

University of Nevada, Reno

Department of Computer Science and Engineering

CS 425: Software Engineering



Optimum Property Fix

Team 05

Aisha Co, Melissa Flores, Joanna Lopez, Nasrin Juana and Araam Zaremenhjardi

Instructor: Sergiu Dascalu, Vihn Le, and Devrin Lee

Advisor: Erin Keith

December 9, 2021

Table of Contents

1. Abstract	2
2. Introduction	2
3. Prototype Objectives & Functionality	3
4. Develop Prototype	4
4.1. Create New Maintenance Request Page	4
4.2. User Sign-in page	5
4.3 Dashboard Page	11
4.4 Student View Maintenance Request Page	12
4.5 Student Feedback Page	14
4.6 Chatbot	16
5. Demo Prototypes	18
5.1. Demo Prototype time	18
5.2. Summary of Comments and Recommendations	18
6. Team Contributions	19
6.1. Araam Zaremehrjardi's Contribution	19
6.2. Joanna Lopez's Contribution	19
6.3. Melissa Flores' Contribution	20
6.4. Nasrin Juana's Contribution	20
6.5. Aisha Co's Contribution	21
7. Implemented Code	21
7.1 Araam Zaremehrjardi	21
7.2. Joanna Lopez	37
7.3. Melissa Flores	46
7.4. Nasrin Juana	50
7.5. Aisha Co	56
7.6. GitHub Repository	56

1. Abstract

The current dormitory maintenance requests system at the University of Nevada, Reno (UNR) provides a system that is flawed and susceptible to complications. Some issues the current system faces are paper trails, leading to lost reports, a lack of maintenance request progress tracking for the students, and does not incentivize students to submit issues that occur within the property. With this, Optimum Property Fix (OPF) aims to revamp UNR's current system through a new and improved maintenance request system, accessible to dormitory students as well as facilities and service members. OPF's functionality fully eliminates the paper-based system and replaces it with a web application that retains all maintenance records in one place. Major accomplishments that were recently made for OPF are functional pages of the student side, consisting of communication means, forms, and a sign-in page. The web application utilizes artificial intelligence and quality data through two different user-focused experiences.

2. Introduction

Optimum Property Fix (OPF) is a web application designed to assist the University of Nevada, Reno (UNR) dormitory facility services. By acting as a communication bridge between facility management members and dormitory students, OPF will act as a tool to manage UNR living spaces. The current system has paper trails, is vulnerable to being lost, does not incentivize submission of minor issues within living spaces, and lacks progress tracking of maintenance requests to students. OPF will rectify these issues while providing analytical insights that allow Facilities and Management to preemptively manage dormitory buildings and assess dormitory buildings' health.

Recent progress made towards OPF would be the implementation of the student pages and the chatbot based on the user interface (UI) design developed through Adobe XD. The design followed the initial layout of the UI design as well as the color scheme, so the web application would be clear and uniform. Correspondingly, this would allow the users to intuitively navigate OPF. The student pages that were developed were the user sign-in page, create new maintenance request page, dashboard page, view maintenance requests page, and feedback page. The integration of the student webpage version and the chatbot creates the experience of navigating through the OPF web application. Currently, the web pages and the chatbot allow general interaction between the web application and the user.

OPF's progress is tracked and managed regarding software code via Github version control. Each implementation was isolated onto its branch, so the versions of each implementation were tracked and not affected by other implementations. The README file reflects the progress of the software engineering project. The text file contains the introductions and explanations of the overall project.

3. Prototype Objectives & Functionality

The OPF prototype allows the potential users to understand the user interface aspect from a student's point of view. It focuses more on the front-end aspect of the app and displays what a student will view while using the app. The prototype features six implementations of OPF that can be utilized as a foundation for similar pages for additional users including students, facilities and management members, and administrators. The features implemented in the prototypes are the user log-in page, create new maintenance request page, dashboard page, feedback page, view maintenance requests page, and the chatbot.

The create new maintenance request page allows a student to submit a maintenance request and is an integral part of OPF. This page also contains basic information and components necessary for other pages. For instance, the maintenance requests page from both student' and facilities perspectives will use information submitted by the create maintenance request page. As such, the implementation of this page will play an important role in the OPF application.

The user sign-in page is the first a user views when they view the OPF webpage. This page enables the user to log in with an existing account with OPF by prompting the user to enter their username and password. Another functionality on this page includes an option that allows users to reset their password if forgotten. Additionally, if a user does not already have an existing account with OPF, the user may proceed to create a new account and input any credentials needed. Users can input a username, a password with no restrictions such as character length, email address, and lastly a phone number. This page utilizes AWS Cognito, which enables user authentication through an external identity provider. With this, a confirmation code is sent via email to the user which completes the creation of the account. This page will be used by all the users, so it is an important part of the application.

Continuing onto the dashboard page, this page will be utilized by all the users and it will include a calendar to keep track of a user's appointments and maintenance requests. It will have a dynamic view and will change according to the user's activities and messages. The prototype implements the dashboard from a student's point of view, but the functionality of the dashboard will be similar in all the groups of users. As a result, this implementation can be used as a base for the dashboard in other sections.

Similar to the dashboard, the maintenance requests page implemented from the student's point of view can act as a base for the maintenance requests page for the other groups. For a student user, this page shows the list of completed and in-progress maintenance reports and their associated information. It will serve as an integral part of OPF and can be utilized by all the users to view their maintenance request's status.

Continuing onto the feedback page for the students, this shows the feedback from a student can fill out after a maintenance request has been completed. Through this page, the students and the facilities, and management members would be able to communicate with each other.

Correspondingly, the facilities and management members will also be able to improve their future maintenance.

Next, the chatbot will act as an interactable tool that can be used by all the users to answer various questions. The prototype contains a chatbot that can interact with the users now via text and widget options; however, future implementations to enhance the chatbot will be allowing the chatbot to answer complex questions (based on information stored in a database) and navigate the users through the application.

The implemented features demonstrated in the prototype focuses on the six main parts of the front-end side of the application. The implementation of these features also includes many components that can be factored into other features in the future. These parts can be utilized in the future with other components and can act as a foundation for the other features.

4. Develop Prototype

The snapshots demonstrate functional prototypes of the student view of OPF through five essential pages of the web application. The figures in section 4 include the create new maintenance request page, user sign-in page, student dashboard, student maintenance request page, student feedback page, and the chatbot feature. Each component displays the user interface and interactive design of these pages along with brief descriptions.

4.1. Create New Maintenance Request Page

The create new maintenance request page allows a student to submit a new maintenance request. The student will be able to enter information into a maintenance request ticket. The information will be submitted and saved to a local database. The maintenance request will then be sent to an administrator for further action and will be displayed in the user's dashboard. Figure 4.1.1 demonstrates how a student will view this page.

The figure shows a screenshot of a web application titled 'React App' at the top left. The address bar indicates the URL is 'localhost:3000/createmaintenancerequest'. The main content area is titled 'Submit Maintenance Request'. On the left, there's a sidebar with a user icon, the name 'Welcome, John Doe', and several navigation links: 'Dashboard', 'View Maintenance Request', 'Create New Maintenance Request' (which is currently selected), and 'Feedback'. The main form has several input fields: 'Request Title' (text input), 'Request Description' (text input), 'Set Priority' (radio buttons for 'NONE', 'LOW', 'MEDIUM', and 'HIGH'), 'Additional Notes' (text input), and 'Tags' (text input). Below these are buttons for 'Set Severity' (radio buttons for 'LOW', 'MILD', and 'HIGH') and 'Building' (text input). There's also a section for 'Upload Images or Video' with a placeholder 'Drop images or video here' and a gear icon in the top right corner.

Fig. 4.1.1: The figure outlines the ‘Create New Maintenance Request’ page. The page has multiple text boxes for the user to input a request title, description, the building name, upload an image and/or video, and any additional notes and tags that the user would like to add to their respective request.

4.2. User Sign-in page

The user sign-in page is the first page viewed when visiting the OPF website. On this page, the user who already has an account is prompted to sign in to their respective account credentials. Next, the text box area is where the user will be able to enter their ‘Username’ and ‘Password’ when first registering their account. The user is also able to click on the ‘Reset password’ link if they have forgotten their credentials. Additionally, the log-in page also has a link where a new user to OPF will be able to ‘Create account’. Lastly, there is a ‘Sign-In’ button that the user can click to enter their respective dashboard. Figures 4.2.1, 4.2.2, 4.2.3, 4.2.4, 4.2.5, and 4.2.6 demonstrate how a user can interact with the Sign-in page.

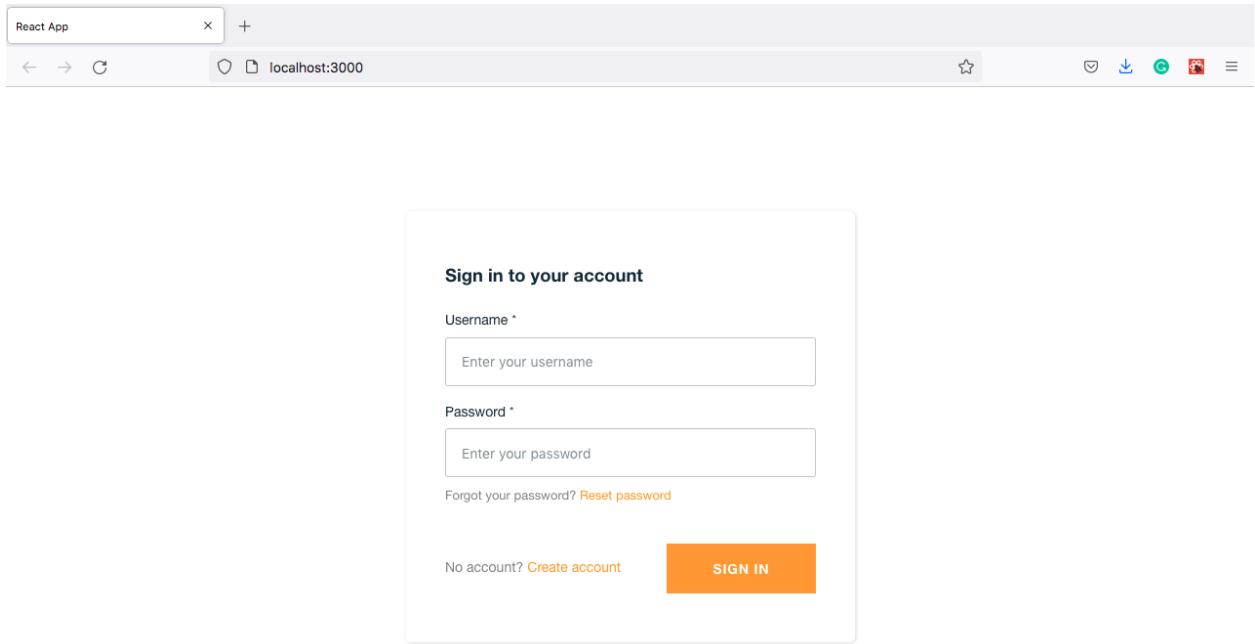


Fig. 4.2.1: The login page for users of the OPF website includes a text box for the ‘username’ and ‘password’. The user will then be able to sign in using the ‘sign in’ button to be taken to their respective account.

The screenshot shows a web browser window titled "React App" with the URL "localhost:3000". The main content is a form titled "Create a new account". The form fields are:

- Username *: An input field containing "Username".
- Password *: An input field containing "Password".
- Email Address *: An input field containing "Email".
- Phone Number *: A field consisting of a dropdown menu with "+1" and a text input with "(555) 555-1212".

At the bottom left, there is a link "Have an account? [Sign in](#)". At the bottom right, there is a large orange button labeled "CREATE ACCOUNT".

Fig. 4.2.2: A new user to the OPF website will be able to create their credentials such as their preferred ‘username’ and ‘password’. The new user will then need to supply their ‘email address’ and their ‘phone number’ for verification purposes. Team 05, has been selected to verify account credentials to the OPF website via email.

The figure shows a screenshot of a web browser window titled "React App". The address bar indicates the page is "localhost:3000". The main content is a form titled "Create a new account". The form fields are as follows:

- Username *: joannalopez
- Password *: (redacted)
- Email Address *: joannalopez@nevada.unr.edu
- Phone Number *:
 - +1
 - (209) 349-1035

Below the form, there are two buttons: "Have an account? [Sign in](#)" and a large orange "CREATE ACCOUNT" button.

Fig. 4.2.3: The figure shows the filled-out text boxes with account information needed to access the OPF website. Once the information has been entered, the user will click on the ‘create account’ button to then take them to a confirmation page.

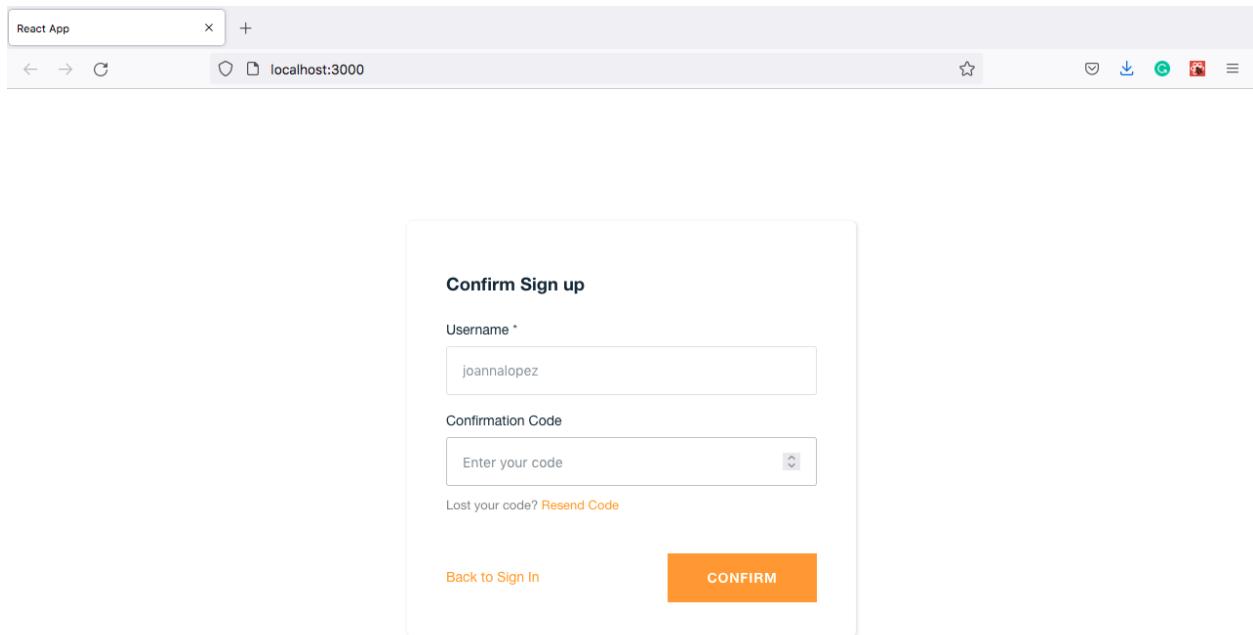


Fig. 4.2.4: The figure shows the ‘confirm sign up’ page with the user's preferred ‘username’ and a ‘confirmation code’ text box where the user can enter the code sent via their provided email.

