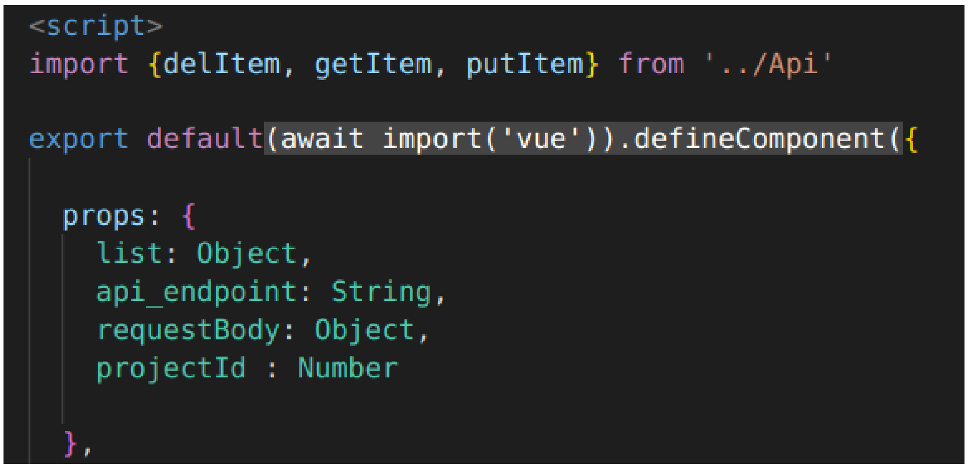
```
<!-- Form to edit -->
<div v-else>
  <form @submit.prevent="updateItem(name)" class="form-control">
    <div class="">
      <div class="m-3">
        <p  class="form-label">New Requirement description</p>
        <input class="form-control" type="text" v-model="name.
        requirement_description">
        <br>

        <label class="form-label p-2">Assign to a project member</label>
        <select v-model="name.assigned_to" class="form-select">
          <option :value="member" v-for="member in memberList">{{ member }}</
          option>
        </select>
        <br>
        <p  class="form-label">Set Priority</p>

        <input class="form-control" type="range" min="1" max="5" v-model="name.
        priority_rating">
        <br>

        <p  class="form-label">Completion status</p>
        <input type="checkbox" class="btn btn-primary m-2" v-model="name.
        completion_status" id="completed">
      </div>
      <input type="submit" value="submit" class="btn btn-primary m-3">
    </div>
  </form>

</div>
```

Figure 38. Template of RequirementList



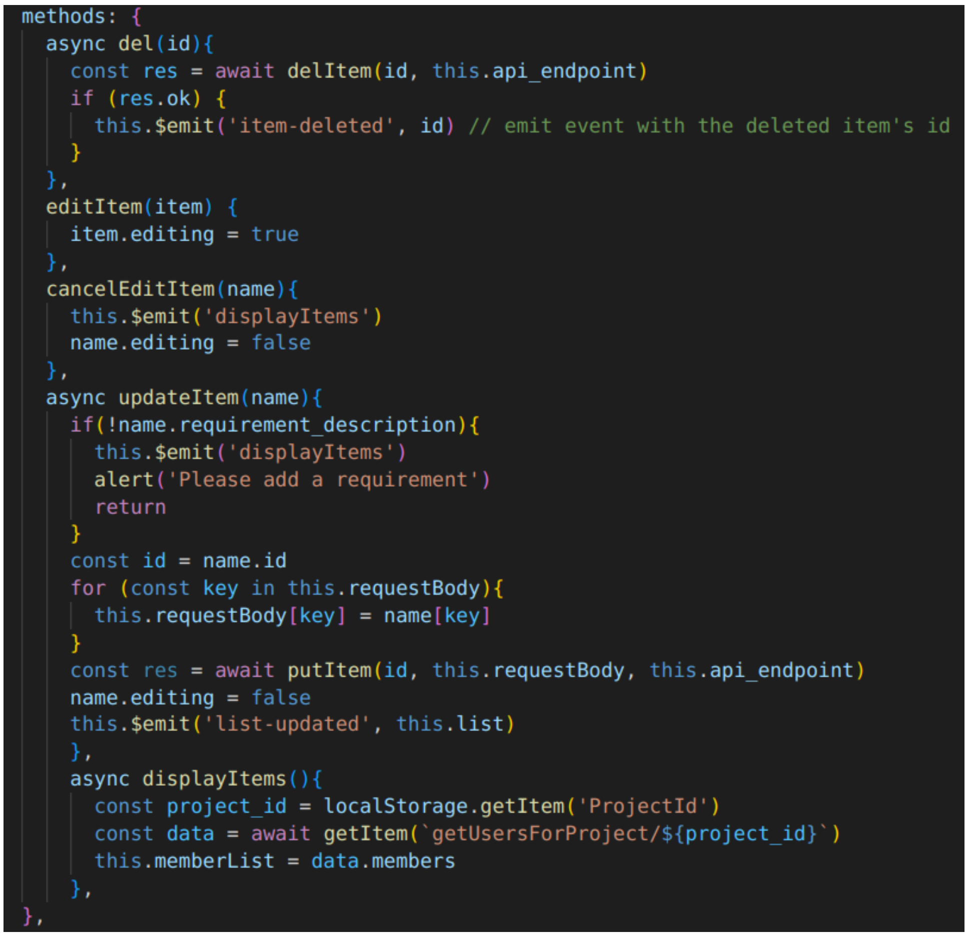
```
          </div>
          <!-- Edit button that loads the form and del button that deletes it -->
          <span class="m-1">
            <button v-if="!name.editing" class="btn btn-warning" @click="editItem(name)">
            Edit</button>
            <button v-else class="btn btn-success" @click="cancelEditItem(name)">Cancel</
            button>
          </span>
          <span class="m-1">
            <button @click="del(name.id)" class="btn btn-danger">Del</button>
          </span>

        </div>

      </li>

    </ol>
  </div>
</template>
```

Figure 39. Template of RequirementList



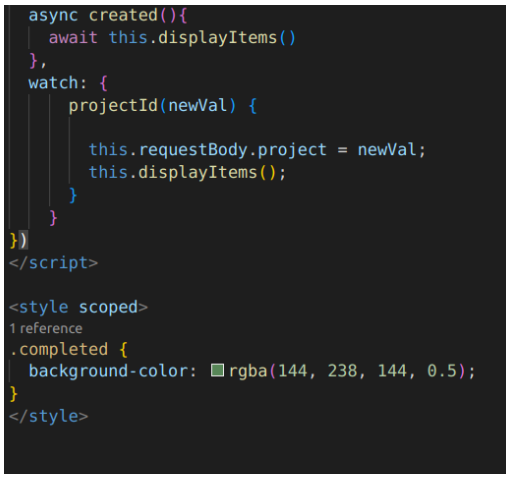
```
<script>
import {delItem, getItem, putItem} from '../Api'

export default(await import('vue')).defineComponent({

  props: {
    list: Object,
    api_endpoint: String,
    requestBody: Object,
    projectId : Number

  },
```

Figure 40. Script of RequirementList



```
methods: {
  async del(id){
    const res = await delItem(id, this.api_endpoint)
    if (res.ok) {
      this.$emit('item-deleted', id) // emit event with the deleted item's id
    }
  },
  editItem(item) {
    item.editing = true
  },
  cancelEditItem(name){
    this.$emit('displayItems')
    name.editing = false
  },
  async updateItem(name){
    if(!name.requirement_description){
      this.$emit('displayItems')
      alert('Please add a requirement')
      return
    }
    const id = name.id
    for (const key in this.requestBody){
      this.requestBody[key] = name[key]
    }
    const res = await putItem(id, this.requestBody, this.api_endpoint)
    name.editing = false
    this.$emit('list-updated', this.list)
    },
    async displayItems(){
      const project_id = localStorage.getItem('ProjectId')
      const data = await getItem(`getUsersForProject/${project_id}`)
      this.memberList = data.members
    },
},
```

