University of Nevada, Reno

Department of Computer Science and Engineering

CS 425: Software Engineering

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

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.

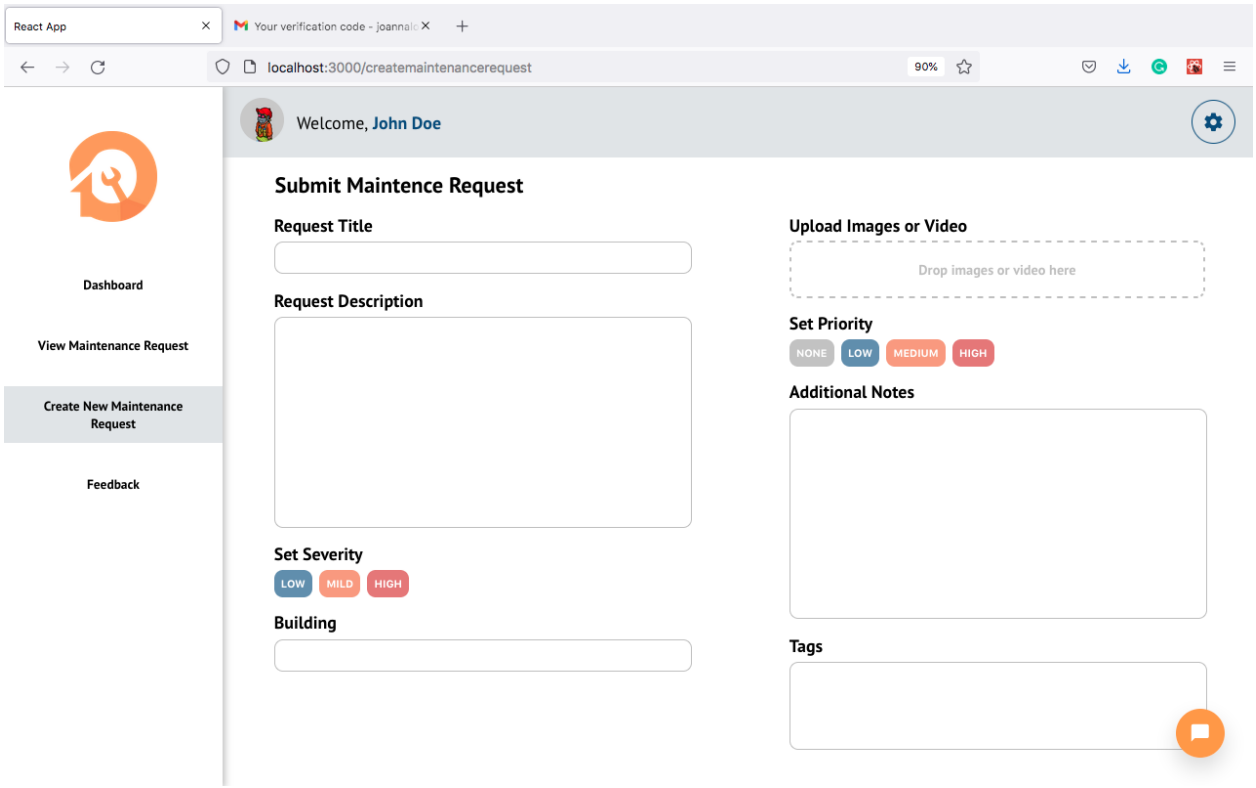


Fig. 4.1.1: The figure outlines the 'Create New Maintenance Request' page. The page has multiple text boxes for the user to input a request title, description, the building name, upload an image and/or video, and any additional notes and tags that the user would like to add to their respective request.

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.

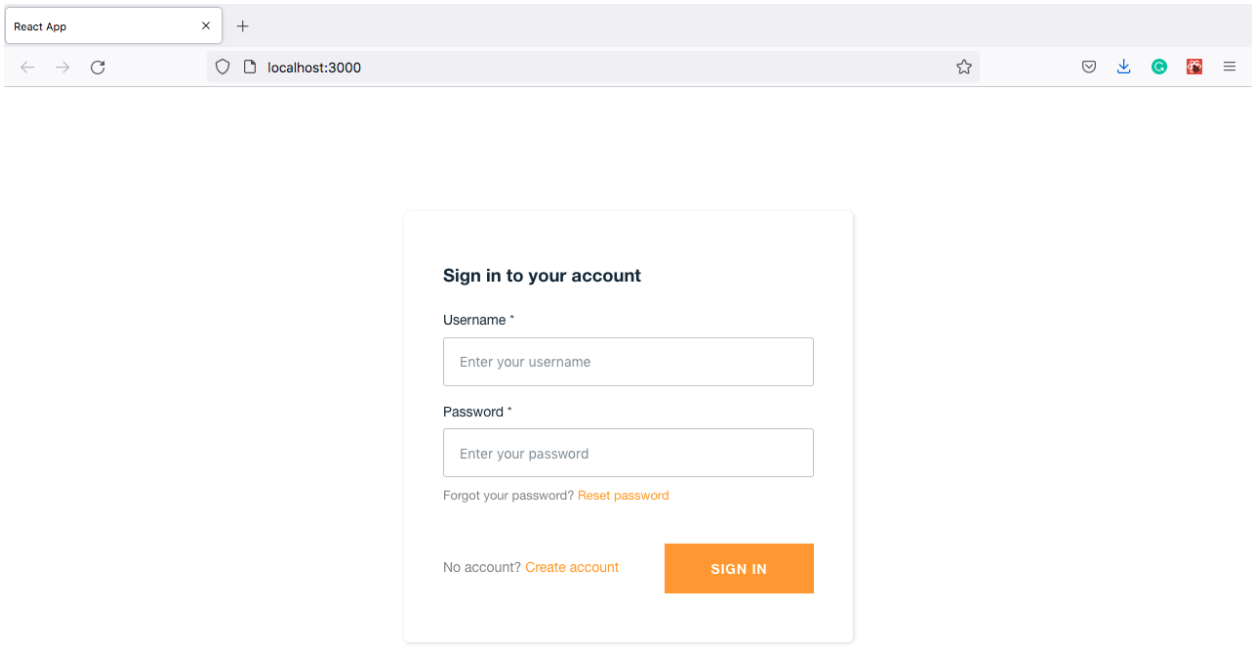


Fig. 4.2.1: The login page for users of the OPF website includes a text box for the ‘username’ and ‘password’. The user will then be able to sign in using the ‘sign in’ button to be taken to their respective account.

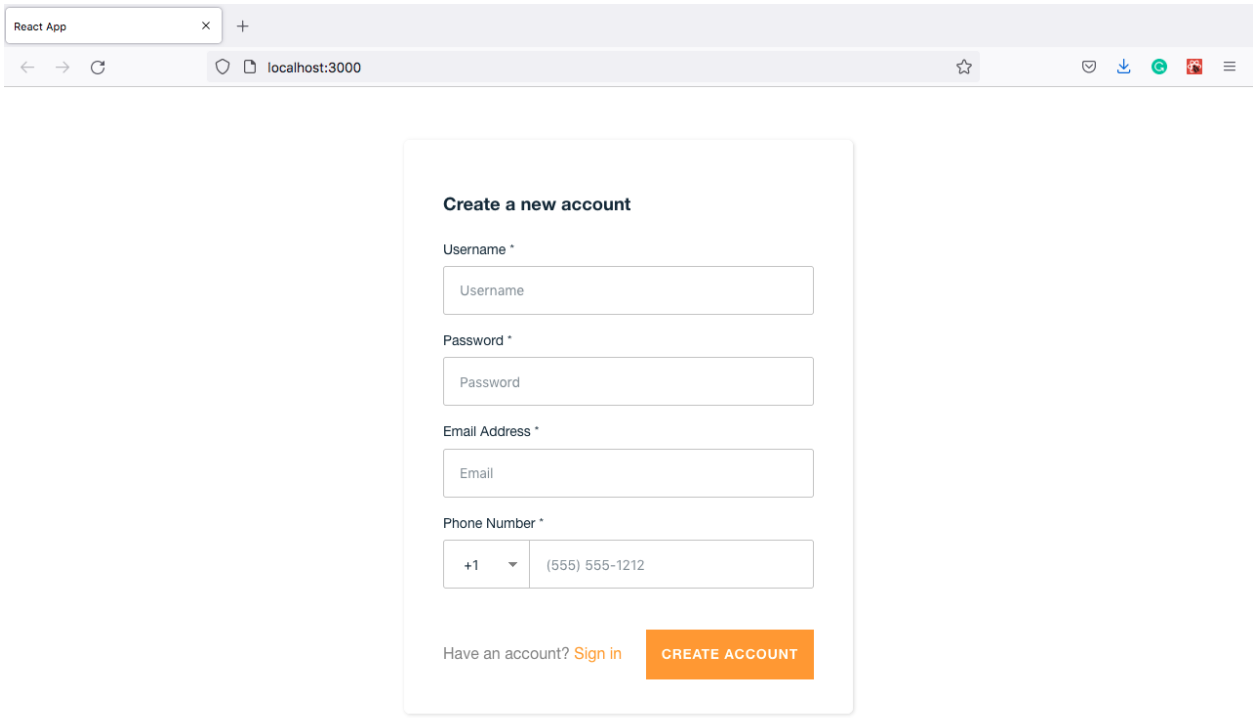


Fig. 4.2.2: A new user to the OPF website will be able to create their credentials such as their preferred ‘username’ and ‘password’. The new user will then need to supply their ‘email address’ and their ‘phone number’ for verification purposes. Team 05, has been selected to verify account credentials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