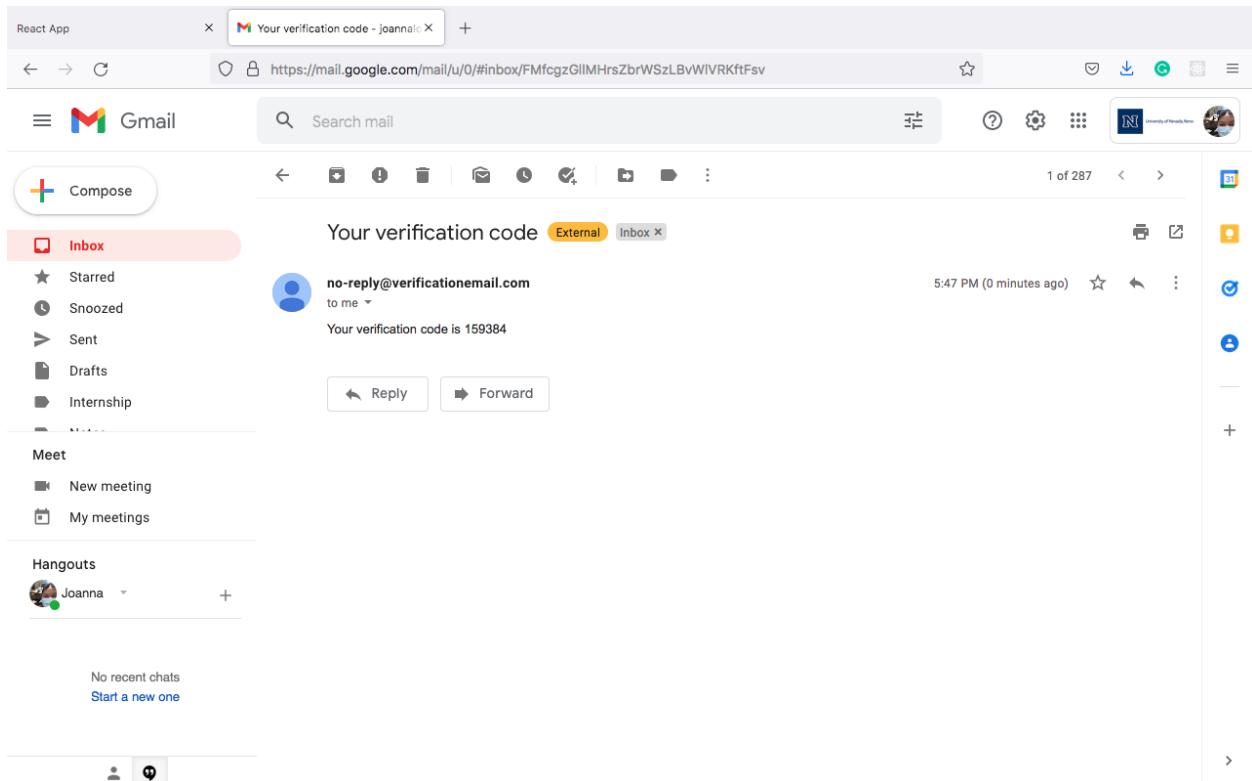


Fig. 4.2.5: The figure shows the email sent to the user and a six-digit verification code to authenticate their OPF account.

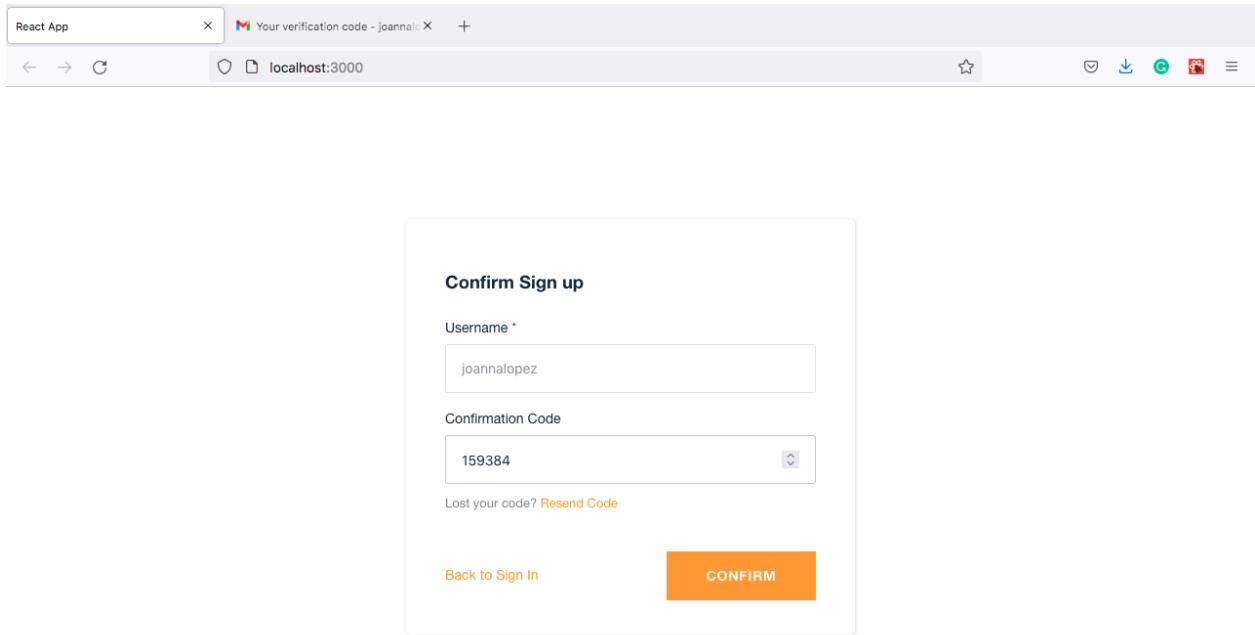


Fig. 4.2.6: The figure shows the inputting of the six-digit verification code to the ‘Confirm sign up’ page on the OPF account. The user will then be able to confirm their account by clicking the ‘Confirm’ button. The ‘confirm sign up’ page also has a ‘resend code’ in case the code was not received to their email or lost.

4.3 Dashboard Page

The dashboard page is a page that the users first view after logging into an OPF account. The dashboard page has a side navigation bar on the left side which the user can navigate to different pages. The top of the web page includes a header welcoming the user alongside a picture icon supplied by the user and a settings icon where the user can change their profile settings. There is also a chatbot icon that the user may interact with at the bottom right of each page. Additionally, the dashboard page contains a calendar to aid the users to keep track of their respective appointments, faculty messages, and any current maintenance requests tickets that are pending. Figure 4.3.1 demonstrates an overview of the dashboard page from a student perspective.

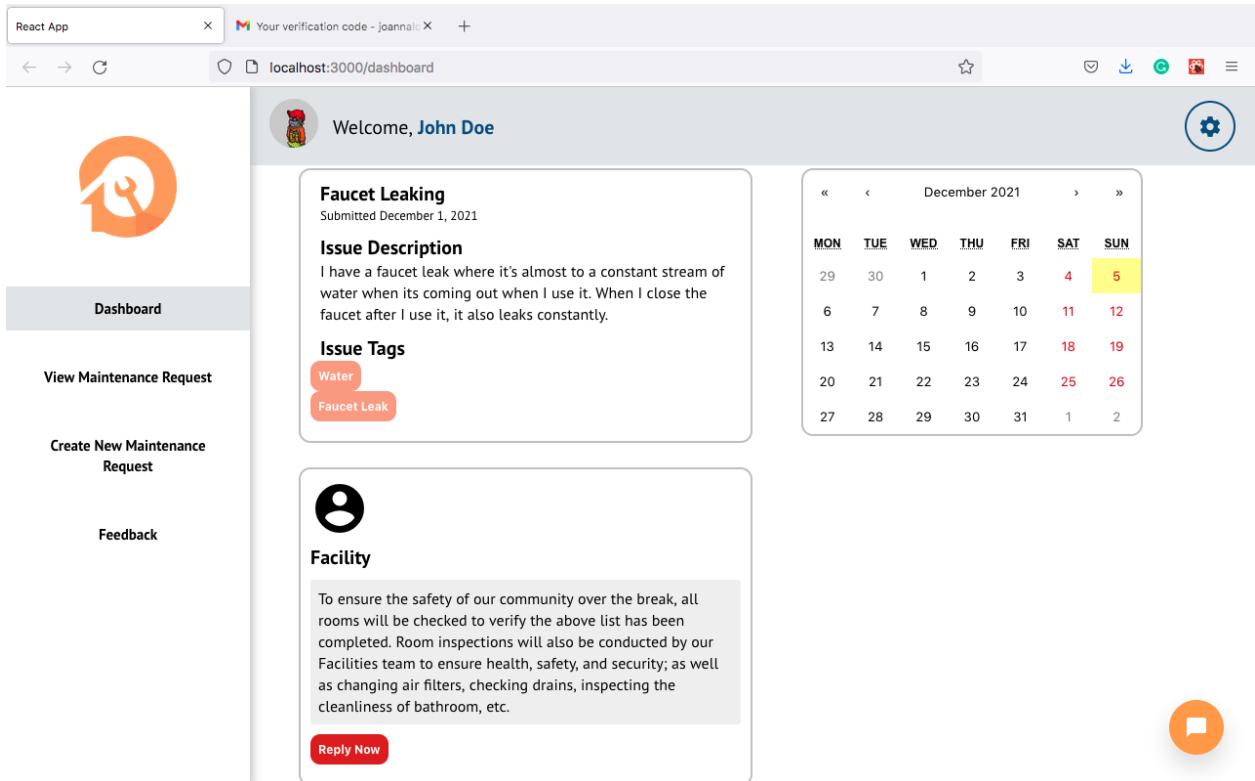


Fig. 4.3.1: The figure outlines the overview of the dashboard page including a faucet leaking ticket submitted by the user, a faculty message provided by an R.A at UNR, and a calendar that the user will be able to view any upcoming appointments.

4.4 Student View Maintenance Request Page

The Student View Maintenance Request page displays the list of completed and in-progress maintenance reports and their associated information such as their ID, date the maintenance was requested, the priority level, the severity level, and the location to a student. Currently, the student is able to search for their maintenance requests and the page includes a quick filter button that opens a drop-down menu. In the future, this button will allow the user to sort and filter the reports according to their priority, severity, and date. Figure 4.4.1 demonstrates the view maintenance request page and figure 4.4.2 demonstrates the features of the page.

The screenshot shows a web browser window titled 'React App' with the URL 'localhost:3000/viewmaintancerequest'. The page is titled 'Welcome, John Doe' and features a sidebar with icons for Dashboard, View Maintenance Request (selected), Create New Maintenance Request, and Feedback. The main content area is titled 'View Maintenance Requests' and displays a table of past tickets. The table columns are Status, Date requested, ID, Priority, Severity, and Location. The data includes:

Status	Date requested	ID	Priority	Severity	Location
Completed	26-Aug-2021	201345	High	Medium	Kitchen
Completed	19-Jan-2021	123456	High	High	Bathroom
Completed	31-Oct-2021	546373	Low	Medium	Bedroom
In Progress	26-Nov-2021	845327	High	Low	Kitchen
Completed	20-Oct-2020	968534	High	Low	Living room
Completed	19-Nov-2021	496548	Low	Medium	Living room
Completed	27-Mar-2021	675849	Medium	High	Bedroom
In Progress	20-Nov-2021	680573	Low	Medium	Kitchen
Completed	09-May-2021	860471	Medium	Low	Bedroom
Completed	19-Jan-2021	895640	Low	High	Living room

Fig 4.4.1: The ‘View Maintenance Request’ page shows all requests made by the user. The maintenance request shows the status of past tickets labeled as ‘Status’. The headers include ‘Date requested’, ‘ID’ of the request, ‘Priority’ of the request, the ‘Severity’, and the ‘Location’. The page also includes a search text box where a student is able to search for a maintenance request. In addition, a quick filter option to sort through the requests is also included for the user.

Status	Date requested	ID	Priority	Severity	Location
Completed	26-Aug-2021	201345	High	Medium	Kitchen
Completed	19-Jan-2021	123456	High	High	Bathroom
Completed	31-Oct-2021	546373	Low	Medium	Bedroom
Completed	20-Oct-2020	968534	High	Low	Living room
Completed	19-Nov-2021	496548	Low	Medium	Living room
Completed	27-Mar-2021	675849	Medium	High	Bedroom
Completed	09-May-2021	860471	Medium	Low	Bedroom
Completed	19-Jan-2021	895640	Low	High	Living room

Fig. 4.4.2: The figure shows the requests made by filtering the tickets by a ‘Completed’ keyword in the search text box. The ‘Quick Filter’ option also shows other filters the user may use such as the request being ‘Priority’, ‘Severity’, and ‘Date’.

4.5 Student Feedback Page

The student feedback page presents the student with a form that contains a set of questions to evaluate the facility’s services. It consists of a drop-down question, allowing the student to choose what request they would like to provide feedback for and required text areas to provide an evaluation of the service. Figures 4.5.1 and 4.5.2 demonstrate the Feedback page and how a user can interact with the page.

The screenshot shows a web application interface titled "React App" at the top left. The main content area is titled "Your verification code - joanna" and "localhost:3000/feedback". The page is titled "Send Us Your Feedback". On the left sidebar, there is a user profile icon and the name "John Doe". The sidebar also includes links for "Dashboard", "View Maintenance Request", "Create New Maintenance Request", and "Feedback". The "Feedback" link is highlighted with a grey background. The main content area has two sections: "Select Your Request" (with a dropdown menu labeled "Which request?") and "Send Us Your Feedback". Under "Send Us Your Feedback", there is a rating scale from 1 to 10 with radio buttons. Below this is a section titled "Questionnaire" containing three text input fields: "Were you satisfied with your service?", "Did maintenance attend to your needs right away?", and "Feedback for the facility?". To the right of the questionnaire is a large text input field labeled "Include any more notes that need to be addressed?". A small orange circular icon with a white speech mark is located in the bottom right corner of the main content area.

Fig. 4.5.1: Once the user has created a maintenance request and that request has been completed, they will have the ability to submit feedback for the request. The page presents the user with a set of questions that will help evaluate the service that was provided. The student may select which request they would like to submit feedback for and then proceed by answering a few questions. The questions include a rating component and then simple text boxes.