Figure 41. Script of RequirementList

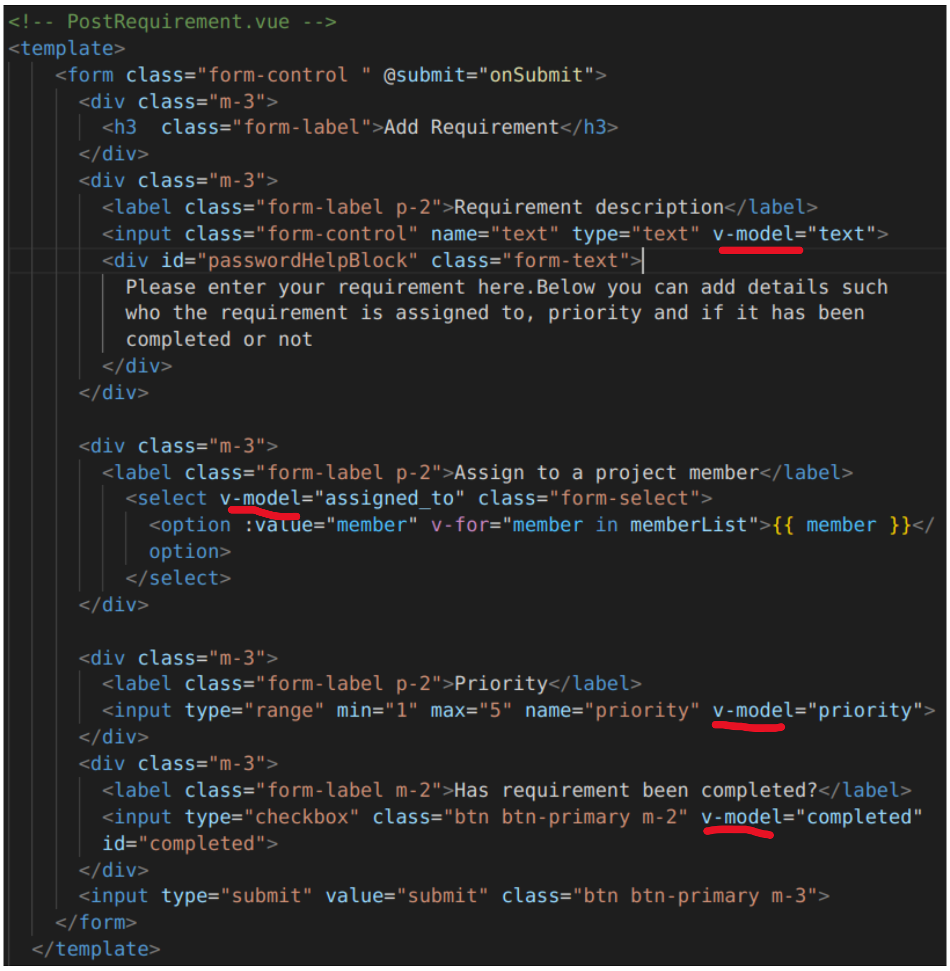


```
  async created(){
    await this.displayItems()
  },
  watch: {
      projectId(newVal) {

        this.requestBody.project = newVal;
        this.displayItems();
      }
    }
})
</script>

<style scoped>
1 reference
.completed {
  background-color: 🟩rgba(144, 238, 144, 0.5);
}
</style>
```

Figure 42. Script of RequirementList

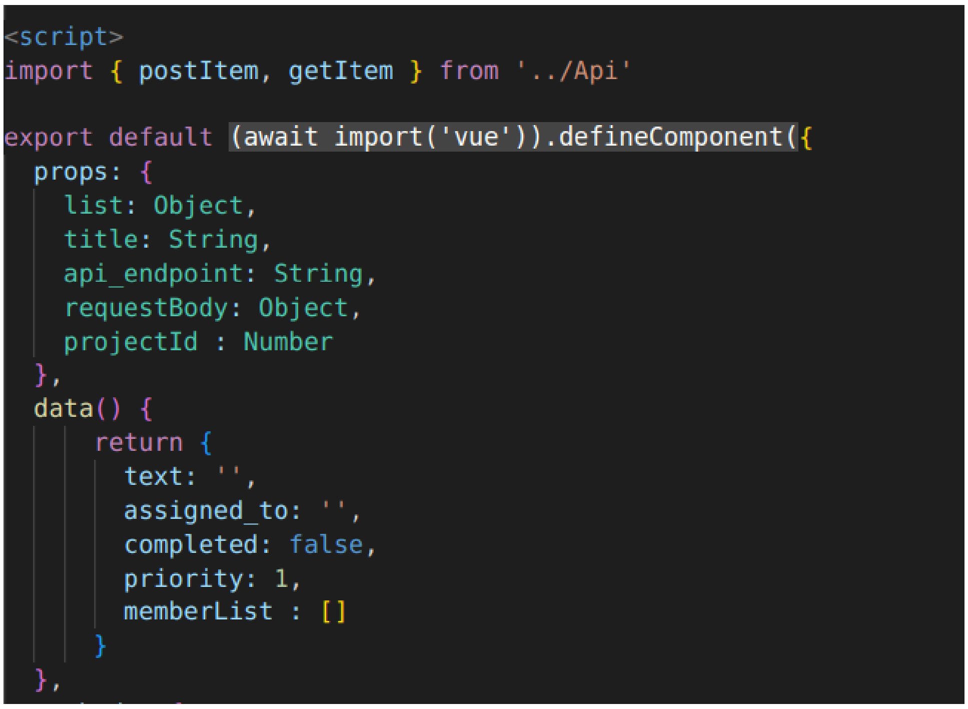
4.1.8.2 PostRequirement

PostRequirement just contains a form that collects all the details such as the ones shown in Chapter 3 Figure using a v-model and a data attribute in the input tags. This is then copied to the requestBody object (Figure 40) being passed as a prop item from Requirement and then sent to the API using the postItem (Figure 30) function from Api.js.



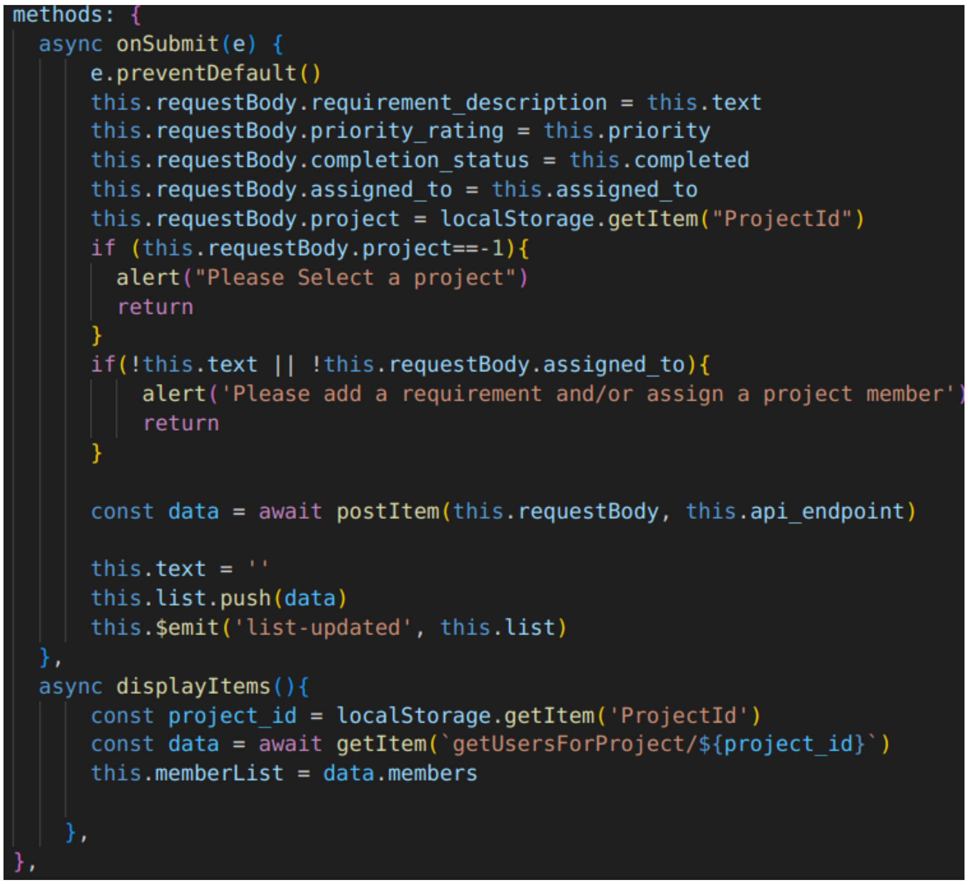
```
<!-- PostRequirement.vue -->
<template>
    <form class="form-control " @submit="onSubmit">
      <div class="m-3">
        <h3  class="form-label">Add Requirement</h3>
      </div>
      <div class="m-3">
        <label class="form-label p-2">Requirement description</label>
        <input class="form-control" name="text" type="text" v-model="text">
        <div id="passwordHelpBlock" class="form-text">
          Please enter your requirement here.Below you can add details such
          who the requirement is assigned to, priority and if it has been
          completed or not
        </div>
      </div>

      <div class="m-3">
        <label class="form-label p-2">Assign to a project member</label>
          <select v-model="assigned_to" class="form-select">
            <option :value="member" v-for="member in memberList">{{ member }}</
            option>
          </select>
      </div>

      <div class="m-3">
        <label class="form-label p-2">Priority</label>
        <input type="range" min="1" max="5" name="priority" v-model="priority">
      </div>
      <div class="m-3">
        <label class="form-label m-2">Has requirement been completed?</label>
        <input type="checkbox" class="btn btn-primary m-2" v-model="completed"
        id="completed">
      </div>
      <input type="submit" value="submit" class="btn btn-primary m-3">
    </form>
  </template>
```