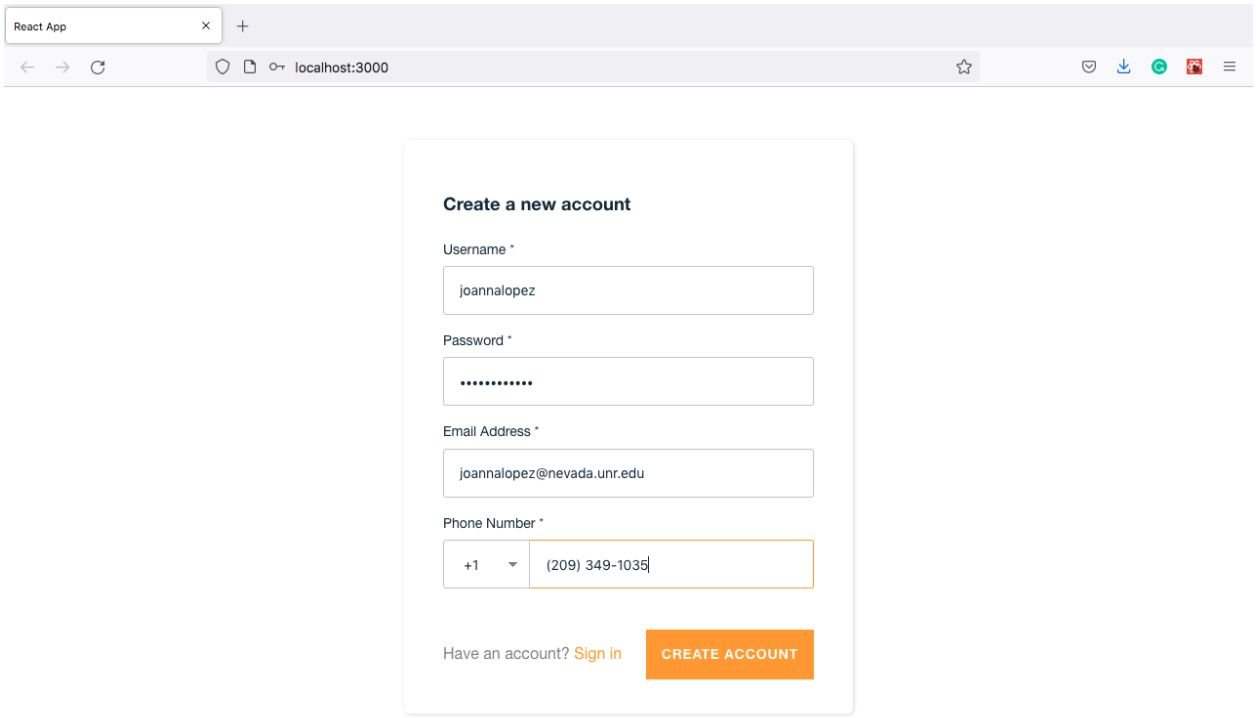


Fig. 4.2.3: The figure shows the filled-out text boxes with account information needed to access the OPF website. Once the information has been entered, the user will click on the 'create account' button to then take them to a confirmation page.

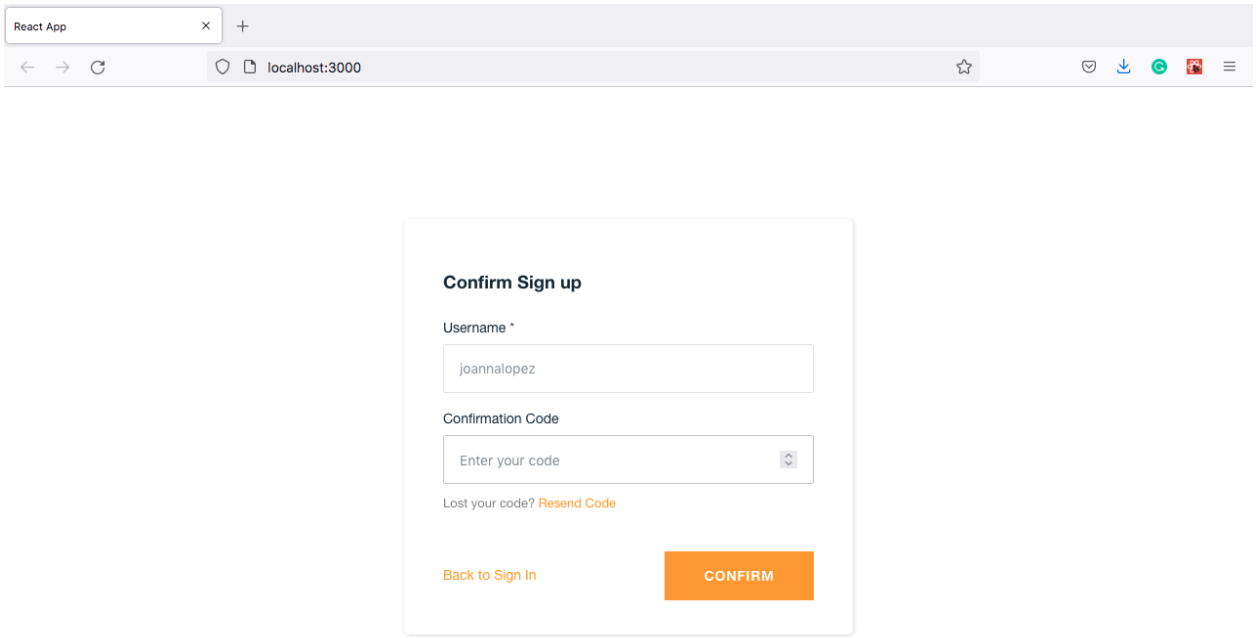


Fig. 4.2.4: The figure shows the ‘confirm sign up’ page with the user's preferred ‘username’ and a ‘confirmation code’ text box where the user can enter the code sent via their provided email.

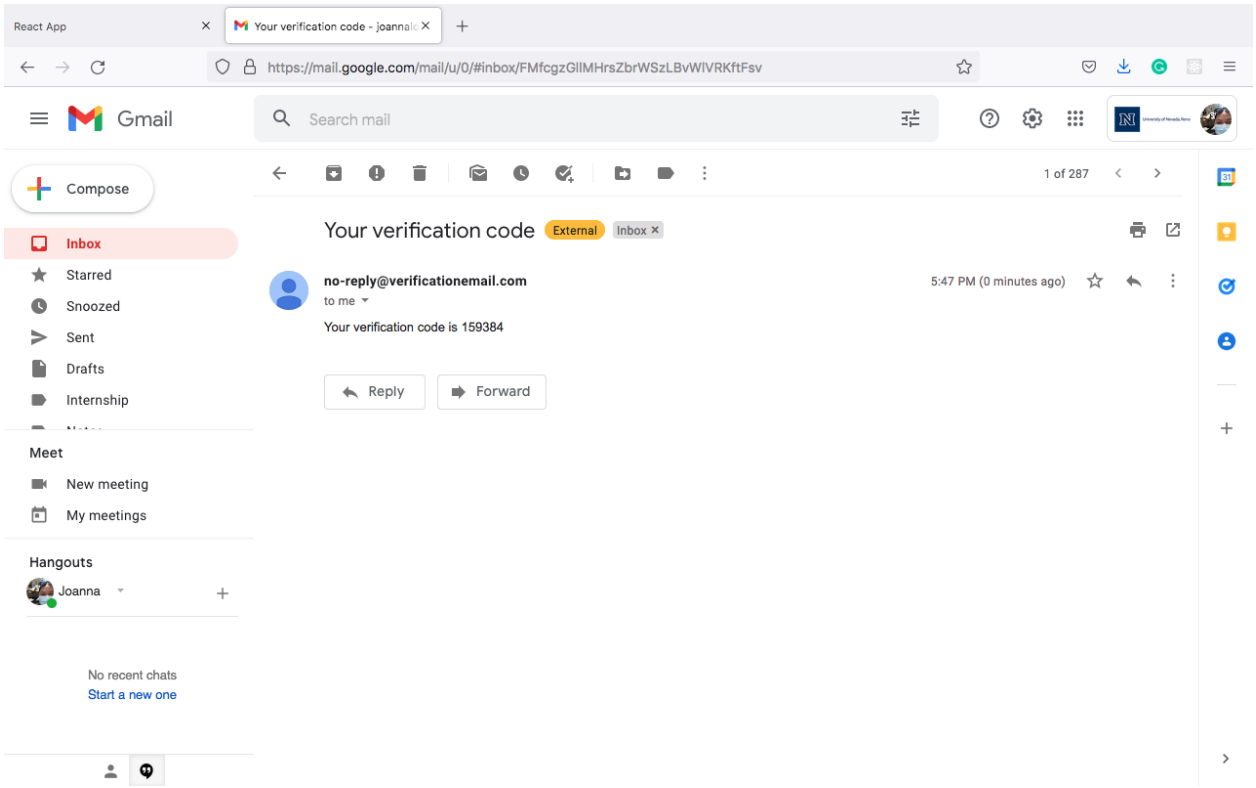


Fig. 4.2.5: The figure shows the email sent to the user and a six-digit verification code to authenticate their OPF account.