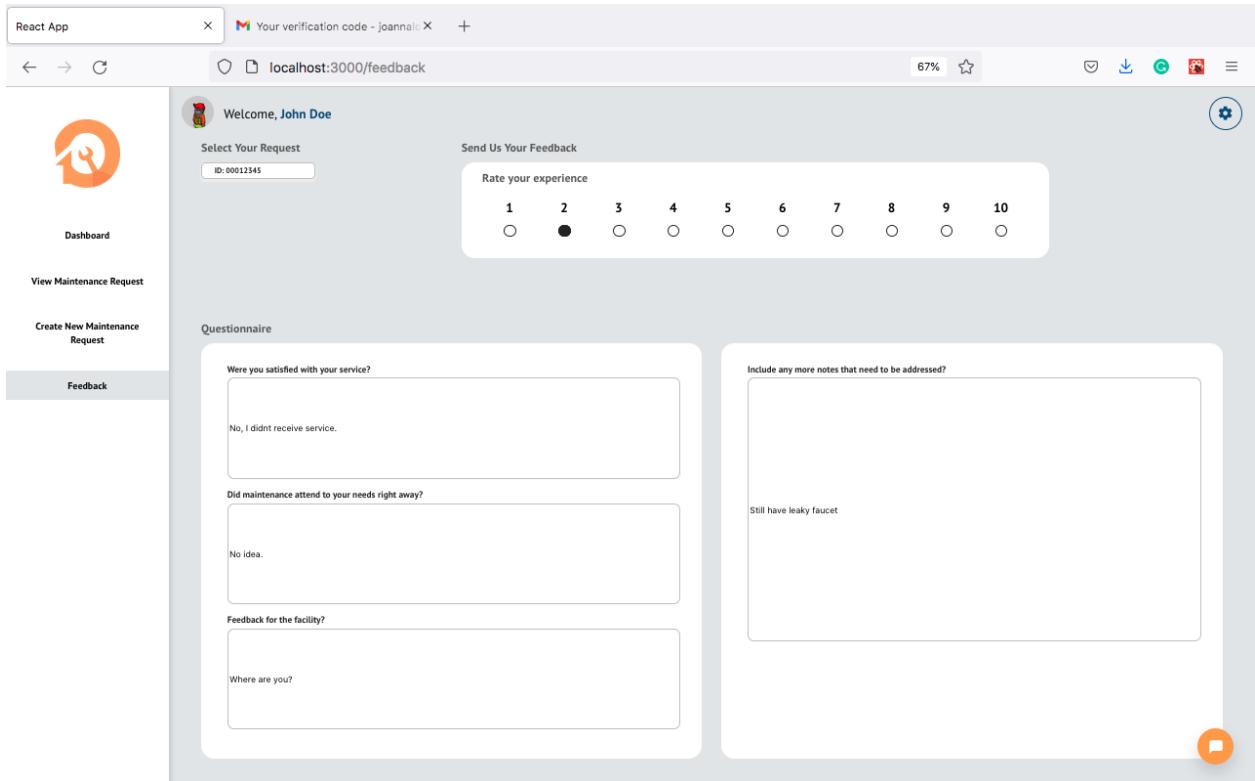


Fig. 4.5.2: This figure shows the functionality of the feedback page. The top left demonstrates that the user has selected the request they'd like to provide feedback for. The top right shows that the user selected a 2 out of 10 rating and the bottom half of the page shows the user answering the following questions.

4.6 Chatbot

The chatbot acts as an interactable tool for the users that can answer various questions. Currently, the chatbot is contained in a small window on the bottom-right corner of the screen that can minimize into an icon. When open, the user can interact with the bot via text through a textbox or pick a provided option given by the bot. Figures 4.6.1, 4.6.2, and 4.6.3 demonstrate how a user can interact with the Chatbot.

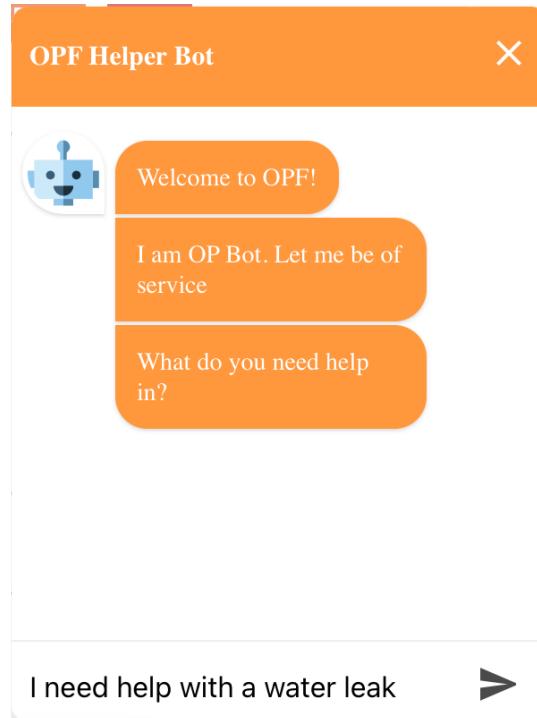


Fig. 4.6.1: This figure shows the first set of “steps” the chatbot would display to the user. The three messages are the default introduction messages the chatbot displays. In the input, the user can type input for the bot.

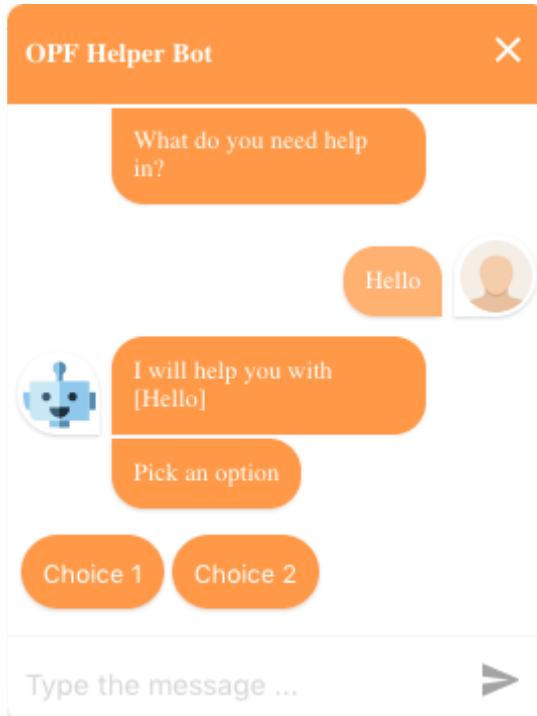


Fig. 4.6.2: This figure shows the message(s) the bot returns after the user types a message. The first message from the bot shows that the message was read from the bot. The second message leads to the bot displaying options for the user to pick.

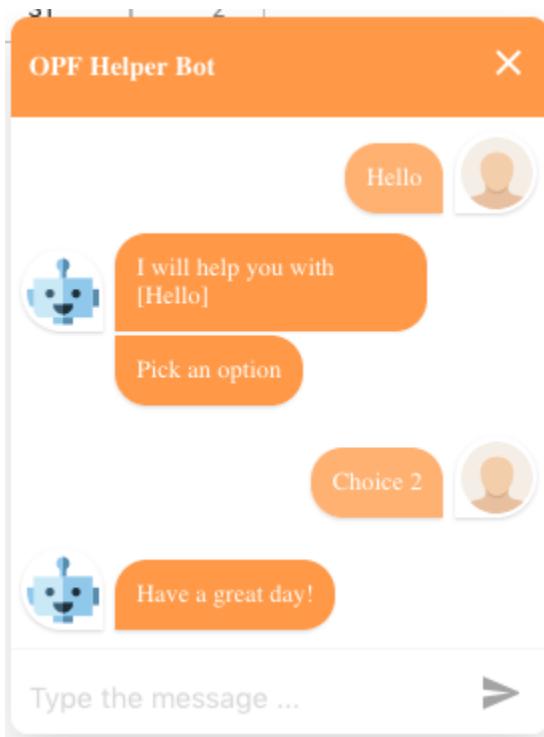


Fig. 4.6.3: This figure demonstrates after the user picks an option; the “Choice 2” option was chosen for this figure. The bot would create a message based on the option the user picks.

5. Demo Prototypes

5.1. Demo Prototype time

The date of the prototype demonstration is on December 6, 2021, from 1:00 pm -1:30 pm.

5.2. Summary of Comments and Recommendations

During the Demo, Team 05 received recommendations about how to improve their front-end design and also discussed the possible backend technologies with the instructors. In the View Maintenance Request page, the instructors recommended creating options for the users to download data in CSV files, text files, or other forms and make dynamic filter options for each header section and have a better range for the data. For the create Maintenance Request page, the instructors recommended having the option for the administrators to delegate tasks. Additionally, the pages, in general, can be more colorful to make the user interface more interesting.

Continuing onto the discussion for the backend technologies, Team 05 and the instructors discussed the Relational database and the Non-Relational database. Non-Relational is not suitable for Team 05’s project. As such, Relational SQL databases implemented with MySQL,

SQLite, or Postgres are recommended. As for the backend frameworks to connect the database with the website, Django and the Flask frameworks were discussed. Django includes a lot of file structure and features that are attractive to a project managing database. However, Flask provides better control and granularity of the API. Finally, Team 05 will need to examine the OPF structure and metadata to decide on the database system and backend technologies for OPF.

6. Team Contributions

6.1. Araam Zaremehrjardi's Contribution

Araam Zaremehrjardi's total time working on the project prototype totals 120+ hours. The number of hours worked on the prototype includes the 'create new maintenance request' page. The creation of universal components such as:

- Additional Notes
- Damage Location
- Page Query
- Request Building
- Request Date
- Request Description
- Request Media
- Request Priority
- Request Severity
- Request Tags
- Request Title
- Current User
- Header
- Header Widget
- Navigation

In addition, the realignment of CSS styling throughout the entire project and uploading the final project on Github.

6.2. Joanna Lopez's Contribution

Joanna Lopez's total time worked on the project prototype totals 24 hours. The number of hours worked on the prototype includes the 'dashboard' and the login page with AWS Cognito. This includes the creation of the following universal components such as:

- Log In
- Active Message
- Active Request
- Dash
- Post Widget

- ReadMe
- index.js
- App.js
- Pages:
 - CreateRequest
 - Dashboard
 - Feedback
 - ViewRequest

In addition, the calendar integration, creation of each respective web page i.e. Create Request, Dashboard, Feedback, and View Request. The routing of all of the javascript code to each respective page and the functionality to App.js. The integration of AWS Cognito into the OPF project for users to verify and log into the website. Lastly, the realignment and organization of code within the project folders and debugging of code as necessary when putting all respective Github branches together for the demonstration.

6.3. Melissa Flores' Contribution

Melissa Flores's total time worked on the project prototype totals 24 hours. The number of hours worked on the prototype includes the writing of the sections, Abstract, Develop Prototype, and the implementations of the feedback page which includes the creation of components such as:

- Drop Down question
- Simple text area for user input

In addition, overlooking and editing the entire document once each team member has completed their section. This includes adding any extra details and final touches, fixing the formatting, and making sure it looks organized and professional.

6.4. Nasrin Juana's Contribution

Nasrin Juana's total time worked on the project prototype totals 44 hours. The number of hours worked on the prototype includes the implementation of the Maintenance Requests page. This includes the creation of the following universal components such as:

- Columns
- Quick Filter
- Request Data
- Request Table
- Search
- Search Request

In addition, writing of the Prototype Objective and Functionality, Demo Prototype, and the introductions of the Dashboard page, the Student Maintenance page, and the Maintenance Requests page in the Develop prototype section are also included. Finally, overlooking and editing the entire document once each team member has completed their section.

6.5. Aisha Co's Contribution

Aisha Co's total time worked on the project prototype totals 18 hours. The number of hours worked on the prototype includes the implementation of the Chatbot. This includes the creation of the following universal components such as:

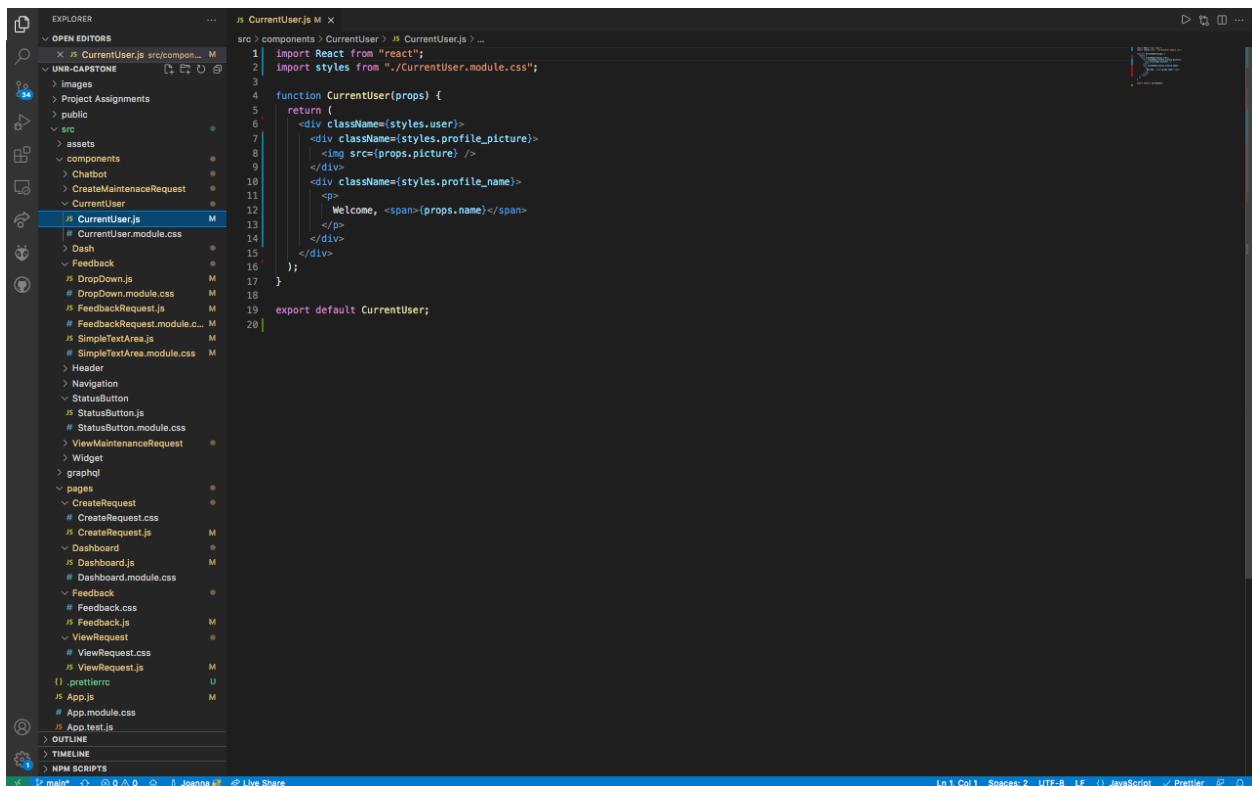
- Chatbot

In addition, writing the introduction, sections about the chatbot. Minor contributions include some editing on the Prototype Objectives & Functionality and Develop Prototype sections.

7. Implemented Code

The snapshots demonstrate the implemented code developed by each team member in Team 05. The screenshots display the files created by the team members which help indicate what parts of the code each of the team members implemented. In addition, a Github repository containing the implemented code is also included in this document. Finally, a zip file containing the developed code is submitted in the Project Part 4 assignment in canvas along with the document.

7.1. Araam Zaremehrjardi



A screenshot of a code editor (Visual Studio Code) showing the `CurrentUser.js` file. The file contains the following code:

```
import React from "react";
import styles from "./CurrentUser.module.css";

function CurrentUser(props) {
  return (
    <div className={styles.user}>
      <div className={styles.profile_picture}>
        <img src={props.picture} />
      </div>
      <div className={styles.profile_name}>
        <p> Welcome, <span>{props.name}</span> </p>
      </div>
    </div>
  );
}

export default CurrentUser;
```

The code defines a functional component `CurrentUser` that takes a `props` object. It returns a `div` element with a class of `user`, which contains a `div` with a class of `profile_picture` containing an `img` element with the `src` attribute set to `{props.picture}`, and another `div` with a class of `profile_name` containing a `p` element with the text "Welcome," followed by a `span` element with the value `{props.name}`.