Figure 43. Template of PostRequirement



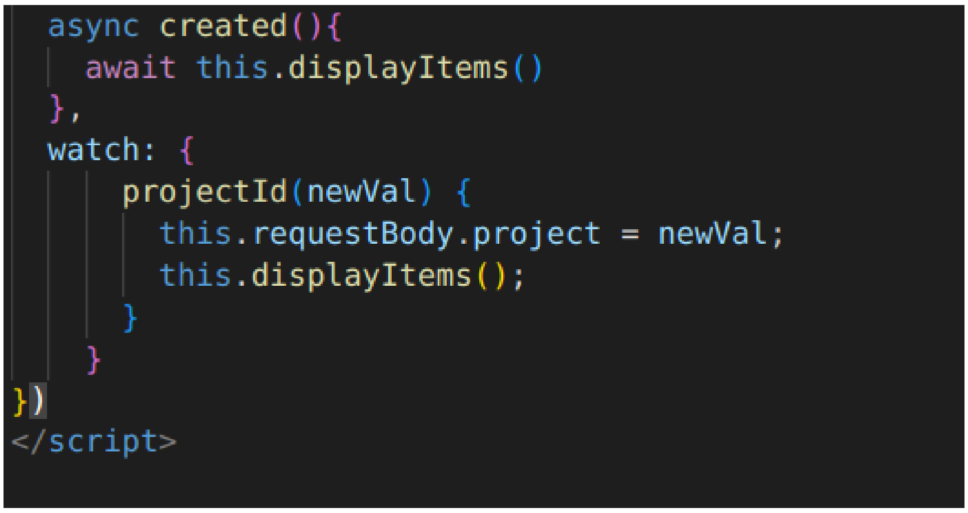
```
<script>
import { postItem, getItem } from '../Api'

export default (await import('vue')).defineComponent({
  props: {
    list: Object,
    title: String,
    api_endpoint: String,
    requestBody: Object,
    projectId : Number
  },
  data() {
      return {
        text: '',
        assigned_to: '',
        completed: false,
        priority: 1,
        memberList : []
      }
  },
```

Figure 44. Script of PostRequirement



```
methods: {
  async onSubmit(e) {
      e.preventDefault()
      this.requestBody.requirement_description = this.text
      this.requestBody.priority_rating = this.priority
      this.requestBody.completion_status = this.completed
      this.requestBody.assigned_to = this.assigned_to
      this.requestBody.project = localStorage.getItem("ProjectId")
      if (this.requestBody.project==-1){
        alert("Please Select a project")
        return
      }
      if(!this.text || !this.requestBody.assigned_to){
          alert('Please add a requirement and/or assign a project member')
          return
      }

      const data = await postItem(this.requestBody, this.api_endpoint)

      this.text = ''
      this.list.push(data)
      this.$emit('list-updated', this.list)
  },
  async displayItems(){
      const project_id = localStorage.getItem('ProjectId')
      const data = await getItem(`getUsersForProject/${project_id}`)
      this.memberList = data.members

    },
},
```

Figure 45. Script of PostRequirement



```
  async created(){
    await this.displayItems()
  },
  watch: {
      projectId(newVal) {
        this.requestBody.project = newVal;
        this.displayItems();
      }
    }
})
</script>
```

Figure 46. Script of PostRequirement

4.2 Sever-side: Django Api

We made several references to the Api in the section 4.1. Here we shall discuss these and most of the important concepts. An Api is a set of protocols and tools that allow different software applications to communicate with each other. According to Roy T. Fielding, an API is like a "contract" between a client and a server that specifies the types of requests and responses that can be made [19]. It allows client applications to request and receive data from a server over the internet which can also include formats, such as JSON or XML.

Django is a high-level Python web framework that provides a powerful toolkit for building web applications quickly and efficiently. One of its key features is the ability to easily create APIs (Application Programming Interfaces) for web-based services.

Django is based on a Model View Controller (MVC) design. These three works together to operate the Api. The model represents the data and logic of the application, the view displays the data to the user, and the controller handles user input and updates the model and view accordingly. The controller is hidden with a layer of abstraction, and for our purposes of explaining Minimal Manager we do not need look at it in detail.

Within our api directory we can see the files named models.py and views.py.

4.2.1 Models

In models (Figures 48-49) we describe our objects. The User model is a subclass of AbstractUser. By inheriting this we can customize the built-in User model by adding or modifying fields. The AbstractUser model provides a basic set of fields and methods for handling user authentication and authorization, such as username, email, password, and permissions.

We have added a type of the user admin, student, and educator. This is important as in Chapter 3 we showed how in feedback that put, post, and delete a feedback can be used by the educator but not the student.

The other models are all subclasses of models.Models and have a distinct set of fields. The requirements class will be familiar as they are the same as the attributes for request body object, we saw in Chapter 4.1. The objects are related to one another by one to many and many to many relationships (Figure 47).

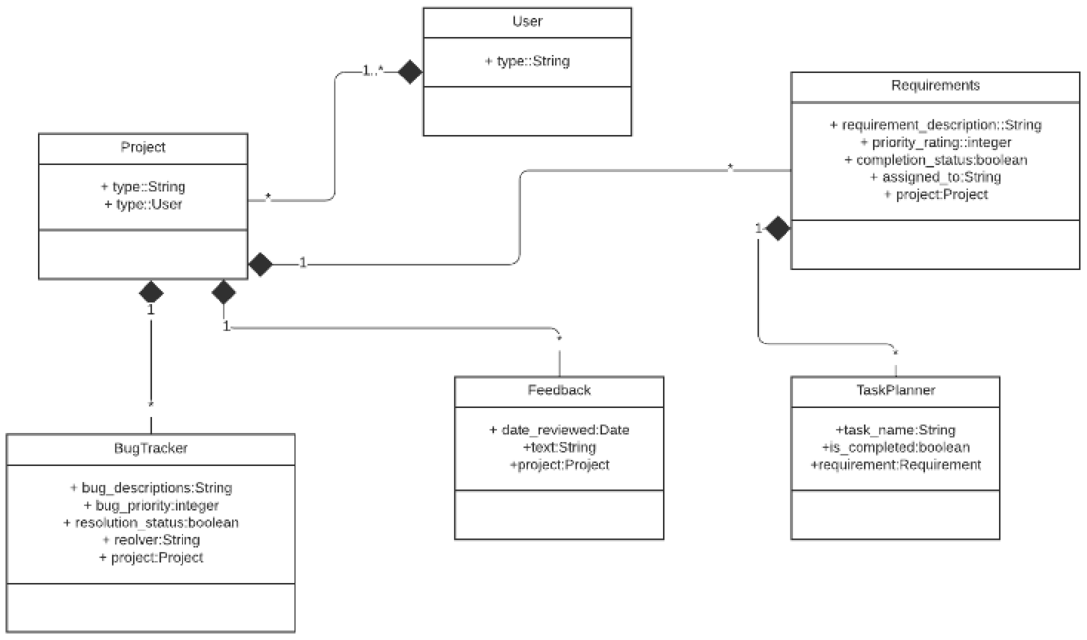
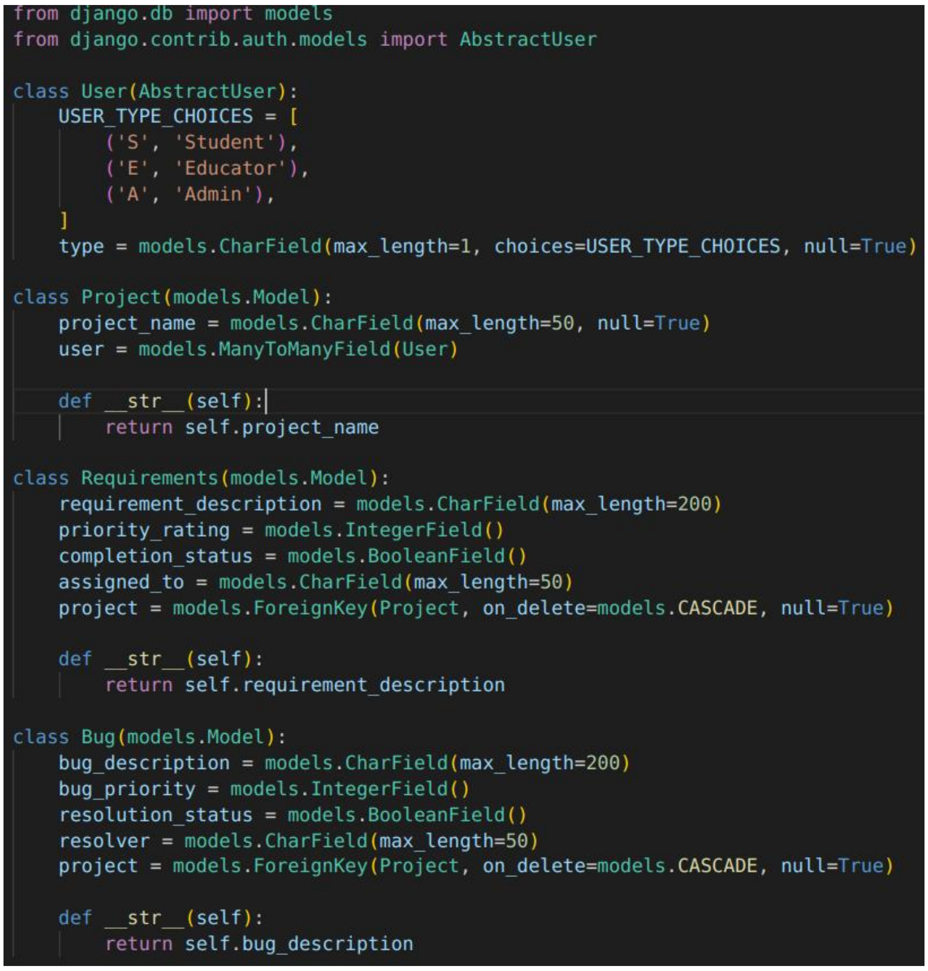