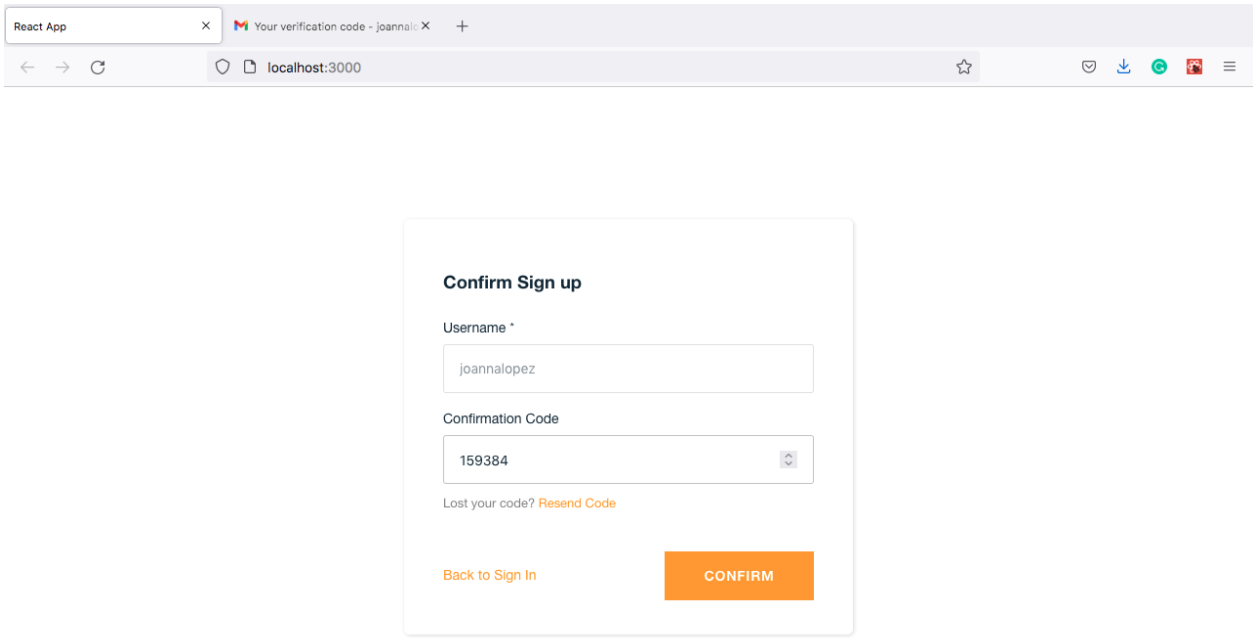


Fig. 4.2.6: The figure shows the inputting of the six-digit verification code to the 'Confirm sign up' page on the OPF account. The user will then be able to confirm their account by clicking the 'Confirm' button. The 'confirm sign up' page also has a 'resend code' in case the code was not received to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

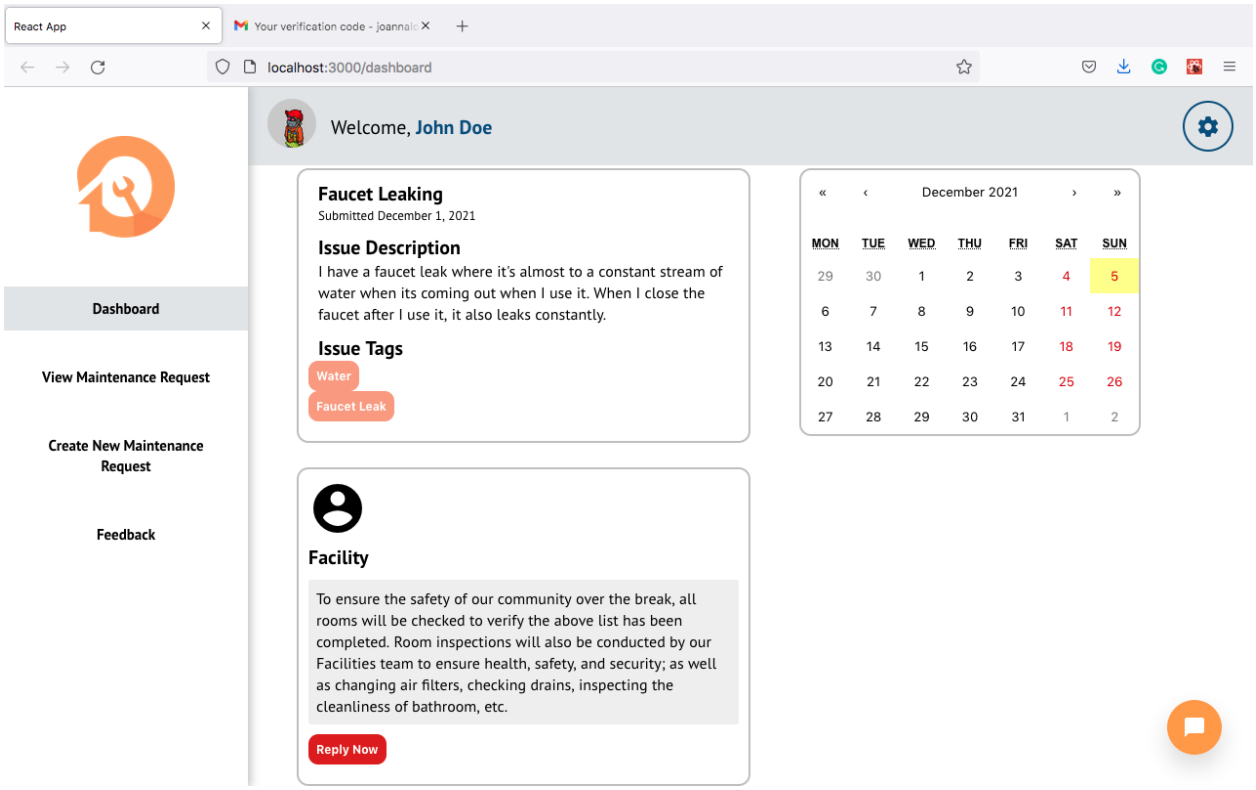


Fig. 4.3.1: The figure outlines the overview of the dashboard page including a faucet leaking ticket submitted by the user, a faculty message provided by an R.A at UNR, and a calendar that the user will be able to view any upcoming appointments.

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.

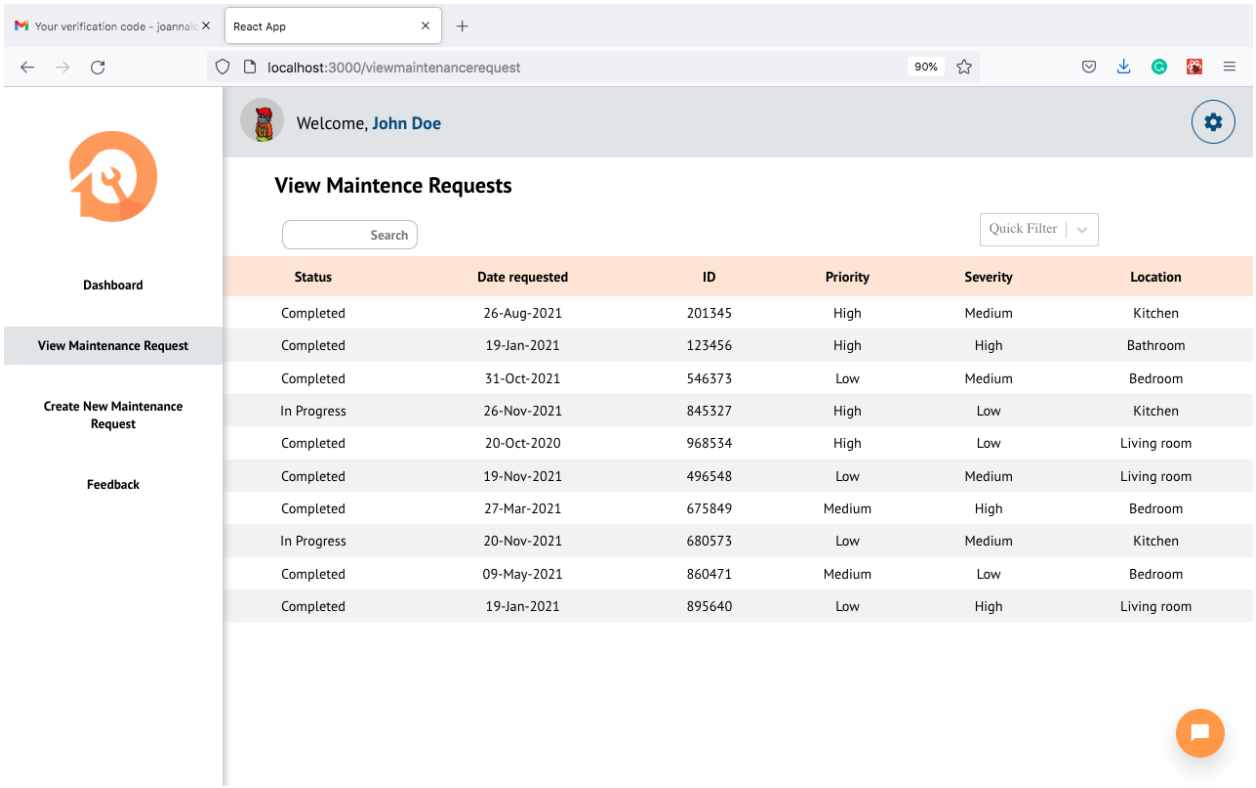


Fig 4.4.1: The 'View Maintenance Request' page shows all requests made by the user. The maintenance request shows the status of past tickets labeled as 'Status'. The headers include 'Date requested', 'ID' of the request, 'Priority' of the request, the 'Severity', and the 'Location'. The page also includes 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.