Fig. 7.1.1: A snapshot demonstrating the `CurrentUser.js` file.

```

# CurrentUser.module.css M
src > components > CurrentUser > # CurrentUser.module.css > user
1  user {
2    display: flex;
3    flex-direction: row;
4  }
5   .profile_picture {
6     width: 50px;
7     align-self: center;
8   }
9   .profile_picture > img {
10    border-radius: 100%;
11    width: 50px;
12  }
13   .profile_name {
14    align-self: center;
15  }
16   .profile_name > p {
17    margin-left: 15px;
18    font-size: 20px;
19  }
20   .profile_name > p > span {
21    color: #d9dabf;
22    font-weight: 700;
23  }
24
25

```

Fig. 7.1.2: A snapshot demonstrating the CurrentUser.module.css file.

```

JS Header.js M
src > components > Header > # Header.js > Header
1 import React from "react";
2 import CurrentUser from "../CurrentUser/CurrentUser";
3 import profile_picture from "../../../../assets/profile_picture.png";
4 import HeaderWidget from "../../Widget/HeaderWidget/HeaderWidget";
5 import styles from "./Header.module.css";
6
7 function Header(props) {
8   return (
9     <div className={styles.header}>
10      <div className={styles.container}>
11        <CurrentUser picture={profile_picture} name="John Doe"></CurrentUser>
12
13        <div className={styles.widgets}>
14          <HeaderWidget icon="settings"></HeaderWidget>
15        </div>
16      </div>
17    </div>
18  );
19}
20
21 export default Header;
22

```

Fig. 7.1.3: A snapshot demonstrating the Header.js file.

```

.js Header.js M # Header.module.css M
src > components > Header > # Header.module.css > header
1 .header {
2   /* Styles for component positioning in App.js */
3   grid-column-start: 2;
4   grid-column-end: 16;
5   grid-row-start: 1;
6   grid-row-end: 2;
7   /* Styles for the component itself */
8   background: #e0e0e0;
9   display: flex;
10  flex-direction: column;
11  justify-content: center;
12 }
13 .container {
14   display: flex;
15   flex-direction: row;
16   margin-left: 20px;
17   margin-right: 20px;
18 }
19 .widgets {
20   flex-grow: 1;
21   display: flex;
22   justify-content: flex-end;
23   align-items: center;
24   gap: 10px;
25 }

```

Fig. 7.1.4: A snapshot showing the Header.module.css file.

```

.js Navigation.js M X
src > components > Navigation > Navigation.js ...
1 import React from "react";
2 import { NavLink } from "react-router-dom";
3 import logo from "./../assets/logo.png";
4 import styles from "./Navigation.module.css";
5
6 const Navigation = () => {
7   return (
8     <div className={styles.navigation}>
9       <div className={styles.logo}>
10         <img src={logo} />
11       </div>
12
13       <nav className={styles.nav}>
14         <NavLink
15           className={styles.NavLink}
16           to="/dashboard"
17           activeClassName={styles.selected}
18         >
19           <">
20           <p> Dashboard </p><">
21         </NavLink>
22         <NavLink
23           className={styles.NavLink}
24           to="/viewmaintenancerequest"
25           activeClassName={styles.selected}
26         >
27           <">
28           <p> View Maintenance Request </p><">
29         </NavLink>
30         <NavLink
31           className={styles.NavLink}
32           to="/createmaintenancerequest"
33           activeClassName={styles.selected}
34         >
35           <">
36           <p> Create New Maintenance Request </p><">
37         </NavLink>
38         <NavLink
39           className={styles.NavLink}
40           to="/feedback"
41           activeClassName={styles.selected}
42         >
43           <">
44           <p> Feedback </p><">
45         </NavLink>
46       </nav>
47     </div>
48   );
49 }
50
51 export default Navigation;

```

Fig. 7.1.5: A snapshot showing the Navigation.js file.

A screenshot of a code editor interface, likely VS Code, showing the `Navigation.module.css` file. The left sidebar shows a tree view of the project structure under the `OPEN EDITORS` tab, including files like `Header.js`, `Header.module.css`, `Navigation.js`, and `Navigation.module.css`. The right pane displays the CSS code for `Navigation.module.css`:

```
# Navigation.module.css
src > components > Navigation > # Navigation.module.css > navigation
1 .navigation {
2   /* Styles for component positioning in App.js */
3   position: relative;
4   z-index: 100;
5   grid-column-start: 1;
6   grid-column-end: 2;
7   grid-row-start: 1;
8   grid-row-end: 16;
9   /* Styles for the component itself */
10  display: flex;
11  flex-direction: column;
12  background: white;
13  box-shadow: 0 0 10px 2px #rgb(185, 185, 185);
14}
15
16 .logo {
17   display: flex;
18   justify-content: center;
19   margin-top: 50px;
20}
21 .logo > img {
22   width: 40%;
23   max-width: 120px;
24}
25 .nav {
26   margin-top: 50px;
27   display: flex;
28   flex-direction: column;
29}
30 .NavLink {
31   margin-bottom: 25px;
32   padding: 12px 0;
33   transition: 0.2s background;
34   display: block;
35   text-decoration: none;
36}
37 .NavLink:hover {
38   background: #e0e4e7;
39}
40 .NavLink > p {
41   width: 70%;
42   font-weight: 700;
43   font-size: 15px;
44   margin: 0 auto;
45   text-align: center;
46   color: black;
47   text-decoration: none;
48}
49 .selected {
50   background: #e0e4e7;
51}
```

Fig. 7.1.6: A snapshot showing the `Navigation.module.css` file.

A screenshot of a code editor interface, likely VS Code, showing the `StatusButton.module.css` file. The left sidebar shows a tree view of the project structure under the `OPEN EDITORS` tab, including files like `Header.js`, `Header.module.css`, `Navigation.js`, and `StatusButton.module.css`. The right pane displays the CSS code for `StatusButton.module.css`:

```
# StatusButton.module.css
src > components > StatusButton > # StatusButton.module.css > ...
1 .button > button {
2   padding: 8px;
3   font-size: 12px;
4   font-weight: 700;
5   border-radius: 10px;
6   border: 0px;
7   color: white;
8   background: #2e2e2e;
9   transition: 0.2s box-shadow ease-out;
10}
11
12 button:hover {
13   box-shadow: 0 2px 10px 0.2px #rgb(185, 185, 185);
14}
15
16 .colorless_button {
17   background: #c2c2c2;
18}
```

Fig 7.1.7: A snapshot showing the `StatusButton.module.css` file.

A screenshot of a code editor interface, likely Visual Studio Code, displaying the `HeaderWidget.js` file. The file content is as follows:

```
import React from "react";
import styles from "./HeaderWidget.module.css";

function HeaderWidget(props) {
  return (
    <div className={styles.widget}>
      | <span className="material-icons">{props.icon}</span>
    </div>
  );
}

export default HeaderWidget;
```

The left sidebar shows a project structure with various components, assets, and pages. The bottom status bar indicates the file is 13 lines long, uses 2 spaces, is in UTF-8 encoding, and is a JavaScript file.

Fig. 7.1.8: A snapshot showing the HeaderWidget.js file.

A screenshot of the Visual Studio Code interface. The left sidebar shows the 'EXPLORER' view with a tree of project files. The main editor area displays the contents of the 'Header.Widget.module.css' file. The code defines a CSS class '.widget' with properties like width, height, border, and color, and a hover state. The status bar at the bottom indicates the file is at line 21, column 1, with 2 spaces, and includes icons for Prettier and other settings.

```
.widget {
    width: 48px;
    height: 48px;
    display: flex;
    align-items: center;
    border: 2px solid #0d4b7f;
    border-radius: 100%;
    color: #0d4b7f;
    transition: 0.3s all ease-in-out;
}

.widget:hover {
    background-color: #0d4b7f;
    color: white;
}

.widget > span {
    margin: 0 auto;
    font-size: 25px;
}
```

Fig. 7.1.9: A snapshot showing the Header.Widget.module.css file.

A screenshot of the Visual Studio Code interface. The left sidebar shows the 'EXPLORER' view with a tree of project files. The main editor area displays the contents of the 'AdditionalNotes.js' file. It imports 'AdditionalNotes.css' and 'PageQuery'. The code defines a function 'AdditionalNotes' that returns a component containing a 'PageQuery' block with an 'Additional Notes' title and a text area. The status bar at the bottom indicates the file is at line 15, column 1, with 2 spaces, and includes icons for Prettier and other settings.

```
import './AdditionalNotes.css';
import PageQuery from './PageQuery/PageQuery';

function AdditionalNotes(props) {
    return (
        <PageQuery title="Additional Notes">
            <form className="additional_notes">
                <textarea rows="15"></textarea>
            </form>
        </PageQuery>
    );
}

export default AdditionalNotes;
```

Fig. 7.1.10: A snapshot showing the AdditionalNotes.js file.

The screenshot shows a code editor interface with the following details:

- EXPLORER** sidebar: Shows the project structure. The current file is `RequestBuilding.js` under the `src` directory of the `UNR-CAPSTONE` project.
- CODE EDITOR**: The main area displays the `RequestBuilding.js` file content:

```
js RequestBuilding.js M ×
src > components > CreateMaintenanceRequest > RequestBuilding > RequestBuilding.js > ...
1 import PageQuery from "./PageQuery/PageQuery";
2 import styles from "./RequestBuilding.module.css";
3
4 function RequestBuilding(props) {
5   return (
6     <PageQuery title="Building">
7       <form>
8         <textarea></textarea>
9       </form>
10      </PageQuery>
11    );
12  }
13
14 export default RequestBuilding;
15
```

The code defines a functional component `RequestBuilding` that wraps a `PageQuery` component with the title "Building". Inside the query, there is a simple form with a single text area.

STATUS BAR: Shows the file path (`RequestBuilding.js`), line number (15), column number (Col 1), spaces configuration (Spaces: 4), encoding (UTF-8 LF), language (JavaScript), and Prettier status.

Fig. 7.1.11: A snapshot showing the RequestBuilding.js file.

The screenshot shows a code editor interface with a dark theme. The left sidebar is the Explorer view, showing a tree structure of project files. The main area is the code editor, displaying the `DamageLocation.js` file. The code defines a `DamageLocation` component that returns a `<PageQuery>` component with a `<select>` dropdown menu for selecting a damage location. The dropdown has options for Bedroom, Living Room, Kitchen, and Bedroom again. The code editor status bar at the bottom indicates "Ln 23, Col 1" and "JavaScript".

```
js DamageLocation.js M ...
src > components > CreateMaintenanceRequest > DamageLocation > DamageLocation.js > ...
1 import './DamageLocation.css';
2 import PageQuery from './PageQuery/PageQuery';
3
4 function DamageLocation(props) {
5   return (
6     <PageQuery title="Location of damage">
7       <div className="damage_location">
8         <select name="Bedroom">
9           <option value="" selected disabled hidden>
10             Select
11           </option>
12           <option value="Bedroom">Bedroom</option>
13           <option value="Living Room">Living Room</option>
14           <option value="Kitchen">Kitchen</option>
15           <option value="Bedroom">Bedroom</option>
16         </select>
17       </div>
18     </PageQuery>
19   );
20 }
21
22 export default DamageLocation;
23 |
```

Fig 7.1.12: A snapshot showing the DamageLocation.js file.

The screenshot shows a code editor interface with a dark theme. The left sidebar is the Explorer view, showing a tree structure of project files. The main area is the code editor, displaying the `PageQuery.js` file. The code defines a `PageQuery` component that returns a `<div>` element with a title and content area. The code editor status bar at the bottom indicates "Ln 17, Col 1" and "JavaScript".

```
js PageQuery.js M ...
src > components > CreateMaintenanceRequest > PageQuery > PageQuery.js > ...
1 import React from "react";
2 import styles from "./PageQuery.module.css";
3
4 function PageQuery(props) {
5   return (
6     <div className={styles.query}>
7       <div className={styles.title}>
8         <p>{props.title}</p>
9       </div>
10      <div className={styles.content}>{props.children}</div>
11    </div>
12  );
13
14 }
15
16 export default PageQuery;
17 |
```

Fig. 7.1.13: A snapshot showing the PageQuery.js file.

A screenshot of a code editor interface, likely VS Code, showing the `PageQuery.module.css` file. The left sidebar shows a project structure with folders like `src`, `components`, and `pages`. The main editor area displays the following CSS code:

```
src > components > CreateMaintenanceRequest > PageQuery > # PageQuery.module.css > ...  
1 .query {  
2   margin: 15px auto;  
3   width: 80%;  
4 }  
5 .title p {  
6   font-size: 20px;  
7   font-weight: 700;  
8 }  
9 .content {  
10  margin-top: 5px !important;  
11 }  
12 }  
13 }
```

The status bar at the bottom indicates the file is 13 lines long, with 4 spaces per tab, in UTF-8 encoding, and has CSS syntax highlighting.

Fig 7.1.14: A snapshot showing the `PageQuery.module.css` file.

```

# RequestDate.module.css JS RequestDate.js M x
src > components > CreateMaintenanceRequest > RequestDate > RequestDate.js > RequestDate.module.css

1 | import PageQuery from './PageQuery/PageQuery';
2 | import styles from './RequestDate.module.css';
3 |
4 | function RequestDate(props) {
5 |   return (
6 |     <PageQuery title="Date Requested">
7 |       <div>
8 |         <select name="Day">
9 |           <option value="" selected disabled hidden>
10 |             Day
11 |           </option>
12 |           <option value="Monday">Monday</option>
13 |           <option value="Tuesday">Tuesday</option>
14 |           <option value="Wednesday">Wednesday</option>
15 |           <option value="Thursday">Thursday</option>
16 |           <option value="Friday">Friday</option>
17 |           <option value="Saturday">Saturday</option>
18 |           <option value="Sunday">Sunday</option>
19 |         </select>
20 |         <select name="Month">
21 |           <option value="" selected disabled hidden>
22 |             Month
23 |           </option>
24 |           <option value="January">January</option>
25 |           <option value="February">February</option>
26 |           <option value="March">March</option>
27 |           <option value="April">April</option>
28 |           <option value="May">May</option>
29 |           <option value="June">June</option>
30 |           <option value="July">July</option>
31 |           <option value="August">August</option>
32 |           <option value="September">September</option>
33 |           <option value="October">October</option>
34 |           <option value="November">November</option>
35 |           <option value="December">December</option>
36 |         </select>
37 |         <select name="Year">
38 |           <option value="" selected disabled hidden>
39 |             Year
40 |           </option>
41 |           <option value="2021">2021</option>
42 |           <option value="2022">2022</option>
43 |           <option value="2023">2023</option>
44 |           <option value="2024">2024</option>
45 |           <option value="2025">2025</option>
46 |           <option value="2026">2026</option>
47 |           <option value="2027">2027</option>
48 |         </select>
49 |       </div>
50 |     </PageQuery>
51 |   );
52 | }
53 |
54 | export default RequestDate;

```

Fig. 7.1.15: A snapshot showing the RequestDate.js file.

```

# RequestDescription.module.css JS RequestDescription.js M x
src > components > CreateMaintenanceRequest > RequestDescription > RequestDescription.js > RequestDescription.module.css

1 | import styles from './RequestDescription.module.css';
2 | import PageQuery from './PageQuery/PageQuery';
3 |
4 | function RequestDescription(props) {
5 |   return (
6 |     <PageQuery title="Request Description">
7 |       <form>
8 |         <textarea rows="15"></textarea>
9 |       </form>
10 |     </PageQuery>
11 |   );
12 | }
13 |
14 | export default RequestDescription;
15 |

```

Fig. 7.1.16: A snapshot showing the RequestDescription.js file.