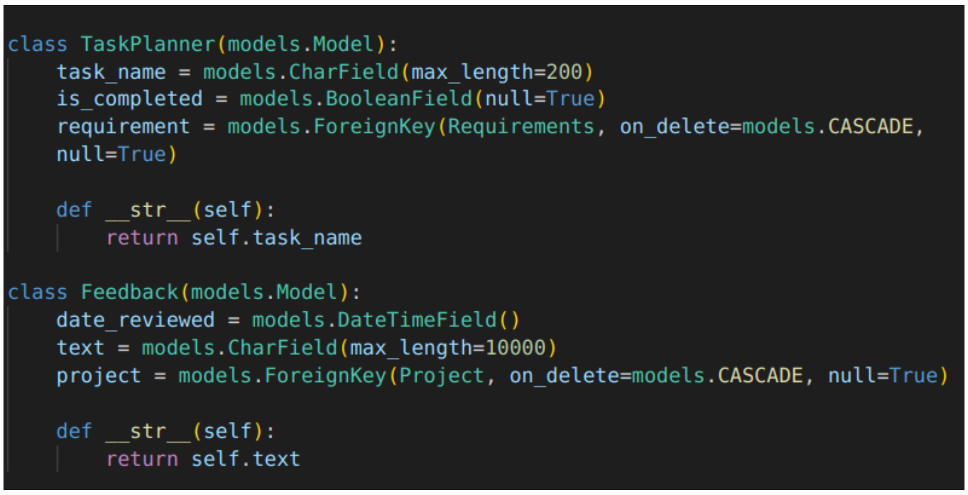
Figure 47. Class DIagram for models

Furthermore, each model is registered in admins.py so the administrator can see all the data except for fields such as passwords, where they can see the encrypted password. (Django models documentation)



```python
from django.db import models
from django.contrib.auth.models import AbstractUser

class User(AbstractUser):
    USER_TYPE_CHOICES = [
        ('S', 'Student'),
        ('E', 'Educator'),
        ('A', 'Admin'),
    ]
    type = models.CharField(max_length=1, choices=USER_TYPE_CHOICES, null=True)

class Project(models.Model):
    project_name = models.CharField(max_length=50, null=True)
    user = models.ManyToManyField(User)

    def __str__(self):
        return self.project_name

class Requirements(models.Model):
    requirement_description = models.CharField(max_length=200)
    priority_rating = models.IntegerField()
    completion_status = models.BooleanField()
    assigned_to = models.CharField(max_length=50)
    project = models.ForeignKey(Project, on_delete=models.CASCADE, null=True)

    def __str__(self):
        return self.requirement_description

class Bug(models.Model):
    bug_description = models.CharField(max_length=200)
    bug_priority = models.IntegerField()
    resolution_status = models.BooleanField()
    resolver = models.CharField(max_length=50)
    project = models.ForeignKey(Project, on_delete=models.CASCADE, null=True)

    def __str__(self):
        return self.bug_description
```

Figure 48. File models.py

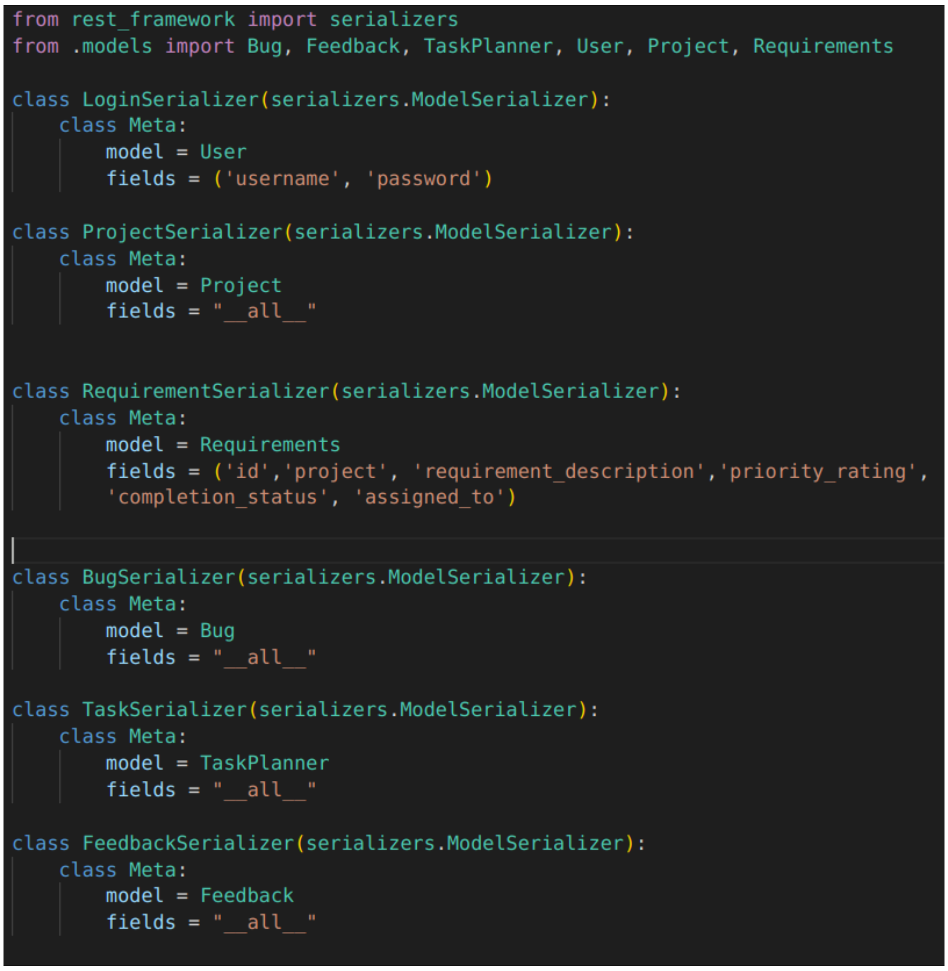


```python
class TaskPlanner(models.Model):
    task_name = models.CharField(max_length=200)
    is_completed = models.BooleanField(null=True)
    requirement = models.ForeignKey(Requirements, on_delete=models.CASCADE,
    null=True)

    def __str__(self):
        return self.task_name

class Feedback(models.Model):
    date_reviewed = models.DateTimeField()
    text = models.CharField(max_length=10000)
    project = models.ForeignKey(Project, on_delete=models.CASCADE, null=True)

    def __str__(self):
        return self.text
```

Figure 49. File models.py

4.2.1.1 Serializers

We are using Django rest framework for our project. So, we are provided with the option to use serializers (Figure 50). These can be found in the serializers.py file. Each class here is a subclass of the built-in serializers.ModelSerializer (Django Rest Framework documentation). The serializers create a meta class with selected fields from the models we saw in Figure to be used by the views.



```python
from rest_framework import serializers
from .models import Bug, Feedback, TaskPlanner, User, Project, Requirements

class LoginSerializer(serializers.ModelSerializer):
    class Meta:
        model = User
        fields = ('username', 'password')

class ProjectSerializer(serializers.ModelSerializer):
    class Meta:
        model = Project
        fields = "__all__"


class RequirementSerializer(serializers.ModelSerializer):
    class Meta:
        model = Requirements
        fields = ('id','project', 'requirement_description','priority_rating',
        'completion_status', 'assigned_to')


class BugSerializer(serializers.ModelSerializer):
    class Meta:
        model = Bug
        fields = "__all__"

class TaskSerializer(serializers.ModelSerializer):
    class Meta:
        model = TaskPlanner
        fields = "__all__"

class FeedbackSerializer(serializers.ModelSerializer):
    class Meta:
        model = Feedback
        fields = "__all__"
```

Figure 50. File serializers.py

4.2.2 Views

We saw views in Chapter 4.1 for Vue JavaScript. Views in Django are different. These contain python functions that must take in a request and return a response. They can also be grouped together to form classes as well. For the purposes of minimal manager, a combination of both have been used.