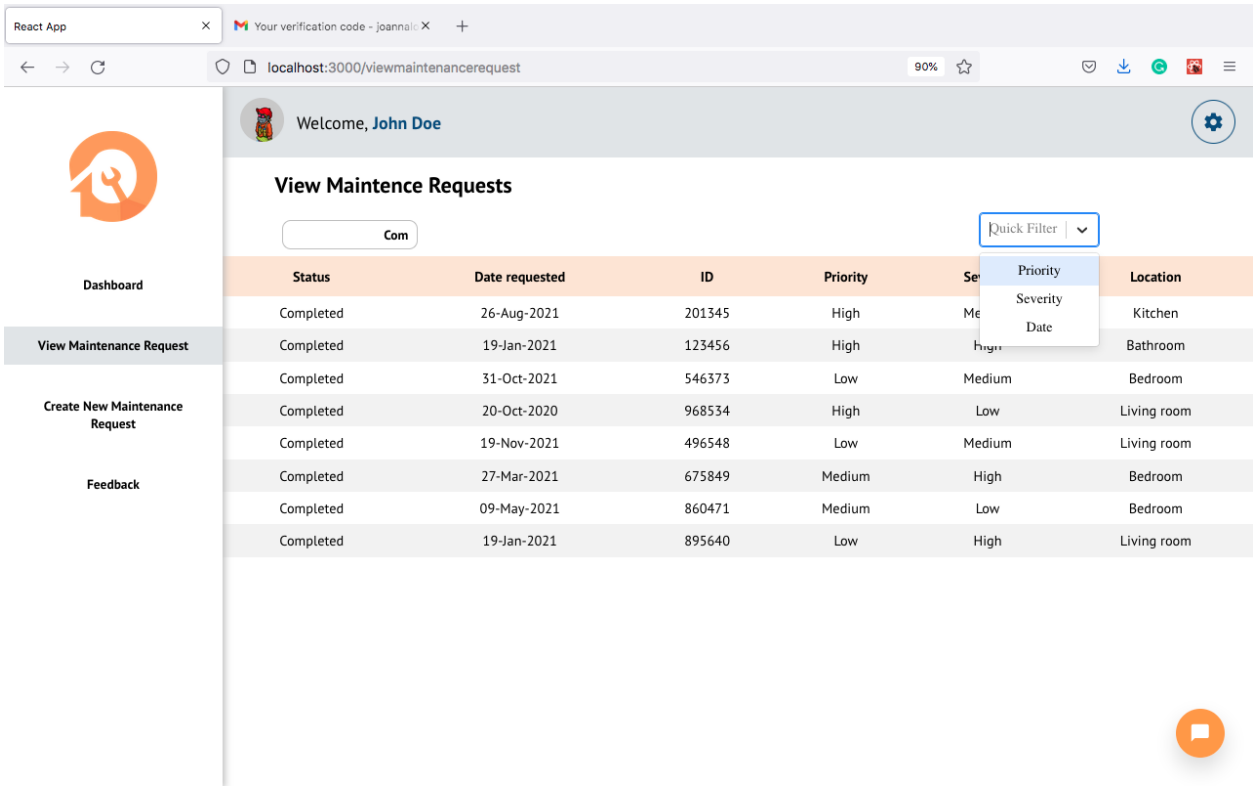


Fig. 4.4.2: The figure shows the requests made by filtering the tickets by a 'Completed' keyword in the search text box. The 'Quick Filter' option also shows other filters the user may use such as the request being 'Priority', 'Severity', and 'Date'.

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.

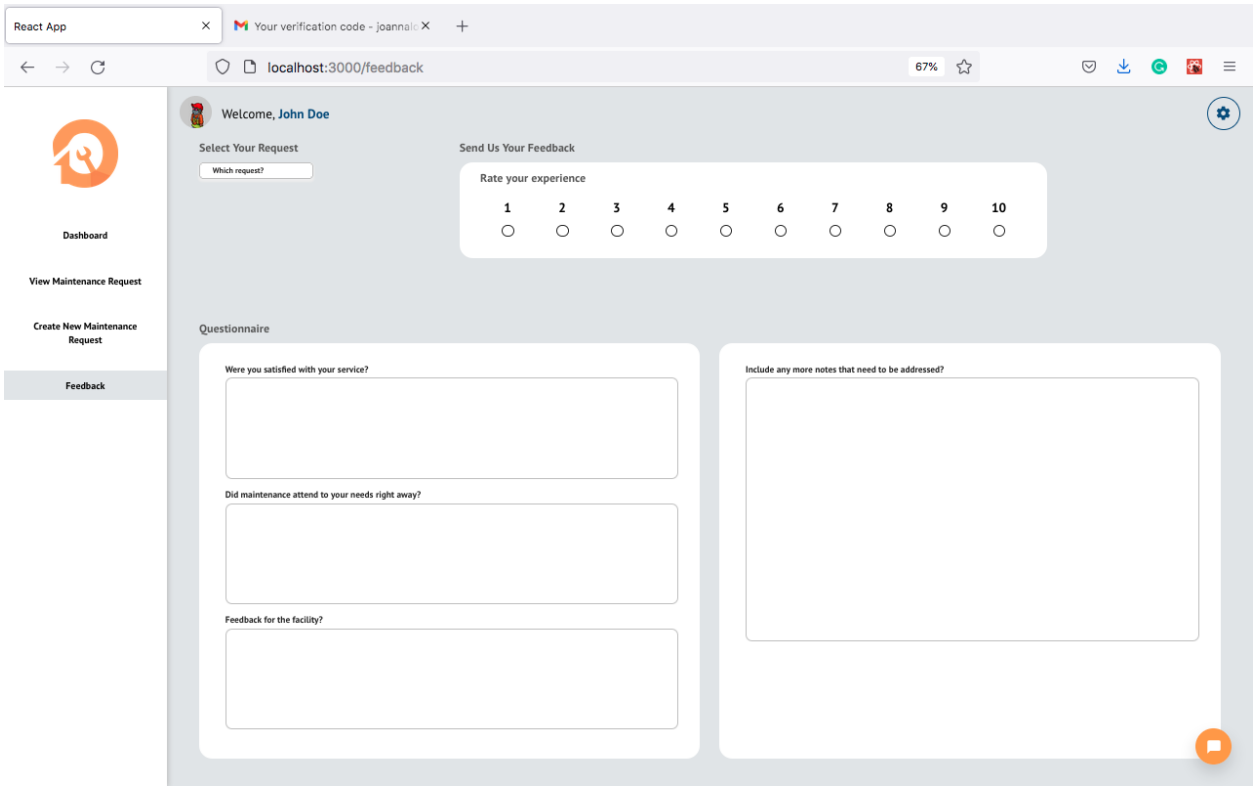


Fig. 4.5.1: Once the user has created a maintenance request and that request has been completed, they will have the ability to submit feedback for the request. The page presents the user with a set of questions that will help evaluate the service that was provided. The student may select which request 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.