A screenshot of the Visual Studio Code interface. The left sidebar shows the 'EXPLORER' view with a tree of files and folders. The current file is 'RequestMedia.js' located at 'src > components > CreateMaintenanceRequest > RequestMedia'. The code editor displays the following JavaScript code:

```
js RequestMedia.js M x
src > components > CreateMaintenanceRequest > RequestMedia > JS RequestMedia.js > ...
1 import styles from './RequestMedia.module.css';
2 import PageQuery from './PageQuery/PageQuery';
3
4 function RequestMedia(props) {
5   return (
6     <PageQuery title="Upload Images or Video">
7       <div className={styles.request_media}>
8         <p>Drop images or video here</p>
9       </div>
10    </PageQuery>
11  );
12}
13
14 export default RequestMedia;
15
```

The status bar at the bottom indicates 'Ln 15, Col 1 Spaces: 4 UTF-8 LF () JavaScript ✓ Prettier'.

Fig. 7.1.17: A snapshot showing the RequestMedia.js file.

A screenshot of the Visual Studio Code interface. The left sidebar shows the 'EXPLORER' view with a tree of files and folders. The current file is 'RequestMedia.module.css' located at 'src > components > CreateMaintenanceRequest > RequestMedia'. The code editor displays the following CSS code:

```
# RequestMedia.module.css M x
src > components > CreateMaintenanceRequest > RequestMedia > # RequestMedia.module.css > ...
1 .request_media {
2   width: 100px;
3   padding: 20px 0;
4   border: dashed 3px #fbfbfb;
5   border-radius: 10px;
6 }
7
8 .request_media > p {
9   color: #fbfbfb;
10  text-align: center;
11  font-weight: 700;
12  text-align: center;
13 }
14
```

The status bar at the bottom indicates 'Ln 14, Col 1 Spaces: 4 UTF-8 LF CSS ✓ Prettier'.

Fig. 7.1.18: A snapshot showing the RequestMedia.module.css file.

A screenshot of the Visual Studio Code interface. The left sidebar shows the 'EXPLORER' view with a tree structure of files and folders. In the center, the main editor window displays the code for 'RequestPriority.js'. The code defines a component named 'RequestPriority' that renders a 'PageQuery' component with a title 'Set Priority'. Inside the 'PageQuery', there is a series of nested 'div' elements with specific class names and styles for different priority levels: 'NONE', 'LOW', 'MEDIUM', and 'HIGH'. The code uses inline CSS and imports from 'RequestPriority.module.css'. The bottom status bar shows the file path 'src/components/CreateMaintenanceRequest/RequestPriority/RequestPriority.js', line 21, column 13, and other settings like 'Spaces: 4', 'UTF-8', and 'JavaScript'.

```
js RequestPriority.js M
src > components > CreateMaintenanceRequest > RequestPriority > RequestPriority.js > RequestPriority
1 import styles from './RequestPriority.module.css';
2 import PageQuery from './PageQuery/PageQuery';
3 import StatusButton from '../StatusButton/StatusButton';
4
5 function RequestPriority(props) {
6   return (
7     <PageQuery title="Set Priority">
8       <div className={styles.request_priority}>
9         <div className={styles.container}>
10          <StatusButton label="NONE" color="#c2c2c2"/>
11        </div>
12        <div className={styles.container}>
13          <StatusButton label="LOW" color="#628dbd"/>
14        </div>
15        <div className={styles.container}>
16          <StatusButton label="MEDIUM" color="#f59a7a"/>
17        </div>
18        <div className={styles.container}>
19          <StatusButton label="HIGH" color="#e17877"/>
20        </div>
21      </PageQuery>
22    );
23  }
24
25  export default RequestPriority;
```

Fig. 7.1.19: A snapshot showing the RequestPriority.js file.

A screenshot of the Visual Studio Code interface, similar to Fig. 7.1.19 but showing the 'RequestPriority.module.css' file. The code contains CSS rules for the 'request_priority' class, which includes 'display: inline-flex;' and '.container' with 'margin-right: 8px;'. The left sidebar shows the same file structure as Fig. 7.1.19. The bottom status bar shows the file path 'src/components/CreateMaintenanceRequest/RequestPriority/RequestPriority.module.css', line 7, column 1, and other settings like 'Spaces: 4', 'UTF-8', and 'CSS'.

```
# RequestPriority.module.css M
src > components > CreateMaintenanceRequest > RequestPriority > RequestPriority.module.css > ...
1 .request_priority {
2   display: inline-flex;
3 }
4 .container {
5   margin-right: 8px;
6 }
```

Fig. 7.1.20: A snapshot showing the RequestPriority.module.css file.

The screenshot shows a code editor interface with the following details:

- EXPLORER** sidebar: Shows a tree view of the project structure under the "UNR-CAPSTONE" folder. The "RequestSeverity.js" file is selected.
- EDITOR**: The main area displays the content of the "RequestSeverity.js" file. The code is as follows:

```
js RequestSeverity.js M ×
src > components > CreateMaintenanceRequest > RequestSeverity > RequestSeverity.js > RequestSeverity
1 import styles from './RequestSeverity.module.css';
2 import PageQuery from "./PageQuery/PageQuery";
3 import StatusButton from "../../StatusButton/StatusButton";
4
5 function RequestSeverity(props) {
6   return (
7     <PageQuery title="Set Severity">
8       <div className={styles.request_severity}>
9         <div className={styles.container}>
10          <StatusButton label="LOW" color="#628db0"/>
11        </div>
12        <div className={styles.container}>
13          <StatusButton label="MILD" color="#f59a7a"/>
14        </div>
15        <div className={styles.container}>
16          <StatusButton label="HIGH" color="#e17877"/>
17        </div>
18      </div>
19    </PageQuery>
20  );
21
22  export default RequestSeverity;
23
24 |
```

- STATUS BAR**: Shows "Ln 21, Col 2" and other settings like "Spaces: 4", "UTF-8", "LF", "JavaScript", and "Prettier".

Fig. 7.1.21: A snapshot showing the RequestSeverity.js file.

A screenshot of the Visual Studio Code interface. The left sidebar shows the 'EXPLORER' view with a tree of files under 'UNR-CAPSTONE'. The 'OPEN EDITORS' tab is active, showing the 'RequestSeverity.module.css' file. The code in the editor is:

```
# RequestSeverity.module.css M x
src > components > CreateMaintenanceRequest > RequestSeverity > # RequestSeverity.module.css > ...
1 .request_severity {
2   display: inline-flex;
3 }
4 .container {
5   margin-right: 8px;
6 }
```

Fig. 7.1.22: A snapshot showing the RequestSeverity.module.css file.

A screenshot of the Visual Studio Code interface. The left sidebar shows the 'EXPLORER' view with a tree of files under 'UNR-CAPSTONE'. The 'OPEN EDITORS' tab is active, showing the 'RequestTags.js' file. The code in the editor is:

```
js RequestTags.js M x
src > components > CreateMaintenanceRequest > RequestTags > JS RequestTags.js > ...
1 import styles from './RequestTags.module.css';
2 import PageQuery from './PageQuery/PageQuery';
3
4 function RequestTags(props) {
5   return (
6     <PageQuery title="Tags">
7       <form>
8         <textarea rows="6"></textarea>
9       </form>
10    </PageQuery>
11  );
12}
13
14 export default RequestTags;
```

Fig 7.1.23: A snapshot showing the RequestTags.js file.

```

js RequestTitle.js M x
src > components > CreateMaintenanceRequest > RequestTitle > RequestTitle.js > ...
1 import styles from './RequestTitle.module.css';
2 import PageQuery from './PageQuery/PageQuery';
3
4 function RequestTitle(props) {
5   return (
6     <PageQuery title="Request Title">
7       <form>
8         <textarea>/<textarea>
9       </form>
10    </PageQuery>
11  );
12}
13
14 export default RequestTitle;
15

```

Fig. 7.1.24: A snapshot showing the RequestTitle.js file.

```

js CreateMaintenanceRequest.js M x
src > components > CreateMaintenanceRequest > CreateMaintenanceRequest.js > (e) default
1 import React, { Component } from "react";
2 import styles from "./CreateMaintenanceRequest.module.css";
3 import RequestTitle from "./RequestTitle/RequestTitle";
4 import RequestDescription from "./RequestDescription/RequestDescription";
5 import RequestSeverity from "./RequestSeverity/RequestSeverity";
6 import RequestBuilding from "./RequestBuilding/RequestBuilding";
7 import RequestMedia from "./RequestMedia/RequestMedia";
8 import RequestPriority from "./RequestPriority/RequestPriority";
9 import AdditionalNotes from "./AdditionalNotes/AdditionalNotes";
10 import RequestTags from "./RequestTags/RequestTags";
11
12 class CreateMaintenanceRequest extends Component {
13   render() {
14     return (
15       <div className={styles.create_maintenance_request}>
16         <div className={styles.page_title}>
17           <p>Submit Maintence Request</p>
18         </div>
19
20         <div className={styles.content}>
21           <div className={styles.left_pane}>
22             <RequestTitle>/<RequestTitle>
23             <RequestDescription>/<RequestDescription>
24             <RequestSeverity>/<RequestSeverity>
25             <RequestBuilding>/<RequestBuilding>
26           </div>
27
28           <div className={styles.right_pane}>
29             <RequestMedia>/<RequestMedia>
30             <RequestPriority>/<RequestPriority>
31             <AdditionalNotes>/<AdditionalNotes>
32             <RequestTags>/<RequestTags>
33           </div>
34         </div>
35       );
36     }
37   }
38 }
39
40 export default CreateMaintenanceRequest;
41

```

Fig. 7.1.25: A snapshot showing the CreateMaintenanceRequest.js file.

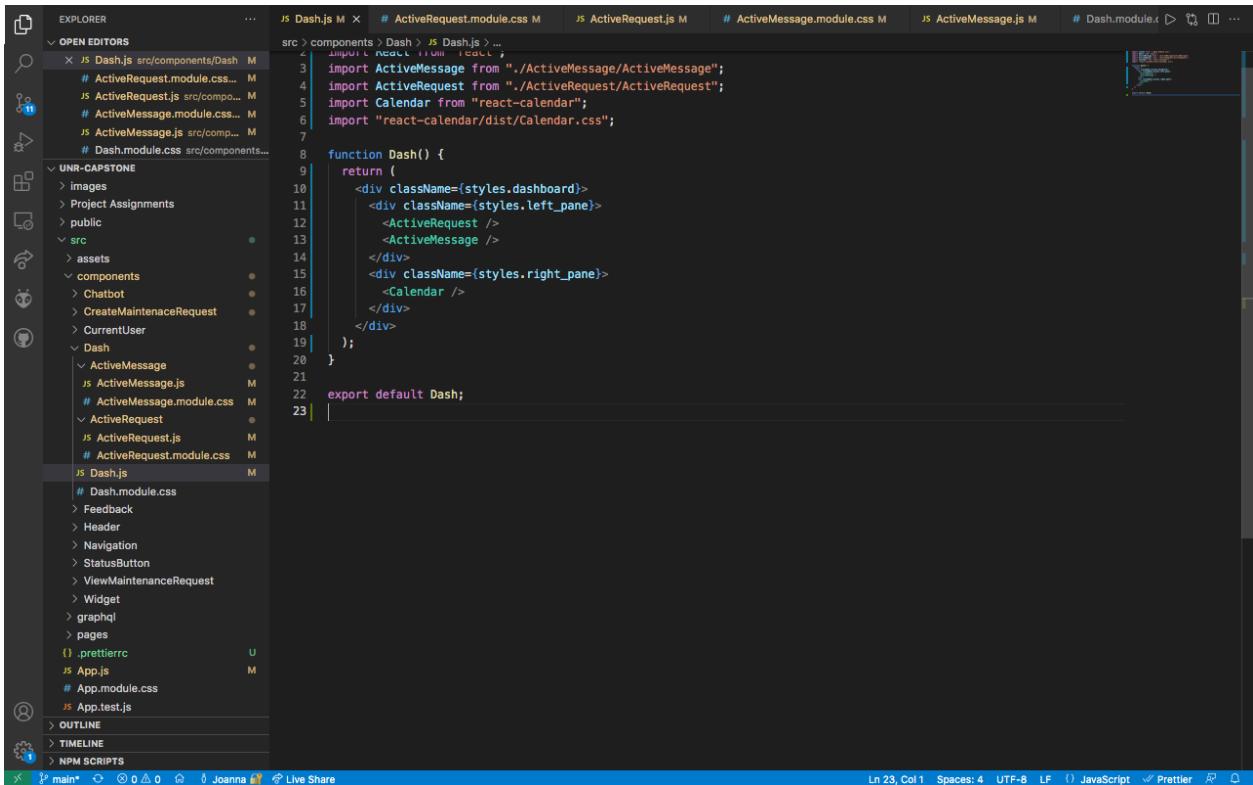
The screenshot shows a code editor interface with the following details:

- OPEN EDITORS:** One editor is open, displaying the file `# CreateMaintenanceRequest.module.css`.
- EXPLORER:** A sidebar showing the project structure. The current folder is `src > components > CreateMaintenanceRequest`. Inside, there are several files and folders:
 - `DamageLocation.js`
 - `DamageLocation.css`
 - `PageQuery.js`
 - `PageQuery.module.css`
 - `RequestBuilding.js`
 - `RequestBuilding.module.css`
 - `RequestDate.js`
 - `RequestDate.module.css`
 - `RequestDescription.js`
 - `RequestDescription.module.css`
 - `RequestMedia.js`
 - `RequestMedia.module.css`
 - `RequestPriority.js`
 - `RequestPriority.module.css`
 - `RequestSeverity.js`
 - `RequestSeverity.module.css`
 - `RequestTags.js`
 - `RequestTags.module.css`
 - `RequestTitle.js`
 - `RequestTitle.module.css`
 - `CreateMaintenanceRequest...` (selected)
 - `CurrentUser`
 - `Dash`
 - `Feedback`
 - `Header`
 - `Navigation`
 - `Statusbutton`
 - `ViewMaintenanceRequest`
 - `Widget`
 - `HeaderWidget`
 - `HeaderWidget.js`
 - `HeaderWidget.module.css`
- OUTLINE:** Shows the outline of the selected file, including sections like `content`, `left_pane`, `right_pane`, and `textarea`.
- TIMELINE:** Shows the history of changes made to the file.
- NPM SCRIPTS:** Shows the available npm scripts.
- STATUS BAR:** Shows the current line (Ln 16), column (Col 11), and encoding (Spaces: 4, UTF-8). It also includes buttons for LF, CSS, and Prettier.

```
# CreateMaintenanceRequest.module.css 1
src > components > CreateMaintenanceRequest > # CreateMaintenanceRequest.module.css > ↵ textarea
  1: .create_maintenance_request {
  2:   ...
  3:   .content {
  4:     display: flex;
  5:   }
  6:
  7:   .left_pane {
  8:     flex-grow: 1;
  9:     flex-basis: 0;
 10:   }
 11:
 12:   .right_pane {
 13:     flex-grow: 1;
 14:     flex-basis: 0px;
 15:   }
 16:
 17:   textarea {
 18:     appearance: none;
 19:     border: 2px solid #bfbfbf;
 20:     resize: none;
 21:     width: 100%;
 22:     border-radius: 10px;
 23:     transition: 0.2s all ease-in-out;
 24:   }
 25:
 26:   textarea:focus {
 27:     box-shadow: 0 2px 10px 0.2px #rgb(185, 185, 185);
 28:     border-color: black;
 29:     outline: none;
 30:   }
 31:   textarea:hover {
 32:     box-shadow: 0 2px 10px 0.2px #rgb(185, 185, 185);
 33:   }
 34:   select {
 35:     outline: none;
 36:     -webkit-appearance: none !important;
 37:     background: #fffffe;
 38:     font-family: "PT Sans", sans-serif;
 39:     font-weight: 700;
 40:     width: 130px;
 41:     height: 30px;
 42:     font-size: 12px;
 43:     border-radius: 12px;
 44:     border: 2px solid #bfbfbf;
 45:     padding: 0 15px;
 46:     color: #777777;
 47:     box-shadow: #eeeeee 0 3px 4px 1px;
 48:     margin-right: 8px;
 49:   }
 50:
 51:   .page_title {
 52:     margin: 15px auto 0 60px;
 53:   }
 54:
 55: 
```

Fig. 7.1.26: A snapshot showing the CreateMaintenanceRequest.css file.