Django Rest Framework has a built-in class viewsets.ModelViewSet (viewsets documentation) which inherits from the GenericAPIView. When we create a subclass of this model with two attributes the queryset which queries the selected model and serializer_class which serializes the models, we get access to the following functions:

list (): Returns a list of all objects in the model.

retrieve (): Returns a single object from the model, based on the specified lookup field (usually the primary key).

create (): Creates a new object in the model.

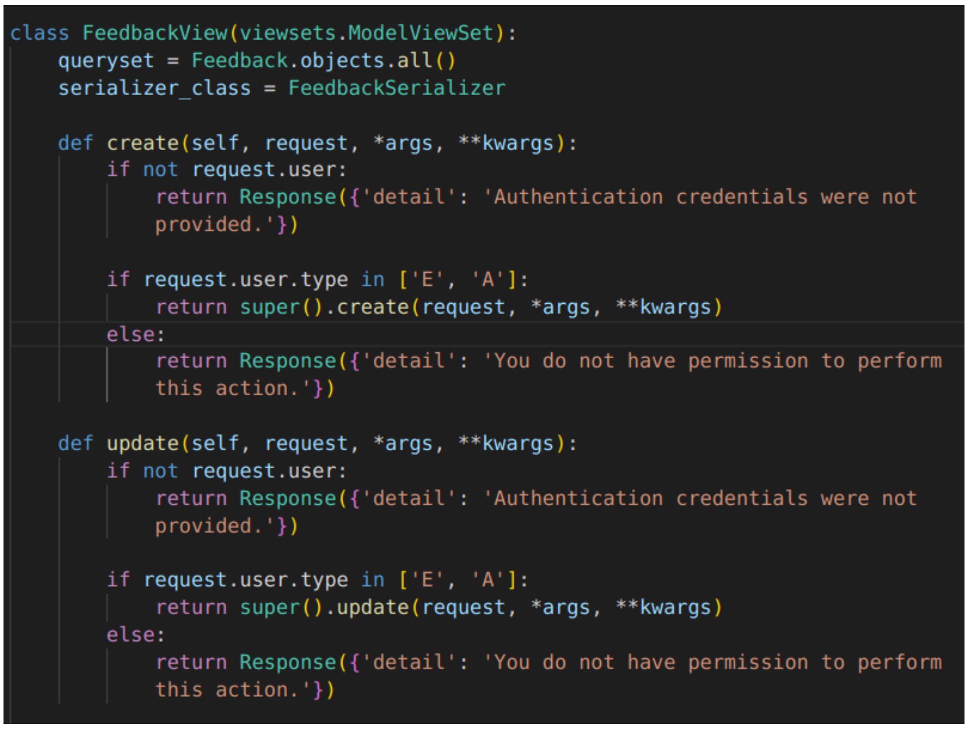
update (): Updates an existing object in the model, based on the specified lookup field.

partial_update (): Updates an existing object in the model, but only modifies the specified fields.

destroy (): Deletes an object from the model, based on the specified lookup field.

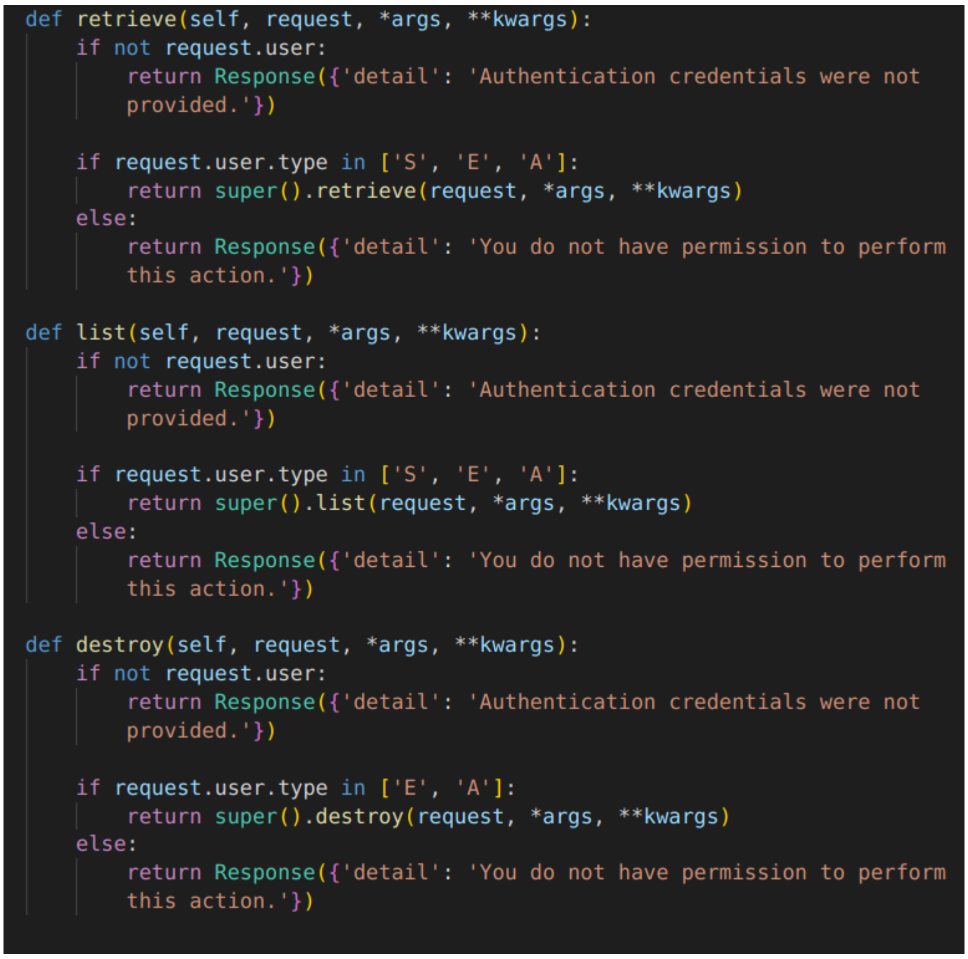
These can be overridden by and perform different actions. The only class where this was done was in FeedbackView (Figures 51-52) where specific access was given to a particular group. For other view functions, if a type of behaviour was needed which was not provided by ModelViewSet we wrote a function outside the scope of the related class.

Each views.ModelViewSet subclass and view function have a unique url defined in urls.py in the directory api. These correspond to the endpoints mentioned in Chapter 4.1.



```
class FeedbackView(viewsets.ModelViewSet):
    queryset = Feedback.objects.all()
    serializer_class = FeedbackSerializer

    def create(self, request, *args, **kwargs):
        if not request.user:
            return Response({'detail': 'Authentication credentials were not 
            provided.'})

        if request.user.type in ['E', 'A']:
            return super().create(request, *args, **kwargs)
        else:
            return Response({'detail': 'You do not have permission to perform 
            this action.'})

    def update(self, request, *args, **kwargs):
        if not request.user:
            return Response({'detail': 'Authentication credentials were not 
            provided.'})

        if request.user.type in ['E', 'A']:
            return super().update(request, *args, **kwargs)
        else:
            return Response({'detail': 'You do not have permission to perform 
            this action.'})
```

Figure 51. Feedback views



```
def retrieve(self, request, *args, **kwargs):
    if not request.user:
        return Response({'detail': 'Authentication credentials were not
        provided.'})

    if request.user.type in ['S', 'E', 'A']:
        return super().retrieve(request, *args, **kwargs)
    else:
        return Response({'detail': 'You do not have permission to perform
        this action.'})

def list(self, request, *args, **kwargs):
    if not request.user:
        return Response({'detail': 'Authentication credentials were not
        provided.'})

    if request.user.type in ['S', 'E', 'A']:
        return super().list(request, *args, **kwargs)
    else:
        return Response({'detail': 'You do not have permission to perform
        this action.'})

def destroy(self, request, *args, **kwargs):
    if not request.user:
        return Response({'detail': 'Authentication credentials were not
        provided.'})

    if request.user.type in ['E', 'A']:
        return super().destroy(request, *args, **kwargs)
    else:
        return Response({'detail': 'You do not have permission to perform
        this action.'})
```

Figure 52. Feedback views

4.2.3 CORS

When making requests in Chapter 3.1 we saw that the Api.js had functions which included the header mode:cors. CORS (StackHawk, 2021) stands for Cross-Origin Resource Sharing. This is a policy implemented by the middleware. Django uses many such middlewares by default.

A middleware sets between the client and the server and performs pre-processing and post-processing on requests and responses. CORS can be used to restrict the number of domains that can access the Api.

One of the advantages of having an Api as a server and a client running separate is that the Api can be accessed several clients. So, a more advanced client which is much faster can be built in the future for example, without ever changing the Api.