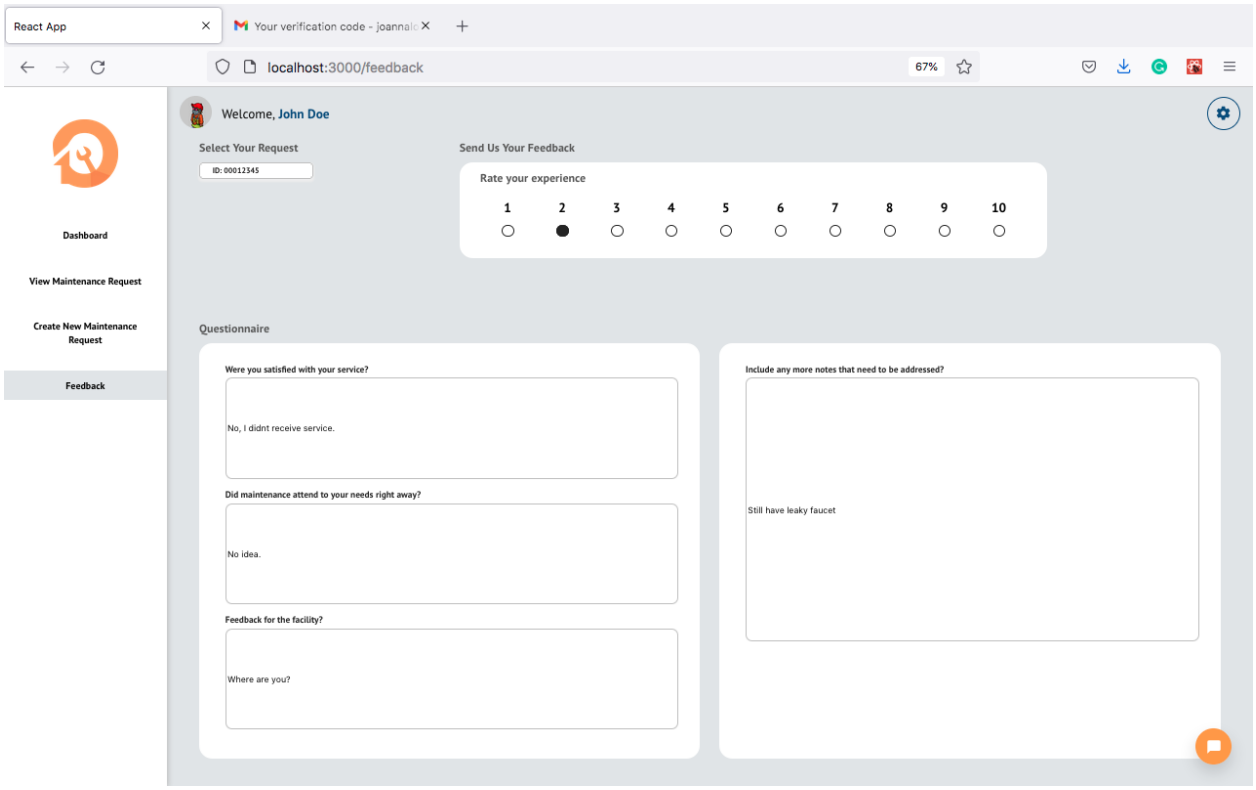


Fig. 4.5.2: This figure shows the functionality of the feedback page. The top left demonstrates that the user has selected the request they'd like to provide feedback for. The top right shows that the user selected a 2 out of 10 rating and the bottom half of the page shows the user answering the following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

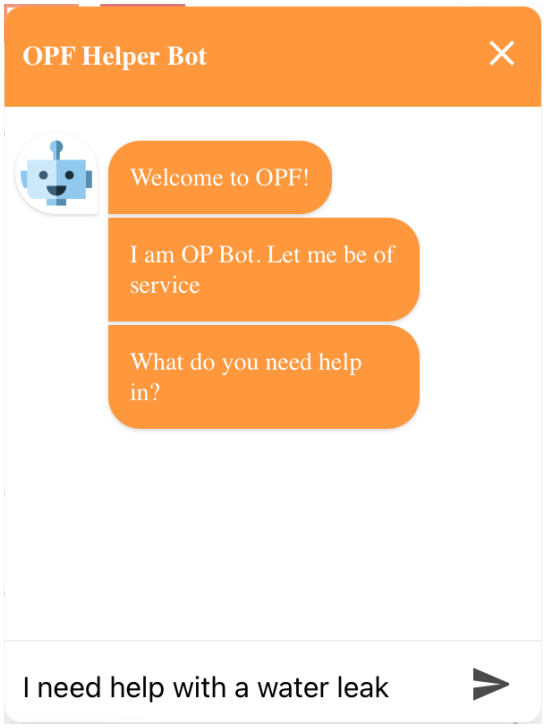


Fig. 4.6.1: This figure shows the first set of "steps" the chatbot would display to the user. The three messages are the default introduction messages the chatbot displays. In the input, the user can type input for the bot.

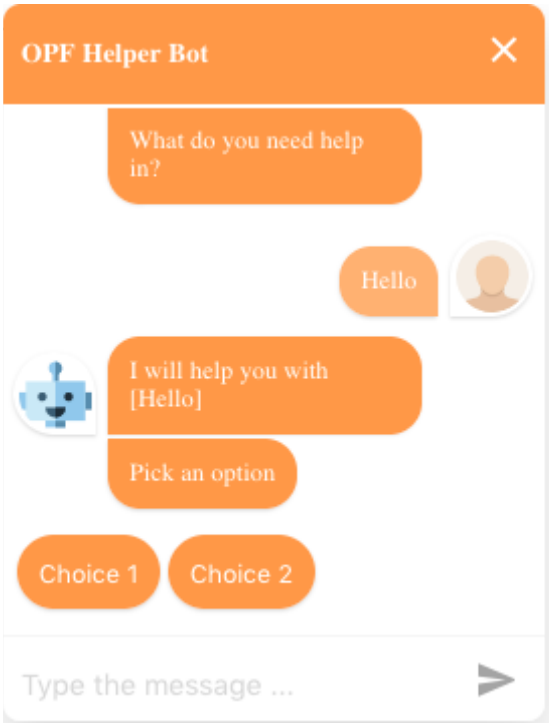


Fig. 4.6.2: This figure shows the message(s) the bot returns after the user types a message. The first message from the bot shows that the message was read from the bot. The second message leads to the bot displaying options for the user to pick.

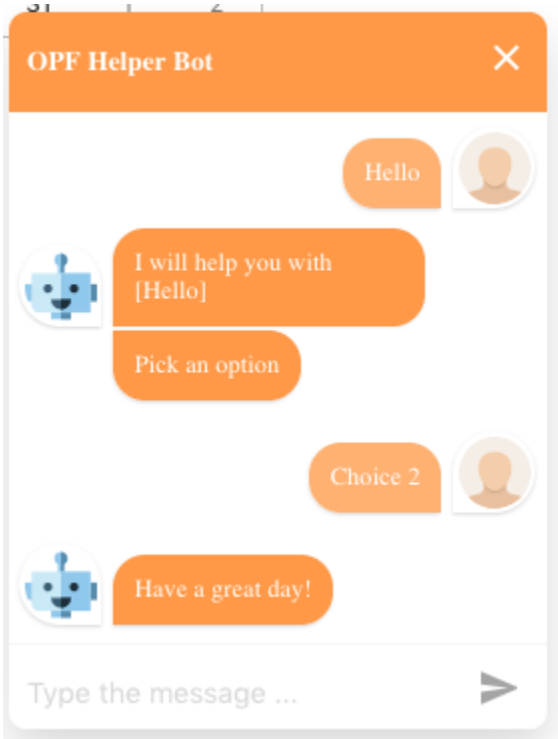


Fig. 4.6.3: This figure demonstrates after the user picks an option; the "Choice 2" option was chosen for this figure. The bot would create a message based on the option the user picks.

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

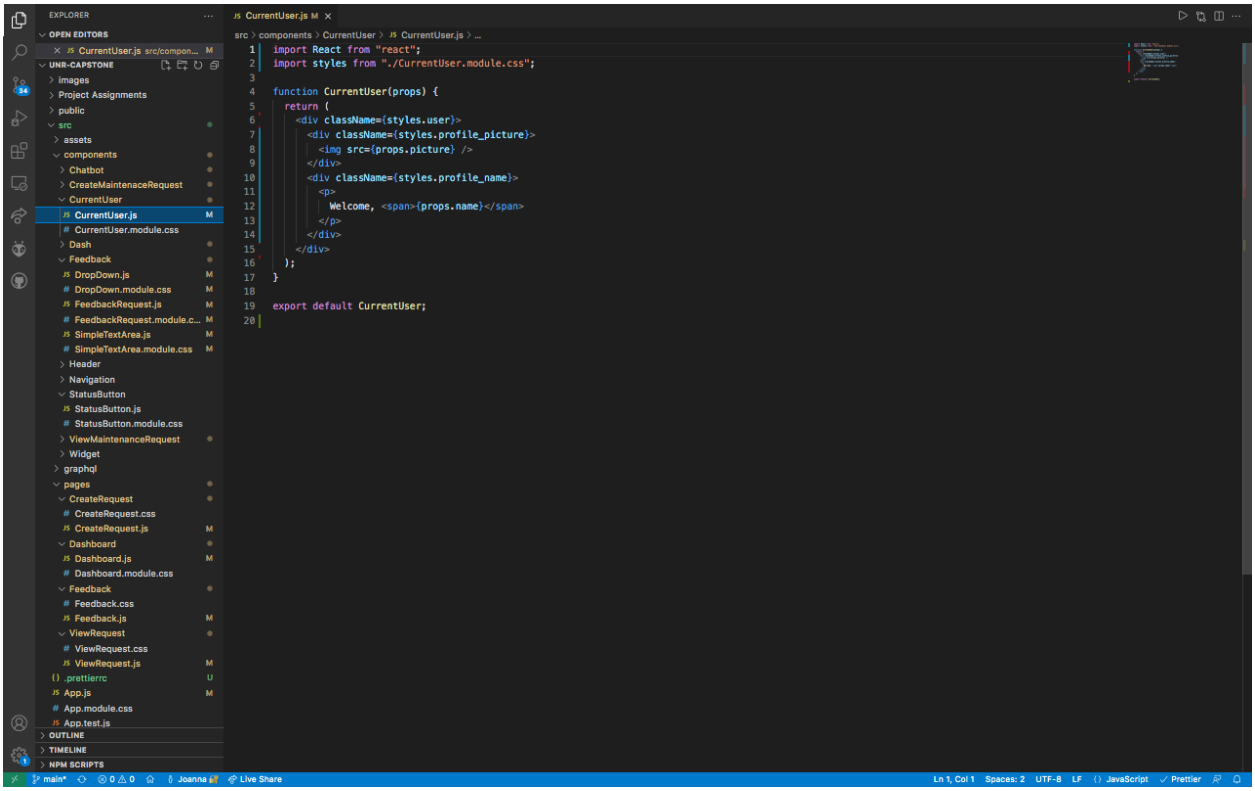


Fig. 7.1.1: A snapshot demonstrating the CurrentUser.js file.