7.2. Joanna Lopez



A screenshot of a code editor interface, likely VS Code, displaying the `Dash.js` file. The code editor shows the following content:

```
src > components > Dash > JS Dash.js > ...
1 import React from 'react';
2 import ActiveMessage from './ActiveMessage/ActiveMessage';
3 import ActiveRequest from './ActiveRequest/ActiveRequest';
4 import Calendar from "react-calendar";
5 import "react-calendar/dist/Calendar.css";
6
7 function Dash() {
8     return (
9         <div className={styles.dashboard}>
10            <div className={styles.left_pane}>
11                <ActiveRequest />
12                <ActiveMessage />
13            </div>
14            <div className={styles.right_pane}>
15                <Calendar />
16            </div>
17        </div>
18    );
19 }
20
21
22 export default Dash;
23 |
```

The left sidebar shows the project structure under the `src` directory, including files like `Dashboard.js`, `ActiveMessage.js`, and `ActiveRequest.js`. The bottom status bar indicates the file is 23 lines long, 4 spaces wide, and uses LF line endings.

Fig. 7.2.1: A snapshot showing the Dash.js file.

```

# Dash.module.css M
src > components > Dash > # Dash.module.css > ...
1 .dashboard {
2   margin: 25px auto 0 0;
3   display: flex !important;
4 }
5 .left_pane {
6   flex-grow: 1 !important;
7   margin: 0 50px;
8   flex-basis: 0;
9 }
10 .right_pane {
11   flex-grow: 1 !important;
12   flex-basis: 0;
13 }
14
15

```

Fig. 7.2.2: A snapshot showing the Dash.module.css file.

```

# ActiveMessage.module.css M JS ActiveMessage.js # ActiveMessage.module.css M JS ActiveMessage.js M X
src > components > Dash > ActiveMessage > JS ActiveMessage.js ...
1 import styles from './ActiveMessage.module.css';
2 import StatusButton from '../../../../../StatusButton/StatusButton';
3 import PostWidget from '../../../../../Widget/PostWidget/PostWidget';
4
5 function ActiveMessage() {
6   return (
7     <div className={styles.activemessage}>
8       <div className="header">
9         <div>
10          | <PostWidget />
11        </div>
12        <div className={styles.title}>
13          | <p>Facility</p>
14        </div>
15
16       <div className={styles.messagedescription}>
17         <p>
18           To ensure the safety of our community over the break, all rooms will
19           be checked to verify the above list has been completed. Room
20           inspections will also be conducted by our Facilities team to ensure
21           health, safety, and security; as well as changing air filters,
22           checking drains, inspecting the cleanliness of bathroom, etc.
23         </p>
24       </div>
25
26       <div className={styles.footer}>
27         <StatusButton label="Reply Now" color="#d62013"/></StatusButton>
28       </div>
29     </div>
30   );
31
32   export default ActiveMessage;
33
34

```

Fig. 7.2.3: A snapshot showing the ActiveMessages.js file.

The screenshot shows a code editor interface with the ActiveMessage.module.css file open. The left sidebar displays a project structure with various components and assets. The main editor area contains the following CSS code:

```
# ActiveMessage.module.css M x
src > components > Dash > ActiveMessage > # ActiveMessage.module.css > .messagedescription

1 .activemessage {
2   border-radius: 12px;
3   border: 2px solid #bfbfbf;
4   background-color: white;
5   padding: 10px;
6   margin-top: 25px;
7 }

8 .title {
9   color: black;
10  font-size: 28px;
11  font-weight: 700;
12}

13 .messagedescription {
14   margin-top: 10px;
15  font-size: 16px;
16  color: black;
17  background: #e6eef0;
18  padding: 8px 8px;
19  border-radius: 3px;
20}

21 .footer {
22  margin-top: 10px;
23  flex: right;
24  margin-bottom: 10px;
25}

26 .prettier {
27  App.js
28}
```

Fig. 7.2.3: A snapshot showing the ActiveMessage.module.css file.

```

js ActiveRequest.js M x
src > components > Dash > ActiveRequest > ActiveRequest.js > ActiveRequest
1 | import styles from "./ActiveRequest.module.css";
2 | import StatusButton from "../../StatusButton/StatusButton";
3 |
4 | function ActiveRequest() {
5 |   return (
6 |     <div className={styles.activerequest}>
7 |       <div className={styles.requesttitle}>
8 |         <p>Faucet Leaking!</p>
9 |         <p>Submitted December 1, 2021</p>
10 |       </div>
11 |
12 |       <div className={styles.requestdescription}>
13 |         <p>Issue Description:</p>
14 |         <p>I have a faucet leak where it's almost to a constant stream of water when its coming out when I use it. When I close the faucet after I use it, it also leaks constantly.</p>
15 |       </div>
16 |
17 |       <div className={styles.issuetag}>
18 |         <p>Issue Tags:</p>
19 |         <div className={styles.requesttitle}>
20 |           <StatusButton label="Water" color="#f59a7a"/></StatusButton>
21 |           <StatusButton label="Faucet Leak" color="#f59a7a"/></StatusButton>
22 |         </div>
23 |       </div>
24 |
25 |     </div>
26 |
27 |   );
28 |
29 | }
30 |
31 | export default ActiveRequest;
32 |
33 |

```

Fig. 7.2.4: A snapshot showing the ActiveRequest.js file.

```

# ActiveRequest.module.css M x js ActiveRequest.js M # ActiveMessage.module.css M
src > components > Dash > ActiveRequest > ActiveRequest.module.css > activerequest.p
1 | .activerequest {
2 |   border-radius: 12px;
3 |   border: 2px solid #fbfbfb;
4 |   background-color: white;
5 |   padding: 10px;
6 |
7 |   .activerequest * {
8 |     margin: 0;
9 |
10 |     .activerequest p {
11 |       color: black;
12 |       text-align: left;
13 |       margin-left: 10px;
14 |
15 |       .requesttitle {
16 |         margin-bottom: 10px;
17 |       }
18 |
19 |       .requesttitle p:nth-child(1) {
20 |         font-size: 20px;
21 |         font-weight: 700;
22 |
23 |         .requesttitle p:nth-child(2) {
24 |           font-size: 13px;
25 |
26 |           .requestdescription p:nth-child(1) {
27 |             font-size: 20px;
28 |             color: black;
29 |             font-weight: 700;
30 |
31 |             .issuetag {
32 |               margin-top: 10px;
33 |               margin-bottom: 10px;
34 |               color: black;
35 |               font-size: 20px;
36 |               font-weight: 700;
37 |
38 |             }
39 |
40 |           }
41 |
42 |         }
43 |
44 |

```

Fig. 7.2.5: A snapshot showing the Active.module.css file.

A screenshot of the Visual Studio Code interface. The left sidebar shows a project structure under 'UNR-CAPSTONE' with 'src' as the active folder. Inside 'src', there are several components like Chatbot, CreateMaintenanceRequest, CurrentUser, Dash, Feedback, Header, Navigation, Statusbutton, ViewMaintenanceRequest, Widget, and GraphQL. Below these are files for testing and configuration: App.js, App.module.css, App.test.js, aws-exports.js, index.css, index.js, and setupTests.js. There are also .gitignore, package-lock.json, package.json, and README.md files. The right pane displays the content of the 'App.js' file, which defines a functional component 'App' that returns a JSX structure with a header, navigation, and chatbot components, followed by a switch statement for different routes including '/' (redirecting to 'dashboard'), '/dashboard' (displaying a dashboard component), '/createmaintenancerequest' (displaying a CreateMaintenanceRequest component), '/viewmaintenancerequest' (displaying a ViewMaintenanceRequest component), and '/feedback' (displaying a Feedback component). The code uses 'withAuthenticator' from '@aws-amplify/ui-react'. The status bar at the bottom indicates the file is 17 lines long, 33 columns wide, with 2 spaces, using LF line endings, and is in JavaScript mode with Prettier enabled.

```
//aws
import { withAuthenticator } from "@aws-amplify/ui-react";
//app
import { Route, Switch, Redirect } from "react-router-dom";
import Dashboard from "./components/Dash/Dash";
import CreateMaintenanceRequest from "./components/CreateMaintenanceRequest/CreateMaintenanceRequest";
import ViewMaintenanceRequest from "./components/ViewMaintenanceRequest/ViewMaintenanceRequest";
import Feedback from "./components/Feedback/FeedbackRequest";
import Header from "./components/Header/Header";
import Navigation from "./components/Navigation/Navigation";
import Chatbot from "./components/Chatbot/Chatbot";
import styles from "./App.module.css";

function App() {
  return (
    <div className={styles.app}>
      <Header className={styles.header} />
      <Navigation className={styles.navigation} />
      <Chatbot />

      <div className={styles.section}>
        <Switch>
          <Route path="/" exact>
            <Redirect to="/dashboard" />
          </Route>

          <Route path="/dashboard" exact>
            <Dashboard />
          </Route>

          <Route path="/createmaintenancerequest">
            <CreateMaintenanceRequest />
          </Route>

          <Route path="/viewmaintenancerequest">
            <ViewMaintenanceRequest />
          </Route>

          <Route path="/feedback">
            <Feedback />
          </Route>
        </Switch>
      </div>
    </div>
  );
}

export default withAuthenticator(App);
```

Fig. 7.2.6: A snapshot showing the App.js file.

A screenshot of the Visual Studio Code interface, similar to Fig. 7.2.6 but showing the 'index.js' file instead. The left sidebar shows the same project structure. The right pane displays the content of 'index.js', which imports ReactDOM and BrowserRouter from 'react-dom', and App from './App'. It then imports Amplify from 'aws-amplify' and awsExports from './aws-exports'. Finally, it calls Amplify.configure(awsExports) and ReactDOM.render() with a BrowserRouter component containing an App component and a root element. The status bar at the bottom indicates the file is 16 lines long, 33 columns wide, with 2 spaces, using LF line endings, and is in JavaScript mode with Prettier enabled.

```
import ReactDOM from 'react-dom';
import { BrowserRouter } from 'react-router-dom';
import App from './App';

import Amplify from "aws-amplify";
import awsExports from "./aws-exports";
Amplify.configure(awsExports);

ReactDOM.render(
  <BrowserRouter>
    <App />
  </BrowserRouter>,
  document.getElementById('root')
);
```

Fig. 7.2.7: A snapshot showing the index.js file.

A screenshot of the Visual Studio Code interface. The left sidebar shows the 'EXPLORER' view with a tree structure of files and folders. In the center, the main editor pane displays the code for 'CreateRequest.js'. The code imports React and CreateMaintenanceRequest from other components. It defines a 'CreateRequest' function that returns a component with a className of 'app' containing a 'CreateMaintenanceRequest' component. The code concludes with an export statement. The bottom status bar shows file details like 'Ln 14, Col 1' and file statistics like 'Spaces: 4'.

```
js CreateRequest.js M x
src > pages > CreateRequest > CreateRequest.js > ...
1 import React from "react";
2 import CreateMaintenanceRequest from "../../../../components/CreateMaintenanceRequest/CreateMaintenanceRequest";
3 import "./CreateRequest.css";
4
5 const CreateRequest = () => {
6   return (
7     <div className="app">
8       | <CreateMaintenanceRequest />
9     </div>
10   );
11 }
12
13 export default CreateRequest;
14 |
```

Fig. 7.2.8: A snapshot showing the CreateRequest.js file.

A screenshot of the Visual Studio Code interface, similar to Fig. 7.2.8. The left sidebar shows the 'EXPLORER' view with a tree structure of files and folders. In the center, the main editor pane displays the code for 'ViewRequest.js'. The code imports React and ViewMaintenanceRequest from other components. It defines a 'ViewRequest' function that returns a component with a className of 'TestPage' containing a 'ViewMaintenanceRequest' component. The code concludes with an export statement. The bottom status bar shows file details like 'Ln 13, Col 1' and file statistics like 'Spaces: 2'.

```
js CreateRequest.js M > ViewRequest.js M x
src > pages > ViewRequest > ViewRequest.js > ...
1 import React from "react";
2 import ViewMaintenanceRequest from "../../../../components/ViewMaintenanceRequest/ViewMaintenanceRequest";
3
4 const ViewRequest = () => {
5   return (
6     <div className="TestPage">
7       | <ViewMaintenanceRequest />
8     </div>
9   );
10 }
11
12 export default ViewRequest;
13 |
```

Fig. 7.2.9: A snapshot showing the viewRequest.js file.

```

# CS 425/426: Senior Capstone Project

The University of Nevada, Reno's (UNR) dormitory maintenance requests system presents itself as a problem in need of improvement ridden with inconveniences and flaws to dormitory students. The intended audience of Optimum Property Fix (OPF) are dormitory students at UNR and facilities services for dormitories. OPF is a planned approach to solving this problem through web technologies and advancements in Artificial Intelligence.

## Documentation

[Project Documentation](https://github.com/joannalopez223/UNR-Capstone/tree/main/Project%20Assignments) - The accompanied project assignment outline requirements and specifications including technology justifications throughout the year for the University of Nevada, Reno senior capstone project, OPF.

## Features

- Light/dark mode toggle
- Fullscreen mode
- Cross platform

## Skills

- Javascript
- HTML
- CSS

## Tech Stack

**Client:** React

## Screenshots

![Sign In Screen](images/Screenshot1.png)
![Dashboard Screen](images/Screenshot2.png)
![Maintenance Request Dashboard](images/Screenshot3.png)
![Feedback Screen](images/Screenshot4.png)

## Demo

Insert gif or link to demo

## Authors

Joanna

```

Fig. 7.2.10: A snapshot showing the README.md file.

```

import React from "react";
import FeedbackRequest from "../../components/Feedback/FeedbackRequest";

const Feedback = () => {
  return (
    <div className="feedback">
      | <FeedbackRequest />
    </div>
  );
};

export default Feedback;

```

Fig 7.2.11: A snapshot showing the Feeback.js file.