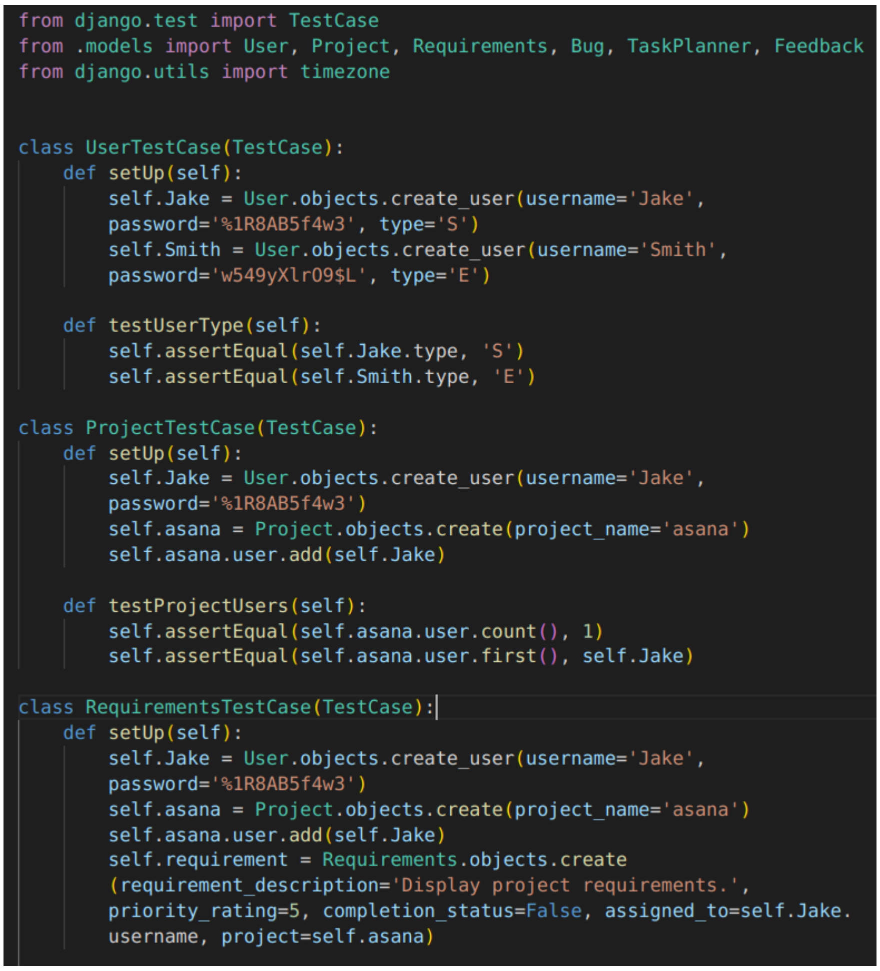
Chapter 5: **Testing**

5.1 Model Testing

Testing was carried out for the classes created in the django Api using the built-in class TestCase. The tests can be found in the api directory in the tests.py (Figures 53-55) file. Each model was imported here, and a separate class was created to test each model. There is a setUp function in every class which is overriding the one in the built-in TestCase class which creates the object. Then another function is written which must start with the word test otherwise it will not execute if the tests are run. These test various aspects such as if the:

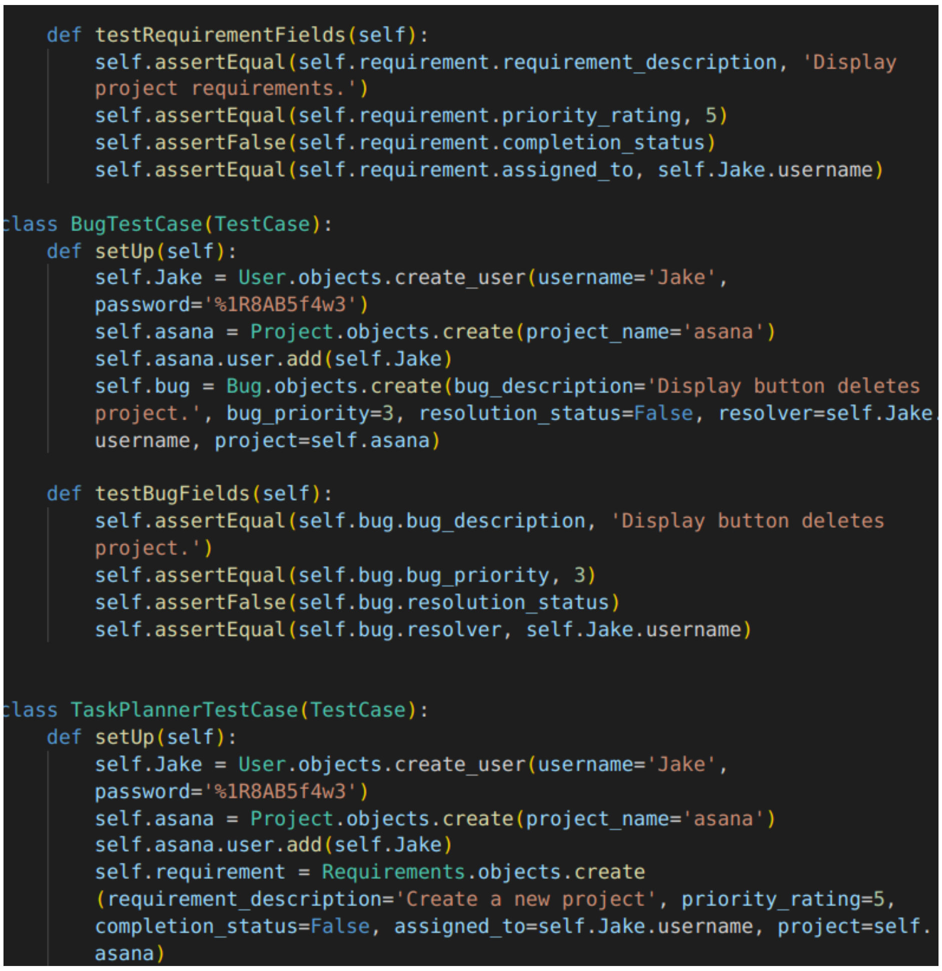
- object is an instance of the model being tested
- string instances are correctly saved in the model
- maximum length for a string field is correctly defined
- time attribute is correctly recorded

A more detailed explanation can be found in the django documentation. No errors were found for the criteria tested.



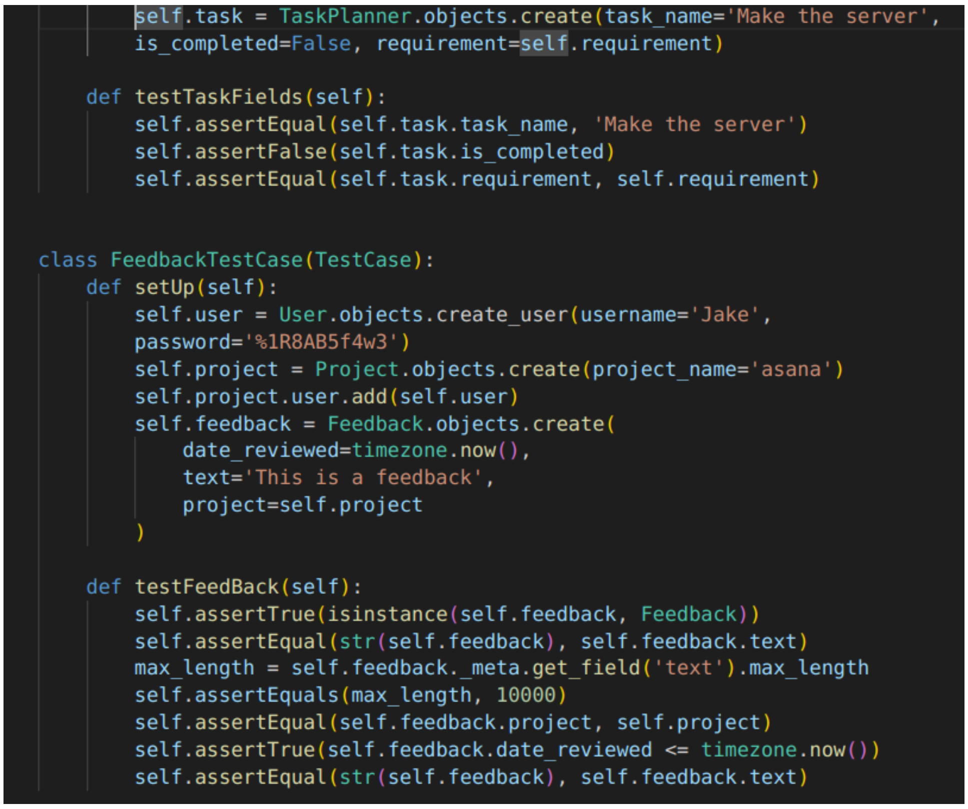
```python
from django.test import TestCase
from .models import User, Project, Requirements, Bug, TaskPlanner, Feedback
from django.utils import timezone


class UserTestCase(TestCase):
    def setUp(self):
        self.Jake = User.objects.create_user(username='Jake',
        password='%1R8AB5f4w3', type='S')
        self.Smith = User.objects.create_user(username='Smith',
        password='w549yXlr09$L', type='E')

    def testUserType(self):
        self.assertEqual(self.Jake.type, 'S')
        self.assertEqual(self.Smith.type, 'E')

class ProjectTestCase(TestCase):
    def setUp(self):
        self.Jake = User.objects.create_user(username='Jake',
        password='%1R8AB5f4w3')
        self.asana = Project.objects.create(project_name='asana')
        self.asana.user.add(self.Jake)

    def testProjectUsers(self):
        self.assertEqual(self.asana.user.count(), 1)
        self.assertEqual(self.asana.user.first(), self.Jake)

class RequirementsTestCase(TestCase):
    def setUp(self):
        self.Jake = User.objects.create_user(username='Jake',
        password='%1R8AB5f4w3')
        self.asana = Project.objects.create(project_name='asana')
        self.asana.user.add(self.Jake)
        self.requirement = Requirements.objects.create
        (requirement_description='Display project requirements.',
        priority_rating=5, completion_status=False, assigned_to=self.Jake.
        username, project=self.asana)
```