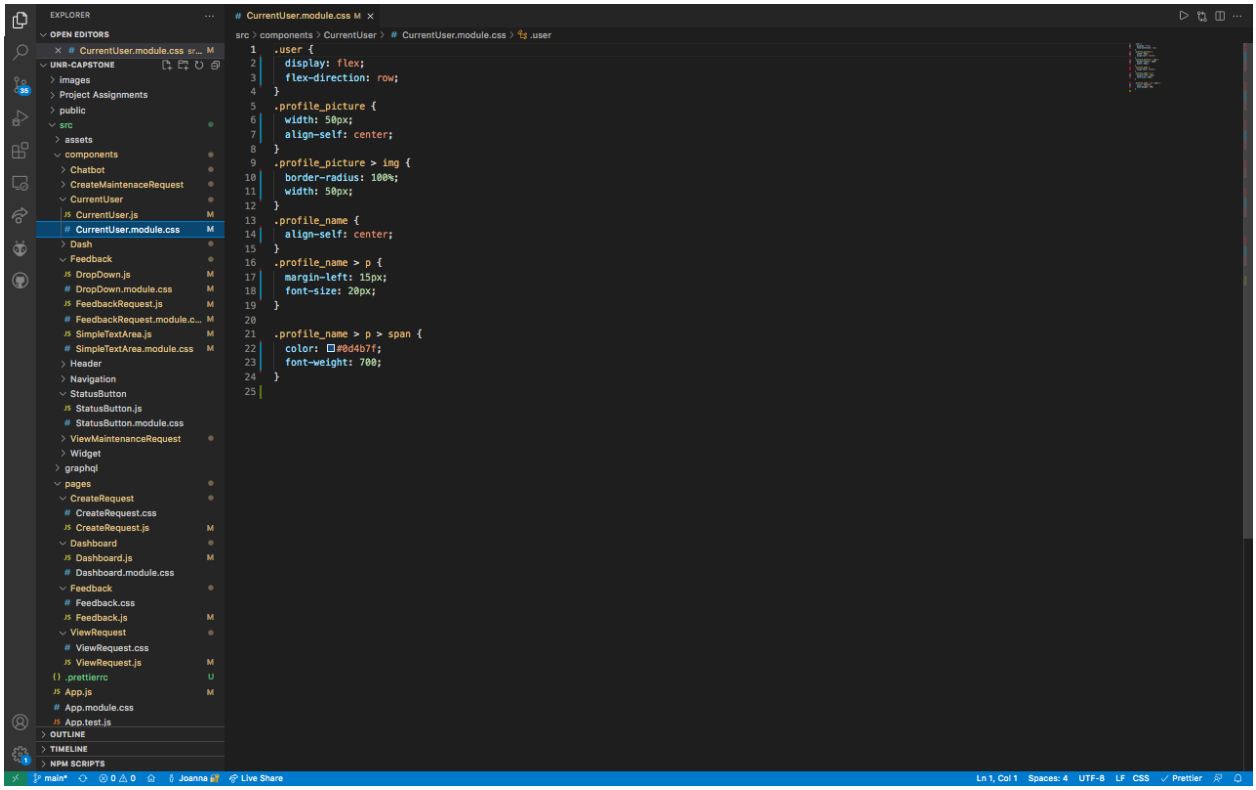


Fig. 7.1.2: A snapshot demonstrating the CurrentUser.module.css file.

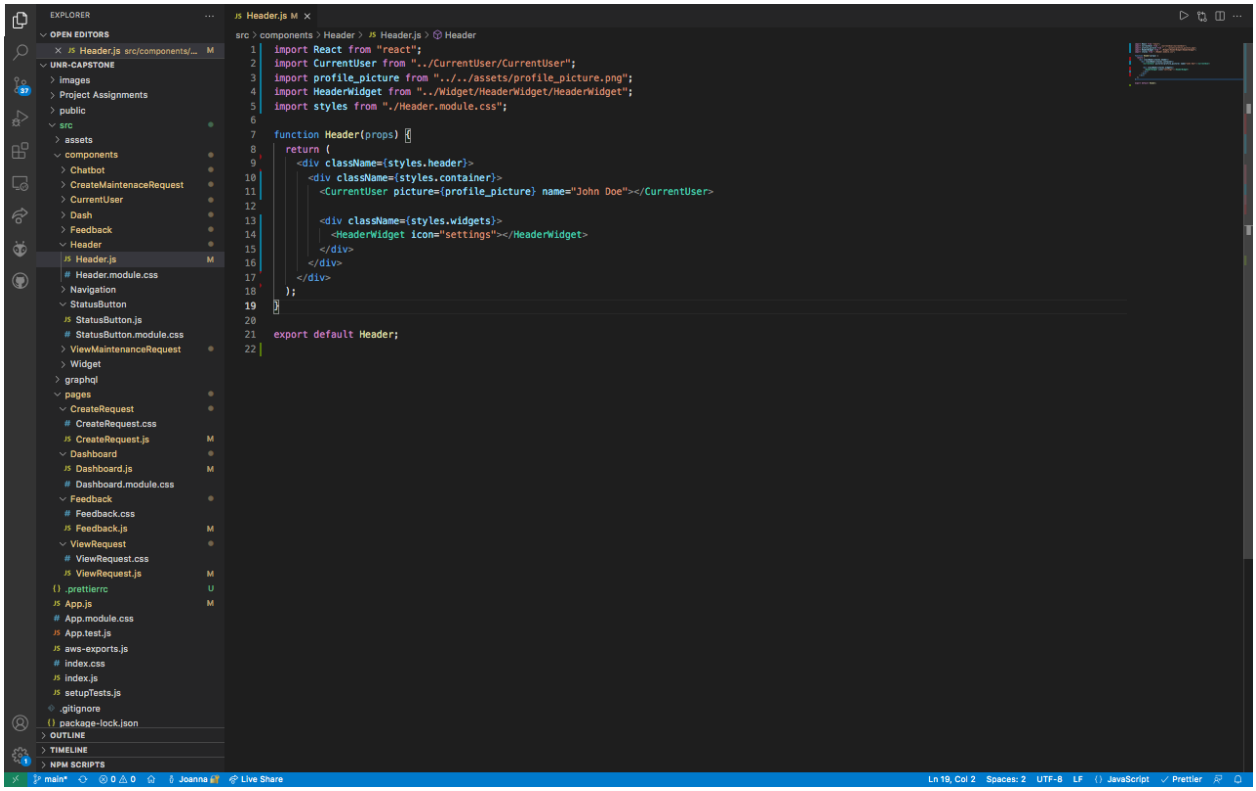


Fig. 7.1.3: A snapshot demonstrating the Header.js file.