The screenshot shows a code editor interface with a dark theme. On the left is the Explorer sidebar, which lists the project structure. The current file, `PostWidget.js`, is open in the main editor area. The code in the editor is as follows:

```
src > components > Widget > PostWidget > PostWidget.js
1 import React from "react";
2 import styles from "../PostWidget/PostWidget.module.css";
3
4 function PostWidget(props) {
5   return (
6     <div className={styles.icon}>
7       | <span className="material-icons">account_circle</span>
8     </div>
9   );
10 }
11
12 export default PostWidget;
13 |
```

The Explorer sidebar shows the following directory structure:

- UNR-CAPSTONE
- src
 - assets
 - components
 - Chatbot
 - CreateMaintenanceRequest
 - CurrentUser
 - Dash
 - ActiveMessage
 - ActiveMessage.js
 - ActiveMessage.module.css
 - ActiveRequest
 - ActiveRequest.js
 - ActiveRequest.module.css
 - Dash.js
 - Dash.module.css
 - Feedback
 - Header
 - Navigation
 - Statusbutton
 - ViewMaintenanceRequest
 - Widget
 - HeaderWidget
 - HeaderWidget.js
 - HeaderWidget.module.css
 - PostWidget
 - PostWidget.js
 - PostWidget.module.css
 - graphql
 - pages
 - CreateRequest
 - CreateRequest.css
 - CreateRequest.js
 - Dashboard
 - Dashboard.js
 - Dashboard.module.css
 - Feedback
 - Feedback.css
 - Feedback.js
 - ViewRequest
 - .prettierrc
 - App.js
 - App.module.css
 - App.test.js

Fig. 7.2.12: A snapshot showing the PostWidget.js file.

The screenshot shows a code editor interface with the following details:

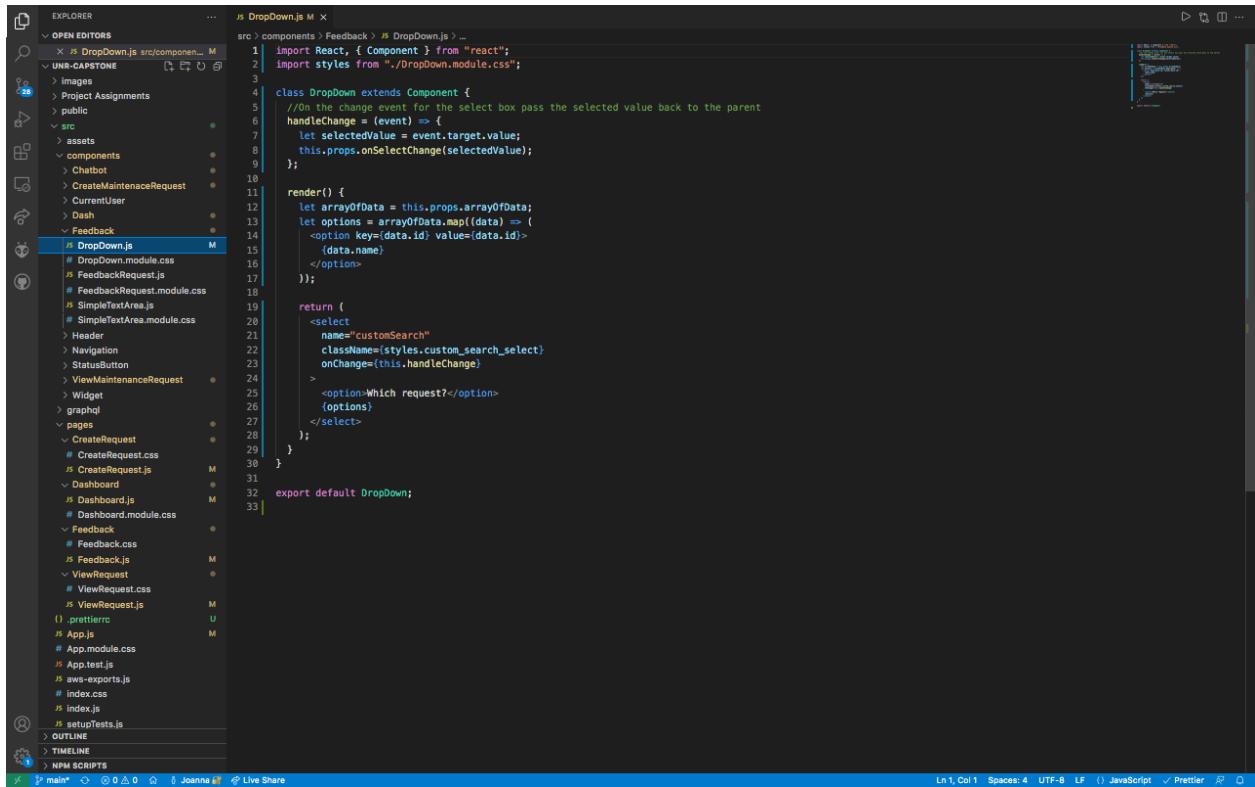
- EXPLORER** sidebar: Shows the project structure with files like `PostWidget.module.css`, `ActiveMessage.js`, `ActiveRequest.js`, `Dash.js`, `HeaderWidget.js`, `PostWidget.js`, `CreateRequest.js`, `Dashboard.js`, `Feedback.js`, `App.js`, and `App.module.css`.
- PostWidget.module.css** file content (highlighted in the editor):

```
# PostWidget.module.css M ...
src > components > Widget > PostWidget > # PostWidget.module.css > ...
1 .icon {
2   align-self: flex-start;
3   display: inline-block;
4 }
5
6 .icon > span {
7   font-size: 60px;
8 }
```

The code editor also displays status bar information: Ln 9, Col 1, Spaces: 4, UTF-8, LF, CSS, Prettier.

Fig. 7.2.13: A snapshot showing the PostWidget.module.css file.

7.3. Melissa Flores



A screenshot of a code editor (Visual Studio Code) displaying the `DropDown.js` file. The file is located in the `src/components/Feedback` directory. The code is a React component named `DropDown` that takes an array of data and a handle change function. It maps the data to options and returns a select element with the options. The code editor shows syntax highlighting for JavaScript and CSS files.

```
OPEN EDITORS
src > components > Feedback > DropDown.js > ...
src > components > Feedback > DropDown.js
1 import React, { Component } from "react";
2 import styles from "./DropDown.module.css";
3
4 class DropDown extends Component {
5   //On the change event for the select box pass the selected value back to the parent
6   handleChange = (event) => {
7     let selectedValue = event.target.value;
8     this.props.onSelectChange(selectedValue);
9   };
10
11   render() {
12     let array0fData = this.props.array0fData;
13     let options = array0fData.map((data) => (
14       <option key={data.id} value={data.id}>
15         {data.name}
16       </option>
17     ));
18
19     return (
20       <select
21         name="customSearch"
22         className={styles.custom_search_select}
23         onChange={(this.handleChange)}
24       >
25         <option>Which request?</option>
26         {options}
27       </select>
28     );
29   }
30 }
31
32 export default DropDown;
```

Fig. 7.3.1: A snapshot showing the `Dropdown.js` file.

```

.js DropDown.js M # DropDown.module.css M x
src > components > Feedback > # DropDown.module.css > custom_search_select
1 .custom_search_select {
2   border-radius: 5px;
3   width: 168px;
4   font-weight: 900;
5   text-align: left;
6   font-size: 12px;
7   color: #rgb(37, 36, 36);
8   width: 175px;
9   height: 25px;
10 }
11

```

The screenshot shows the VS Code interface with the Explorer sidebar open, displaying a project structure for a UNR-CAPSTONE application. The current file being edited is `DropDown.module.css`. The code defines a CSS class `.custom_search_select` with various styling rules.

Fig. 7.3.2: A snapshot showing the DropDown.module.css file.

```

.js FeedbackRequest.js M x
src > components > Feedback > FeedbackRequest.js ...
1 import React, { Component } from "react";
2 import styles from "./FeedbackRequest.module.css";
3 import DropDown from "./DropDown";
4 import SimpleTextArea from "./SimpleTextArea";
5
6 const arrayOfData = [
7   {
8     id: "1",
9     name: "ID: 00012345",
10    },
11   {
12     id: "2",
13     name: "ID: 00023456",
14    },
15   {
16     id: "3",
17     name: "ID: 00034567",
18    },
19  ];
20
21 class FeedbackRequest extends Component {
22   handleSelectChange = (selectedValue) => {
23     this.setState({
24       selectedValue: selectedValue,
25     });
26   };
27
28 render() {
29   return (
30     <div className={styles.feedback}>
31       <div className={styles.feedback_container}>
32         <p>
33           className={`${styles.section_title} ${styles.request_selection_title}`}
34         >
35           Select Your Request
36         </p>
37         <div>
38           className={`${styles.section_container} ${styles.request_selection_container}`}
39         >
40           <div>
41             <DropDown
42               arrayOfData={arrayOfData}
43               onSelectChange={this.handleSelectChange}
44             ></DropDown>
45           </div>
46         </div>
47         <p>
48           className={`${styles.section_title} ${styles.send_feedback_title}`}
49         >
50           Send Us Your Feedback
51         </p>
52       </div>
53     </div>
54   );
55 }
56
57 export default FeedbackRequest;

```

The screenshot shows the VS Code interface with the Explorer sidebar open, displaying a project structure for a UNR-CAPSTONE application. The current file being edited is `FeedbackRequest.js`. The code defines a React component `FeedbackRequest` that includes a dropdown menu and a button to send feedback.

Fig. 7.3.3: A snapshot showing the FeedbackRequest.js file.

```

# FeedbackRequest.module.css M ...
src > components > Feedback > # FeedbackRequest.module.css > #feedback
1 .feedback {
2   background: #e0e4e7;
3   height: 100%;
4   overflow-y: scroll;
5 }
6 .feedback_container {
7   display: grid;
8   grid-template-columns: repeat(12, 1fr);
9   grid-template-rows: auto repeat(1, 1fr) auto repeat(2, 1fr) auto;
10  margin: 0 50px;
11  height: 120px;
12  column-gap: 30px;
13 }
14 .request_selection_title {
15   grid-column-start: 1;
16   grid-column-end: 3;
17   grid-row-start: 1;
18   grid-row-end: 2;
19 }
20 .request_selection_container {
21   grid-column-start: 1;
22   grid-column-end: 3;
23   grid-row-start: 2;
24   grid-row-end: 3;
25 }
26 .send_feedback_title {
27   grid-column-start: 4;
28   grid-column-end: 11;
29   grid-row-start: 1;
30   grid-row-end: 2;
31 }
32 .experience_rate_container {
33   grid-column-start: 4;
34   grid-column-end: 11;
35   grid-row-start: 2;
36   grid-row-end: 3;
37 }
38 .experience_rate {
39   background: white;
40   border-radius: 20px;
41   padding: 10px;
42 }
43 .section_subtext {
44   font-weight: 700;
45   font-size: 18px;
46   color: #rgb(88, 88, 88);
47   width: 95%;
48   max-width: 900px;
49   margin: 0 auto;
50 }
51 .content_experience_rate_form {

```

Fig. 7.3.4: A snapshot showing the FeedbackRequest.module.css file.

```

JS SimpleTextArea.js M ...
src > components > Feedback > JS SimpleTextArea.js > ...
1 import React, { Component } from "react";
2 import styles from "./SimpleTextArea.module.css";
3
4 class SimpleTextArea extends Component {
5   constructor() {
6     super();
7     this.state = {
8       name: "React"
9     };
10 }
11 handleChange(event) {
12   console.log(event.target.value);
13 }
14
15 render() {
16   return (
17     <div className={styles.simple_text_area}>
18       <input
19         className={styles.text_area}
20         type="textarea"
21         name="textValue"
22         onChange={this.handleChange}
23       />
24     </div>
25   );
26 }
27 }
28
29 export default SimpleTextArea;
30
31

```

Fig 7.3.5: A snapshot showing the SimpleTextArea.js file.

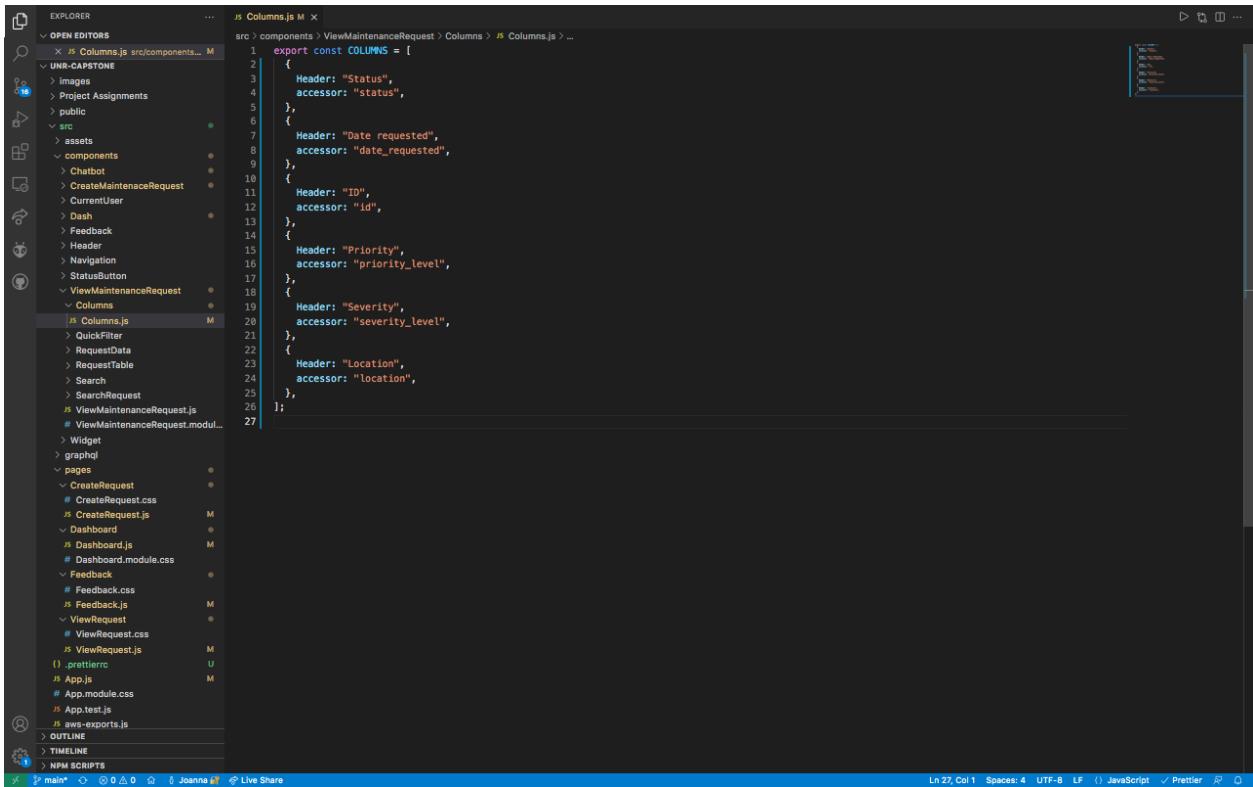
A screenshot of a code editor interface, likely VS Code, showing the file `SimpleTextArea.module.css`. The file contains CSS styles for a text area component. The code is as follows:

```
# SimpleTextArea.module.css M
src > components > Feedback > # SimpleTextArea.module.css > .text_area
1 .text_area {
2   appearance: none;
3   border: 2px solid #bfbfbf;
4   resize: none;
5   width: 100%;
6   border-radius: 10px;
7   transition: 0.2s all ease-in-out;
8   height: 100%;
9   width: 100%;
10 }
11 .text_area:focus {
12   box-shadow: 0 2px 10px 0.2px #rgb(185, 185, 185);
13   border-color: black;
14   outline: none;
15 }
16 .text_area:hover {
17   box-shadow: 0 2px 10px 0.2px #rgb(185, 185, 185);
18 }
19 }
```

The left sidebar shows a project structure for "UNR-CAPSTONE" with various components like Chatbot, CreateMaintenanceRequest, CurrentUser, Dash, Feedback, and others. The bottom status bar indicates the file is at line 1, column 1, with 4 spaces, using UTF-8 encoding, LF line endings, and CSS syntax.