Figure 53. Filename tests.py



```
    def testRequirementFields(self):
        self.assertEqual(self.requirement.requirement_description, 'Display
        project requirements.')
        self.assertEqual(self.requirement.priority_rating, 5)
        self.assertFalse(self.requirement.completion_status)
        self.assertEqual(self.requirement.assigned_to, self.Jake.username)

lass BugTestCase(TestCase):
    def setUp(self):
        self.Jake = User.objects.create_user(username='Jake',
        password='%1R8AB5f4w3')
        self.asana = Project.objects.create(project_name='asana')
        self.asana.user.add(self.Jake)
        self.bug = Bug.objects.create(bug_description='Display button deletes
        project.', bug_priority=3, resolution_status=False, resolver=self.Jake.
        username, project=self.asana)

    def testBugFields(self):
        self.assertEqual(self.bug.bug_description, 'Display button deletes
        project.')
        self.assertEqual(self.bug.bug_priority, 3)
        self.assertFalse(self.bug.resolution_status)
        self.assertEqual(self.bug.resolver, self.Jake.username)


lass TaskPlannerTestCase(TestCase):
    def setUp(self):
        self.Jake = User.objects.create_user(username='Jake',
        password='%1R8AB5f4w3')
        self.asana = Project.objects.create(project_name='asana')
        self.asana.user.add(self.Jake)
        self.requirement = Requirements.objects.create
        (requirement_description='Create a new project', priority_rating=5,
        completion_status=False, assigned_to=self.Jake.username, project=self.
        asana)
```

Figure 54.   Filename tests.py



```
        self.task = TaskPlanner.objects.create(task_name='Make the server',
        is_completed=False, requirement=self.requirement)

    def testTaskFields(self):
        self.assertEqual(self.task.task_name, 'Make the server')
        self.assertFalse(self.task.is_completed)
        self.assertEqual(self.task.requirement, self.requirement)


class FeedbackTestCase(TestCase):
    def setUp(self):
        self.user = User.objects.create_user(username='Jake',
        password='%1R8AB5f4w3')
        self.project = Project.objects.create(project_name='asana')
        self.project.user.add(self.user)
        self.feedback = Feedback.objects.create(
            date_reviewed=timezone.now(),
            text='This is a feedback',
            project=self.project
        )

    def testFeedBack(self):
        self.assertTrue(isinstance(self.feedback, Feedback))
        self.assertEqual(str(self.feedback), self.feedback.text)
        max_length = self.feedback._meta.get_field('text').max_length
        self.assertEquals(max_length, 10000)
        self.assertEqual(self.feedback.project, self.project)
        self.assertTrue(self.feedback.date_reviewed <= timezone.now())
        self.assertEqual(str(self.feedback), self.feedback.text)
```

Figure 55. Filename tests.py

5.2 Functionality Testing

5.2.1 Methodology

All expected behaviour that was described was observed by testing the different forms and buttons. Additionally, the page was refreshed to make sure that the Api responded correctly as well. The following behaviours were tested:

1. The buttons performed the correct actions when clicked.
2. Showed errors when if the user made a mistake
3. Display correct information for appropriate users

Here are some of the functionalities that have been personally tested and the correct behaviour was observed, both on the client and server. Note: CRUD stands for create, read, update, and delete.

5.2.2 For both client and server, the following correct behaviour was seen:

1. Login and logout.
2. CRUD of Project model.
3. CRUD of Requirements model.
4. CRUD of BugTracker model.
5. CRUD of TaskPlanner model.
6. CRUD of FeedBack model.

5.2.3 For client side the following the correct behaviour was seen:

1. Feedback or buttons form not displayed correctly for students but correctly for educators.
2. Navbar items correctly loads the described page.
3. Errors shown if unexpected length of inputs were received.
4. All forms and buttons work correctly as expected.
5. Lists correctly displayed for the members added to a project.

5.2.4 Following errors were revealed and corrected:

1. After posting a new project it was automatically highlighted even though it was not selected.

5.2.5 Following errors could not be corrected:

1. An error was shown when a blank form was submitted but no error was show if just white spaces were submitted. The Api did not accept the input and displayed the correct message in the console.

Chapter 6: **Running on the local server**

6.1 Running on the local server

Unfortunately, deployment was not possible due to some technical failure just before the submission deadline with not enough time to fix the issue. However, the code works on the local server as expected. Please follow these instructions to run the code. If you have any problems, please contact me at ec20542@qmul.ac.uk.

6.1.1 Running Django:

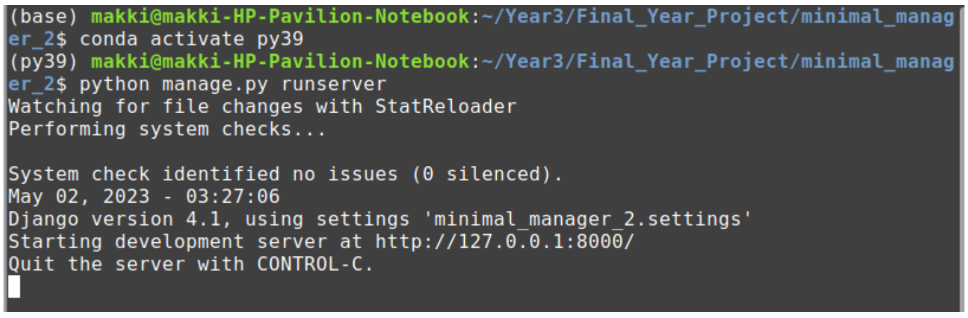
- To run the code that was included in the zip file, we need to first unzip.
- One such software that could be used is Winrar. It is also available for linux and mac and can be downloaded from here.
- After opening the directory minimal_manager in terminal/command prompt we need to first create a conda environment.
- Instructions on how to create a conda environment can be found here.
- We have used python 3.9 to create this project so the version of python used to create the environment must also be the same.
- After that we need to activate the environment. Then we need to install the dependencies in the requirements.txt file by running:

```
pip install -r requirements.txt
```

This will install all the dependencies. Django version 4.1 was used for this project, but it should be automatically installed if the command is run. Once the dependencies are installed, we need to run the command:

```
python manage.py runserver
```

This will start the Django server and look something like this:



```
(base) makki@makki-HP-Pavilion-Notebook:~/Year3/Final_Year_Project/minimal_manag
er_2$ conda activate py39
(py39) makki@makki-HP-Pavilion-Notebook:~/Year3/Final_Year_Project/minimal_manag
er_2$ python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
May 02, 2023 - 03:27:06
Django version 4.1, using settings 'minimal_manager_2.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

6.1.2 Running Vue:

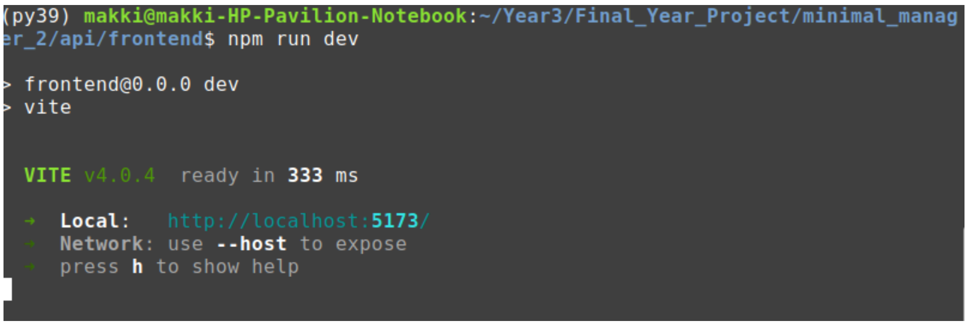
- Before we proceed, we need to install npm and node.
- They can be installed here.
- For this project, the npm version is 8.19.3 and node version 16.18.0.
- Open a new terminal or command prompt in the frontend directory.
- The path for the directory is minimal_manager/api/frontend.
- Here we need to install the node modules.
- To do this we run the command:

```
npm install
```

Finally after successfully installing the node modules we can run:

```
npm run dev
```

Which will look something like this:



```
(py39) makki@makki-HP-Pavilion-Notebook:~/Year3/Final_Year_Project/minimal_manag
er_2/api/frontend$ npm run dev

> frontend@0.0.0 dev
> vite


  VITE v4.0.4  ready in 333 ms

  ➜  Local:   http://localhost:5173/
  ➜  Network: use --host to expose
  ➜  press h to show help
```

The link displayed can be opened in the browser to access the app. It was only tested on Google chrome but should work on most other popular browsers.

Chapter 7: **Conclusion**

Overall, the main objectives that were set in the beginning of the project were covered. Our achievement was the complete implementation of a system with minimalistic but covering the important fundamentals of agile development. This process is vital for project development as dividing the project in tasks and subtasks and then receiving feedback in the initial stages to modify ideas can increase productivity and accuracy of the final output.

The development of minimal manager from scratch has been a wonderful journey that has taught me a lot about the web frameworks Vue and Django. I was made aware of many techniques and concepts of programming that I was not before. For example, I did not how to implement token authentication which I had to learn from scratch. I want to use the lessons I learnt here as a foundation for learning more concepts about web applications and build exemplary systems.

Due to lack of time and skill gap at this current moment, there were many ideas that could not be implemented. Some of them are mentioned below:

1. Testing the client-side code with Svelte
2. Using Vuetify in the client side so that better error messages were shown to the user.
3. Getting others to test the system to find faults and shortcomings so that they can be improved further.
4. Taking further advantage of the reusability of components in Vue.

Finally, I would like to especially thank Professor Mathew Huntbach for his time and imparting his wisdom, without which this project would not have been possible.

References

1. Fowler, M. and Highsmith, J., 2001. The agile manifesto. Software development, 9(8), pp.28-35.

2. Puška, A., Stojanovic, I., Maksimovic, A. and Osmanovic, N., 2020. Project management software evaluation by using the Measurement of Alternatives and Ranking According to Compromise Solution (MARCOS) method. Operational Research in Engineering Sciences: Theory and Applications, 3(1), pp.89-102.

3. Fioravanti, M.L., Sena, B., Paschoal, L.N., Silva, L.R., Allian, A.P., Nakagawa, E.Y., Souza, S.R., Isotani, S. and Barbosa, E.F., 2018, February. Integrating project based learning and project management for software engineering teaching: An experience report. In Proceedings of the 49th ACM technical symposium on computer science education (pp. 806-811)

4. Korean Educational Development Institute (Korean Educational Statistics Service). (2021). Number of enrolled engineering students in South Korean universities in 2020, by disciplinary field. Statista. Statista Inc.. Accessed: November 17, 2022. https://www.statista.com/statistics/823827/south-korea-enrolled-engineering-student-number-by-disciplinary-field/

5. Balaji, S. and Murugaiyan, M.S., 2012. Waterfall vs. V-Model vs. Agile: A comparative study on SDLC. International Journal of Information Technology and Business Management, 2(1), pp.26-30.

6. Marques, J.F., Bernardino, J., Dietz, J.L.G., Aveiro, D. and Filipe, J., 2019, September. Evaluation of Asana, Odoo, and ProjectLibre Project Management Tools using the OSSpal Methodology. In KEOD (pp. 397-403).

7. Asana (2019). Asana Pricing | Premium, Business, and Enterprise pricing plans · Asana. [online] Asana. Available at: https://asana.com/pricing.

8. Mishra, A. and Mishra, D. (2013). Software project management tools. ACM SIGSOFT Software Engineering Notes, 38(3), p.1. doi:10.1145/2464526.2464537.

9.Statista. (2021). Population of internet users worldwide from 2012 to 2021, by operating system (in millions). Statista. Statista Inc.. Accessed: November 24, 2022. https://www-statista-com.ezproxy.library.qmul.ac.uk/statistics/543185/worldwide-internet-connected-operating-system-population/

10. Borozenets, M. (2022). Vue vs. React—Comparison. What Is the Best Choice for 2022? - Fulcrum. [online] https://fulcrum.rocks/.
Available at: https://fulcrum.rocks/blog/vue-vs-react-comparison [Accessed 24 Nov. 2022].

11. Vertegaal, R., 2003. Attentive user interfaces. Communications of the ACM, 46(3), pp.30-33.

12. Roel Vertegaal. 2002. Designing attentive interfaces. In Proceedings of the 2002 symposium on Eye tracking research &amp; applications (ETRA '02). Association for Computing Machinery, New York, NY, USA, 23–30. https://doi-org.ezproxy.library.qmul.ac.uk/10.1145/507072.507077

13. Quitério, J and Ribeiro, J. 2014. Django-CMS overview. In Proceedings of the International Conference on Information Systems and Design of Communication (ISDOC '14). Association for Computing Machinery, New York, NY, USA, 180–181. https://doi-org.ezproxy.library.qmul.ac.uk/10.1145/2618168.2618201

14. Stonebraker, M. and Rowe, L.A. 1986. The design of POSTGRES. In Proceedings of the 1986 ACM SIGMOD international conference on Management of data (SIGMOD '86). Association for Computing Machinery, New York, NY, USA, 340–355. https://dl-acm-org.ezproxy.library.qmul.ac.uk/doi/10.1145/16894.16888

15. MHRD (India). (2021). Number of students enrolled in engineering at an undergraduate level across India in 2020, by discipline (in 1,000s). Statista. Statista Inc.. Accessed: November 26, 2022. https://www-statista-com.ezproxy.library.qmul.ac.uk/statistics/765482/india-number-of-students-enrolled-in-engineering-stream-by-discipline/

16. Bagdonas, C. (2022). Get started (in 15 mins or less). [online] Asana Academy. Available at: https://academy.asana.com/get-started-with-asana/587487 [Accessed 26 Nov. 2022].

17. Smartsheet (2019). Smartsheet: Less Talk, More Action. [online] Smartsheet. Available at: https://www.smartsheet.com/.

18. Davies, A. (2022). Agile vs Waterfall: Which Methodology is Right for Your Project. [online] DevTeam.Space. Available at: https://www.devteam.space/blog/agile-vs-waterfall-which-methodology-is-right-for-your-project/.

19. Fielding, R. T. (2000). Architectural Styles and the Design of Network-based Software Architectures (Doctoral dissertation). University of California, Irvine. Retrieved from https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

19. StackHawk. (2021, July 21). Django CORS Guide. StackHawk. https://www.stackhawk.com/blog/django-cors-guide/

Appendix: Risk Register

Description of Risk	**Description of impact**	**Likelihood Rating**	**Impact Rating**	**Preventative Actions**
Inaccessible resources.	This is likely to occur if I need to access some scientific journals are behind a paywall hence inaccessible. This would lead to a report not based on research but anecdotal evidence.	low	medium	Restrict search to databases such as ACM, Statista, IEEE Xplore that are provided by Queen Mary for free.
Updates of Django, python, JavaScript, etc. during development.	If the languages or frameworks get updates during development process, some syntax, method, or class names might no longer be valid in the newer versions and cause errors.	high	high	Try to keep a list of the languages and frameworks being used and their specific versions. Create virtual environments, where only the versions used when starting the project are installed.

Inaccessible software	Software such as visual paradigm are useful for creating diagrams that are required for the final report. These may be inaccessible if they are behind a paywall.	low	low	Make use of software available by Queen Mary. Visual Paradigm is available to EEECs students in Queen Mary. Also make use of free drawing tools available online.
Lack of depth in knowledge.	If we use a language or framework, we only have a basic understanding of, we may get stuck often during the development process, increasing the time required for project completion and not meeting the deadline.	medium	medium	Use frameworks and languages that we are already familiar with. Take modules relevant to the project being implemented so the need to learn extra technologies get reduced.
Proceeding to the next development stage without testing current stage first.	If we keep on writing code without testing regularly, it may be difficult or even impossible to find bugs when the project becomes large. Hence failure to implement the project on time or even having a working program by the deadline	high	high	It may be useful to use techniques such as abstraction and the DRY (Don't Repeat Yourself) principle to structure the code which will make it easier to find bugs. Also writing test in advance even before writing code may be useful.

	might not be possible.			
Lack of human participants to test code.	Failure to gather human participants to test aspects of the system such as the user interface will result in a system which may contain bugs that were overlooked or an interface that is difficult to use. Hence, even if the development is completed the system may not be of a high quality.	high	low	Try to contact users in person after reaching out to them via email. Carry a tablet or other device where they can immediately test the system give feedback. Create dummy username and passwords so users waste time on creating an account and logging in.
Not meeting set milestones.	Not sticking to set milestones in the beginning of the project is sign of poor time management. It may result in not being able to deliver the project by the deadline or producing a low-quality product.	high	high	Try to set small daily goals to avoid procrastination. Set time aside daily to work on projects. Use a pomodoro method. 20 minutes of study time followed by 5 mins of break.