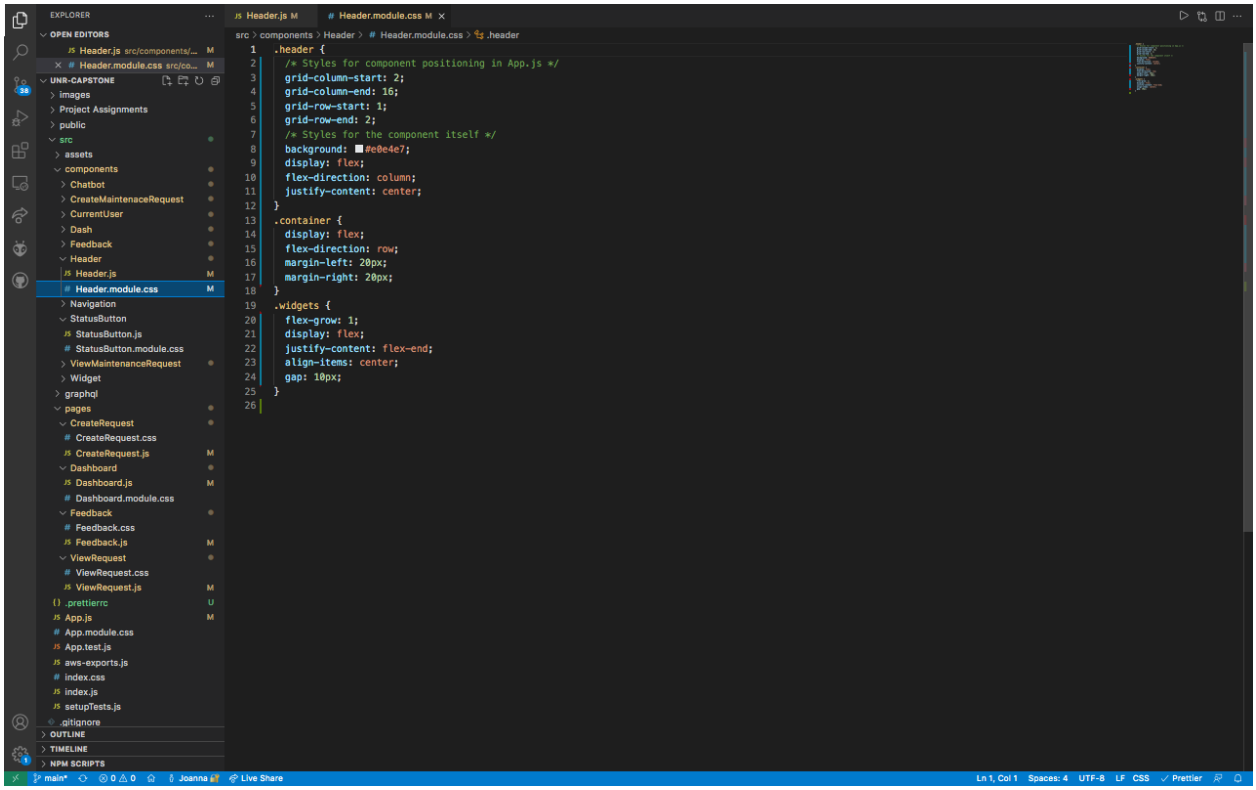


Fig. 7.1.4: A snapshot showing the Header.module.css file.

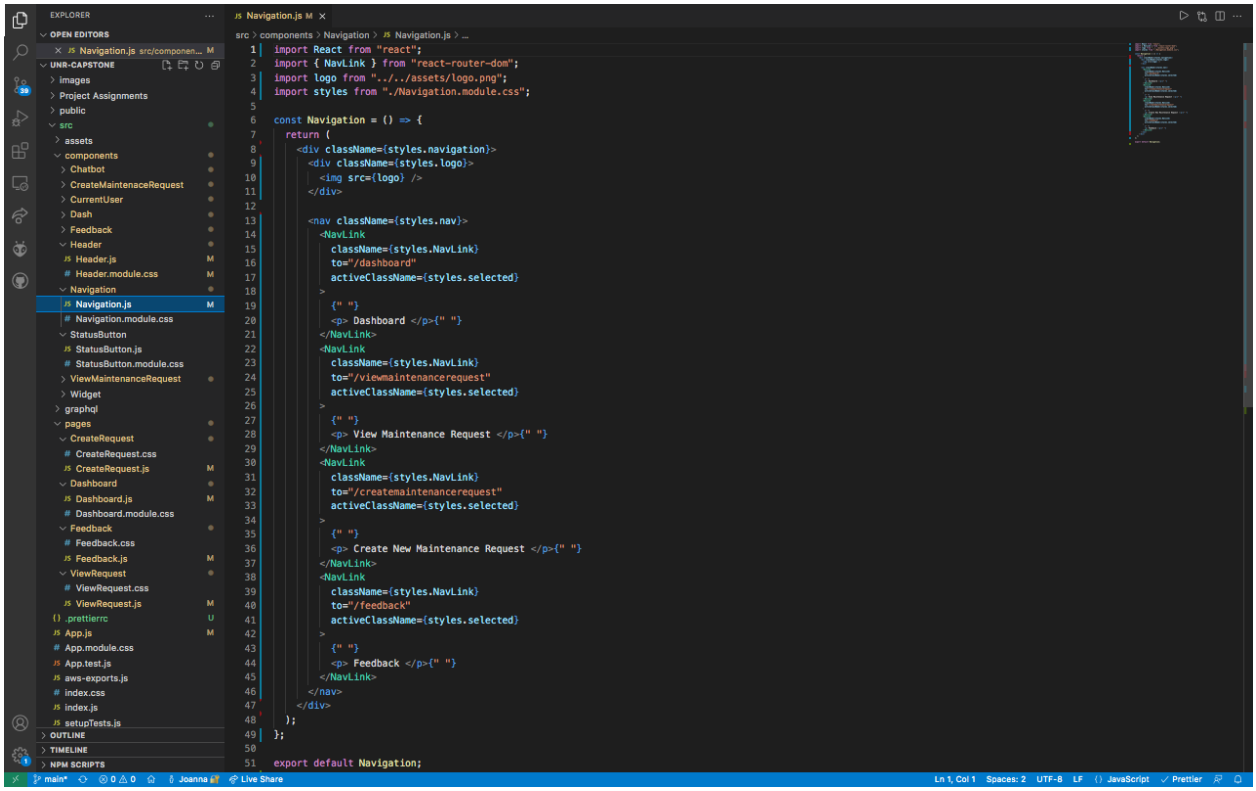


Fig. 7.1.5: A snapshot showing the Navigation.js file.

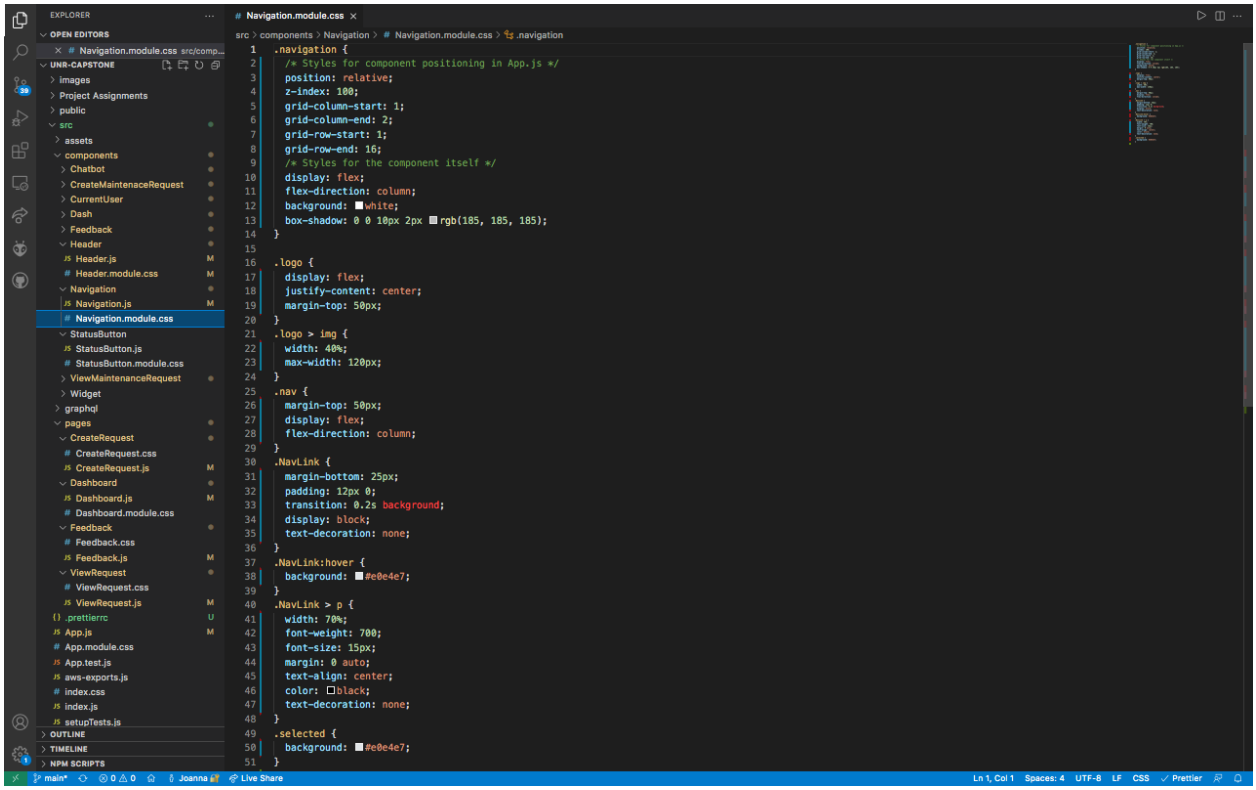


Fig. 7.1.6: A snapshot showing the Navigation.module.css file.