Fig. 7.3.6: A snapshot showing the SimpleTextArea.module.css file.

7.4. Nasrin Juana



A screenshot of a code editor interface, likely VS Code, showing the `Columns.js` file. The left sidebar shows a tree view of the project structure under the `OPEN EDITORS` tab, including files like `Components.js`, `Dashboard.js`, and `ViewMaintenanceRequest.js`. The right pane displays the `Columns.js` code:

```
export const COLUMNS = [
  {
    Header: "Status",
    accessor: "status",
  },
  {
    Header: "Date requested",
    accessor: "date_requested",
  },
  {
    Header: "ID",
    accessor: "id",
  },
  {
    Header: "Priority",
    accessor: "priority_level",
  },
  {
    Header: "Severity",
    accessor: "severity_level",
  },
  {
    Header: "Location",
    accessor: "location",
  },
];
```

Fig. 7.4.1: A snapshot showing the `Columns.js` file.

A screenshot of the Visual Studio Code interface. The left sidebar shows the file structure under 'OPEN EDITORS'. The main editor area displays the 'QuickFilter.js' file. The code defines a 'QuickFilter' component that uses a dropdown menu to select from three options: Priority, Severity, or Date. It includes imports for React, useState, and react-select, and uses a CSS module named 'QuickFilter.module.css'.

```
js QuickFilter.js M ...
src > components > ViewMaintenanceRequest > QuickFilter > QuickFilter.js > QuickFilter > data
1 import React, {useState} from "react";
2 import Select from "react-select";
3 import styles from "./QuickFilter.module.css";
4
5 function QuickFilter() {
6   const data = [
7     {
8       value: 1,
9       label: "Priority",
10      },
11      {
12        value: 2,
13        label: "Severity",
14      },
15      {
16        value: 3,
17        label: "Date",
18      },
19    ];
20
21   const [selectedOption, setSelectedOption] = useState(null);
22
23   // handle onChange event of the dropdown
24   const handleChange = (e) => {
25     setSelectedOption(e);
26   };
27
28
29   return (
30     <div className={styles.QuickFilter}>
31       <Select
32         placeholder="Quick Filter"
33         value={selectedOption} // set selected value
34         options={data} // set list of the data
35         onChange={handleChange} // assign onChange function
36       />
37
38       {selectedOption && (
39         <div style={{ marginTop: "0px", lineHeight: "25px" }}>
40           <div style={{ marginTop: "0px" }}> {selectedOption.label} </div>
41         </div>
42       );
43     }
44   );
45 }
46
47 export default QuickFilter;
```

Fig. 7.4.2: A snapshot showing the QuickFilter.js file.

A screenshot of the Visual Studio Code interface. The left sidebar shows the file structure under 'OPEN EDITORS'. The main editor area displays the 'QuickFilter.module.css' file, which contains a single CSS rule for the '.QuickFilter' class, setting its text-align to center, background-color to #ffd588, color to black, and float to right.

```
# QuickFilter.module.css M ...
src > components > ViewMaintenanceRequest > QuickFilter > QuickFilter.module.css > .QuickFilter
1 .QuickFilter {
2   text-align: center;
3   background-color: #ffd588;
4   color: black;
5   float: right;
6 }
```

Fig. 7.4.3: A snapshot showing the QuickFilter.module.css file.

```

    (1) RequestData.json M ×
    src > components > ViewMaintenanceRequest > RequestData > (1) RequestData.json > () 8
    [
      {
        "id": "201345",
        "status": "Completed",
        "date_requested": "26-Aug-2021",
        "priority_level": "High",
        "severity_level": "Medium",
        "location": "Kitchen"
      },
      {
        "id": "123456",
        "status": "Completed",
        "date_requested": "19-Jan-2021",
        "priority_level": "High",
        "severity_level": "High",
        "location": "Bathroom"
      },
      {
        "id": "546373",
        "status": "Completed",
        "date_requested": "31-Oct-2021",
        "priority_level": "Low",
        "severity_level": "Medium",
        "location": "Bedroom"
      },
      {
        "id": "845327",
        "status": "In Progress",
        "date_requested": "26-Nov-2021",
        "priority_level": "High",
        "severity_level": "Low",
        "location": "Kitchen"
      },
      {
        "id": "968534",
        "status": "Completed",
        "date_requested": "26-Oct-2020",
        "priority_level": "High",
        "severity_level": "Low",
        "location": "Living room"
      },
      {
        "id": "496548",
        "status": "Completed",
        "date_requested": "19-Nov-2021",
        "priority_level": "Low",
        "severity_level": "Medium",
        "location": "Living room"
      },
      {
        "id": "675849",
        ...
    ]
  
```

Fig. 7.4.4: A snapshot showing the RequestData.json file.

```

    JS RequestTable.js M ×
    src > components > ViewMaintenanceRequest > RequestTable > JS RequestTable.js > ...
    import React, { useMemo } from "react";
    import useTable, { useGlobalFilter } from "react-table";
    import RequestData from "../RequestData/RequestData.json";
    import { COLUMNS } from "./Columns/Columns";
    import styles from "./RequestTable.module.css";
    import { SearchRequest } from "./SearchRequest/SearchRequest";
    import StatusButton from "./StatusButton/StatusButton";
    import QuickFilter from "./QuickFilter/QuickFilter";
    //import Search from "./Search/Search";
    ...

    export const RequestTable = () => {
      const columns = useMemo(() => COLUMNS, []);
      const data = useMemo(() => RequestData, []);

      /*
       * creating an instance of a table
       * destructuring table element
       * functions and arrays from useTable hook from react table package given to enable easy table creation
       * use this with html (in return) to make the table
       */
      const {
        getTableProps,
        getTableBodyProps,
        headerGroups,
        footerGroups,
        prepareRow,
        state,
        setGlobalFilter,
      } = useTable(
        {
          columns,
          data,
        },
        useGlobalFilter
      );

      const { globalFilter } = state;

      return (
        <div className={styles.table}>
          <div className={styles.table_filters}>
            <SearchRequest filter={globalFilter} setFilter={setGlobalFilter} />
            <QuickFilter />
          </div>
          <table {...getTableProps()}>
            <thead>
              {headerGroups.map((headerGroup) => (
                <tr {...headerGroup.getHeaderGroupProps()}>
                  {headerGroup.headers.map((column) => (
                    <th {...column.getHeaderProps()}>{column.render("Header")}</th>
                  ))}
                </tr>
              ))}
            </thead>
            <tbody>
              {data.map((row) => (
                <tr {...row.getRowProps()}>
                  {row.cells.map((cell) => (
                    <td {...cell.getCellProps()}>{cell.render("Cell")}</td>
                  ))}
                </tr>
              ))}
            </tbody>
          </table>
        </div>
      );
    }
  
```

Fig. 7.4.5: A snapshot showing the RequestTable.js file.

The screenshot shows a code editor interface with the following details:

- EXPLORER** sidebar: Shows the project structure under "UNR-CAPSTONE". The "src" folder contains components like Chatbot, CreateMaintenanceRequest, CurrentUser, Dash, Feedback, Header, Navigation, StatusButton, ViewMaintenanceRequest, and RequestTable.
- OPEN EDITORS**: The "RequestTable.module.css" file is open in the editor. The code defines a table with specific styling rules for rows and columns.
- RIGHT SIDE**: A vertical panel displays the current file's content, showing the CSS code for the RequestTable module.
- STATUS BAR**: Shows "Ln 35, Col 1" and other file-related information.

```
# RequestTable.module.css 1, M ×  
src > components > ViewMaintenanceRequest > RequestTable > # RequestTable.module.css ...  
1 .table {  
2 }  
3 .table_filters {  
4 width: 80%;  
5 margin: 15px 0px 0px 60px;  
6 }  
7 table {  
8 border-collapse: collapse;  
9 width: 100%;  
10 font-family: "PT Sans", sans-serif;  
11 }  
12  
13 table td,  
14 table th {  
15 padding: 8px;  
16 text-align: center;  
17 margin: 5px 0;  
18 }  
19 table > tr {  
20 margin: 10px 0 !important;  
21 }  
22  
23 table tr:nth-child(even) {  
24 background-color: #f2f2f2;  
25 }  
26  
27 table th {  
28 padding-top: 12px;  
29 padding-bottom: 12px;  
30 text-align: center;  
31 background-color: #fce4d2;  
32 font-weight: 700;  
33 color: black;  
34 }  
35
```

Fig. 7.4.6: A snapshot showing the RequestTable.module.css file.

The screenshot shows a code editor interface with the following details:

- EXPLORER** sidebar: Shows the project structure under "UNR-CAPSTONE". The "src" folder contains components like Chatbot, CreateMaintenanceRequest, CurrentUser, Dash, Feedback, Header, Navigation, StatusButton, ViewMaintenanceRequest, and RequestTable.
- OPEN EDITORS**: The "SearchRequest.module.css" file is open in the editor. The code defines styles for input fields, specifically for the search request component.
- RIGHT SIDE**: A vertical panel displays the current file's content, showing the CSS code for the SearchRequest module.
- STATUS BAR**: Shows "Ln 35, Col 20" and other file-related information.

```
# SearchRequest.module.css M ×  
src > components > ViewMaintenanceRequest > SearchRequest > # SearchRequest.module.css > SearchRequest.module.css ...  
1 .SearchRequest {  
2 color: black;  
3 float: left;  
4 width: 15%;  
5 padding: 8px;  
6 }  
7  
8 .inp {  
9 text-align: right;  
10 appearance: none;  
11 border: 2px solid #bfbfbf;  
12 resize: none;  
13 width: 100%;  
14 height: 30px;  
15 font-family: "PT Sans", sans-serif;  
16 font-weight: 700;  
17 font-size: 15px;  
18 padding-right: 10px;  
19 border-radius: 10px;  
20 transition: 0.2s all ease-in-out;  
21 }  
22  
23 .inp:focus {  
24 box-shadow: 0 2px 10px 0.2px #rgb(185, 185, 185);  
25 border-color: black;  
26 outline: none;  
27 }  
28 .inp:hover {  
29 box-shadow: 0 2px 10px 0.2px #rgb(185, 185, 185);  
30 }
```

Fig. 7.4.7: A snapshot showing the SearchRequest.module.css file.

A screenshot of the Visual Studio Code interface. The left sidebar shows the project structure under 'OPEN EDITORS'. The main editor area displays the code for 'Search.js'. The code defines a class 'Search' that extends 'Component'. It includes a constructor, state management for 'request', and a method 'handleRequestChange' for handling input changes. The render method returns a form containing a single input field with type 'text' and an 'onChange' event handler. The code concludes with an 'export default Search;' statement. The status bar at the bottom indicates 'Ln 34, Col 1'.

```
js Search.js M ...
src > components > ViewMaintenanceRequest > Search > Search.js > ...
1 import React from "react";
2 import { Component } from "react";
3 import styles from "./Search.module.css";
4
5 class Search extends Component {
6   constructor(props) {
7     super(props);
8
9     this.state = {
10       request: ""
11     };
12   }
13
14   handleRequestChange = (event) => {
15     this.setState({ request: event.target.value });
16   };
17
18   render() {
19     return (
20       <form>
21         <div>
22           <input
23             type="text"
24             value={this.state.request}
25             onChange={this.handleRequestChange}
26           />
27         </div>
28       </form>
29     );
30   }
31 }
32
33 export default Search;
```

Fig. 7.4.8: A snapshot showing the Search.js file.

A screenshot of the Visual Studio Code interface. The left sidebar shows the project structure under 'OPEN EDITORS'. The main editor area displays the code for 'Search.module.css'. It contains a single CSS rule for a 'form' element, setting padding, margin, and background-color to zero and beige respectively. The status bar at the bottom indicates 'Ln 1, Col 1'.

```
# Search.module.css M ...
src > components > ViewMaintenanceRequest > Search > Search.module.css > ...
1 form {
2   padding: 0;
3   margin: 0;
4   background-color: beige;
5 }
```

Fig. 7.4.9: A snapshot showing the Search.module.css file.

A screenshot of the Visual Studio Code interface. The left sidebar shows the project structure under 'OPEN EDITORS'. The main editor area displays the code for 'ViewMaintenanceRequest.js'. The code defines a functional component 'ViewMaintenanceRequest' that returns a JSX structure with a title and a 'RequestTable' component. The file is saved with a green checkmark.

```
js ViewMaintenanceRequest.js M ...
src > components > ViewMaintenanceRequest > ViewMaintenanceRequest.js > ...
1 | import React from 'react';
2 | //Import Search from './Search/Search';
3 | import { RequestTable } from './RequestTable/RequestTable';
4 | import QuickFilter from './QuickFilter/QuickFilter';
5 | import styles from './ViewMaintenanceRequest.module.css';
6 |
7 | function ViewMaintenanceRequest() {
8 |   return (
9 |     <div className={styles.ViewMaintenanceRequestSection}>
10 |       <div className={styles.page_title}>
11 |         <p>View Maintenance Requests</p>
12 |       </div>
13 |       <RequestTable />
14 |     </div>
15 |   );
16 |
17 | }
18 | export default ViewMaintenanceRequest;
19 |
```

Fig. 7.4.10: A snapshot showing the ViewMaintenanceRequest.js file.

A screenshot of the Visual Studio Code interface. The left sidebar shows the project structure under 'OPEN EDITORS'. The main editor area displays the code for 'ViewMaintenanceRequest.module.module.css'. It contains CSS rules for the 'ViewMaintenanceRequestSection' and 'page_title' classes, including margin and font properties. The file is saved with a green checkmark.

```
js ViewMaintenanceRequest.js M # ViewMaintenanceRequest.module.css 1 ...
src > components > ViewMaintenanceRequest > # ViewMaintenanceRequest.module.css > ...
1 | .ViewMaintenanceRequestSection {
2 | }
3 |
4 | .page_title {
5 |   margin: 15px auto 0 60px;
6 | }
7 | .page_title p {
8 |   font-weight: 700;
9 |   font-size: 25px;
10|   margin-top: 10px;
11|
12| }
```

Fig. 7.4.11: A snapshot showing the ViewMaintenanceRequest.module.module.css file.

7.5. Aisha Co

Fig. 7.5.1: A snapshot showing the Chatbot.js file which includes everything regarding the chatbot component.

7.6. Github Repository

The Github repository containing Team 05's implemented code is included here in this link.

<https://github.com/joannalopez223/UNR-Capstone>