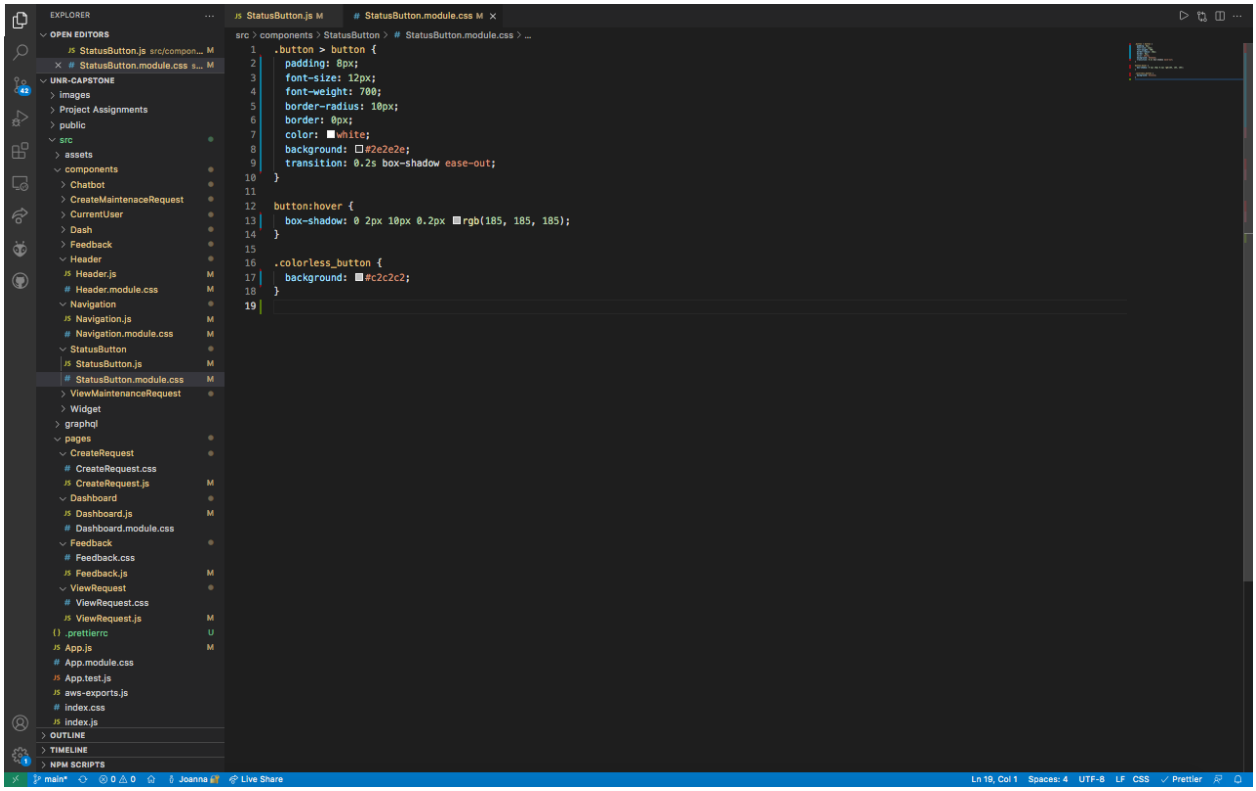


Fig 7.1.7: A snapshot showing the StatusButton.module.css file.

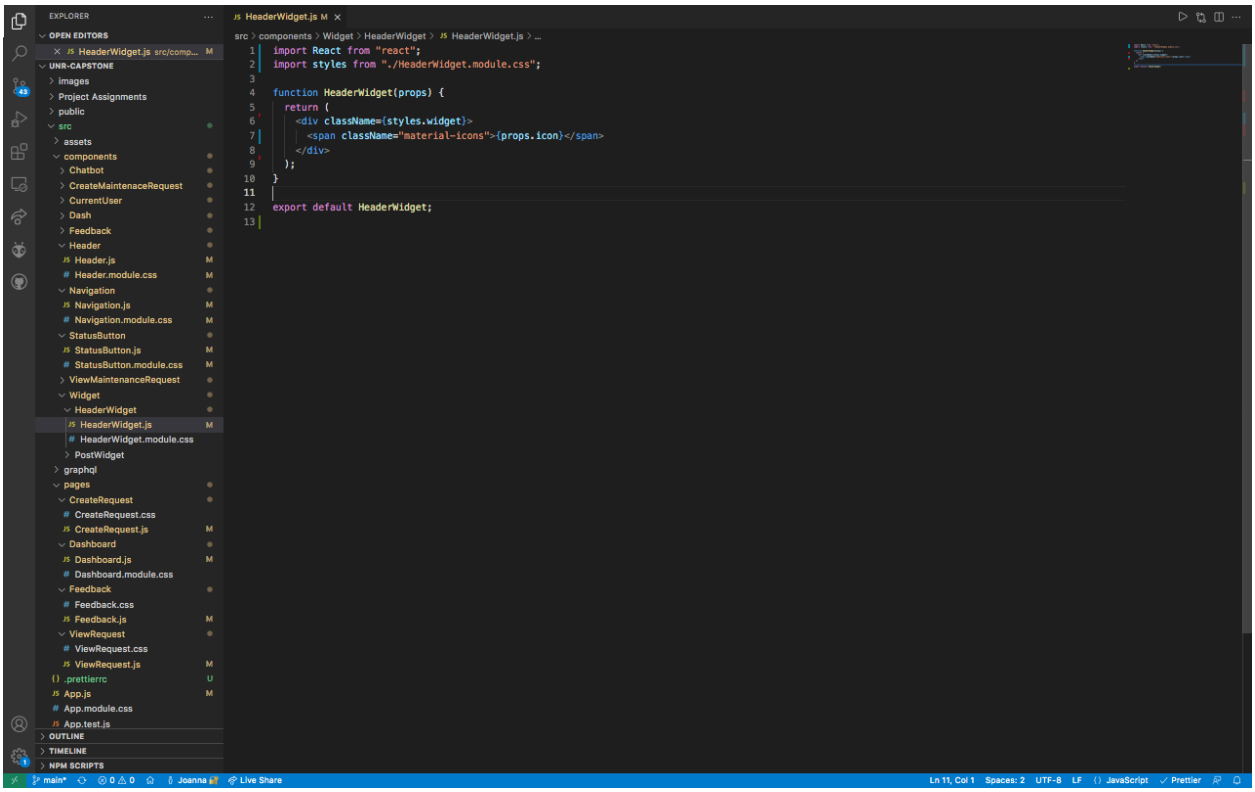


Fig. 7.1.8: A snapshot showing the HeaderWidget.js file.

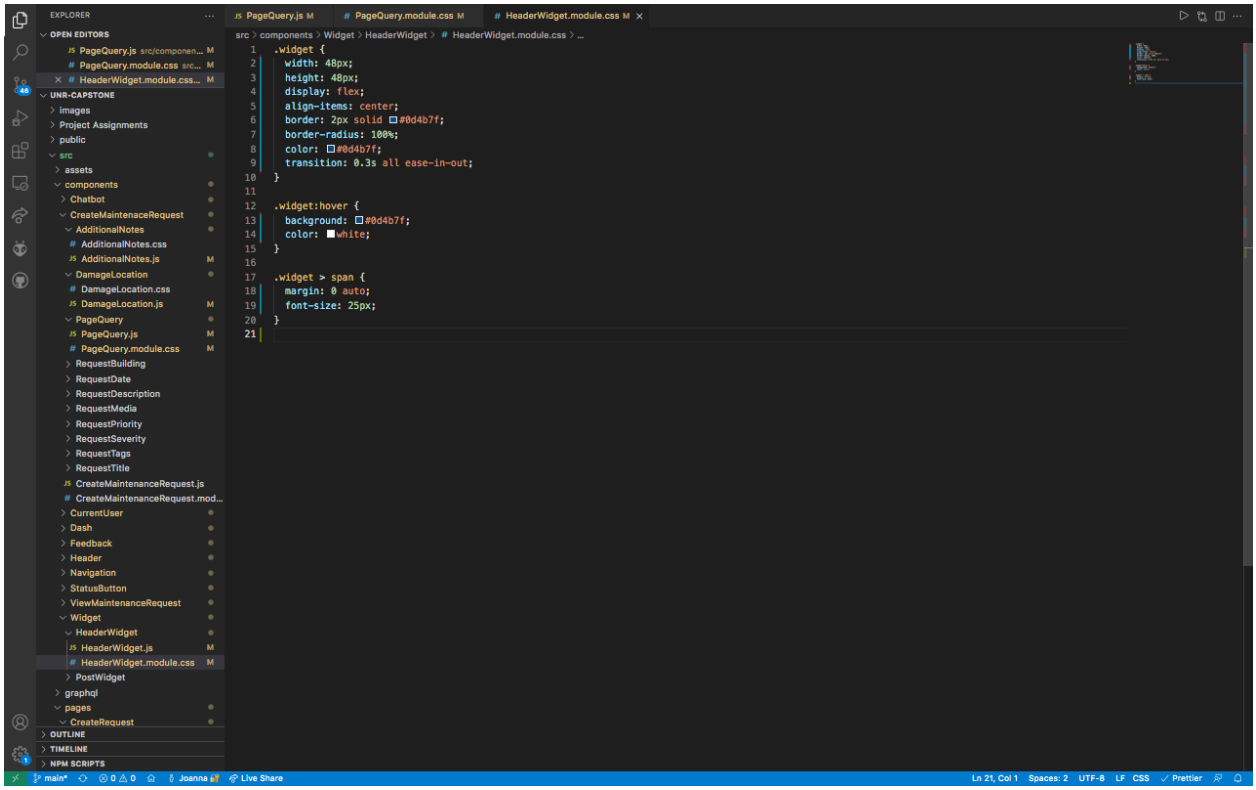


Fig. 7.1.9: A snapshot showing the Header.Widget.module.css file.

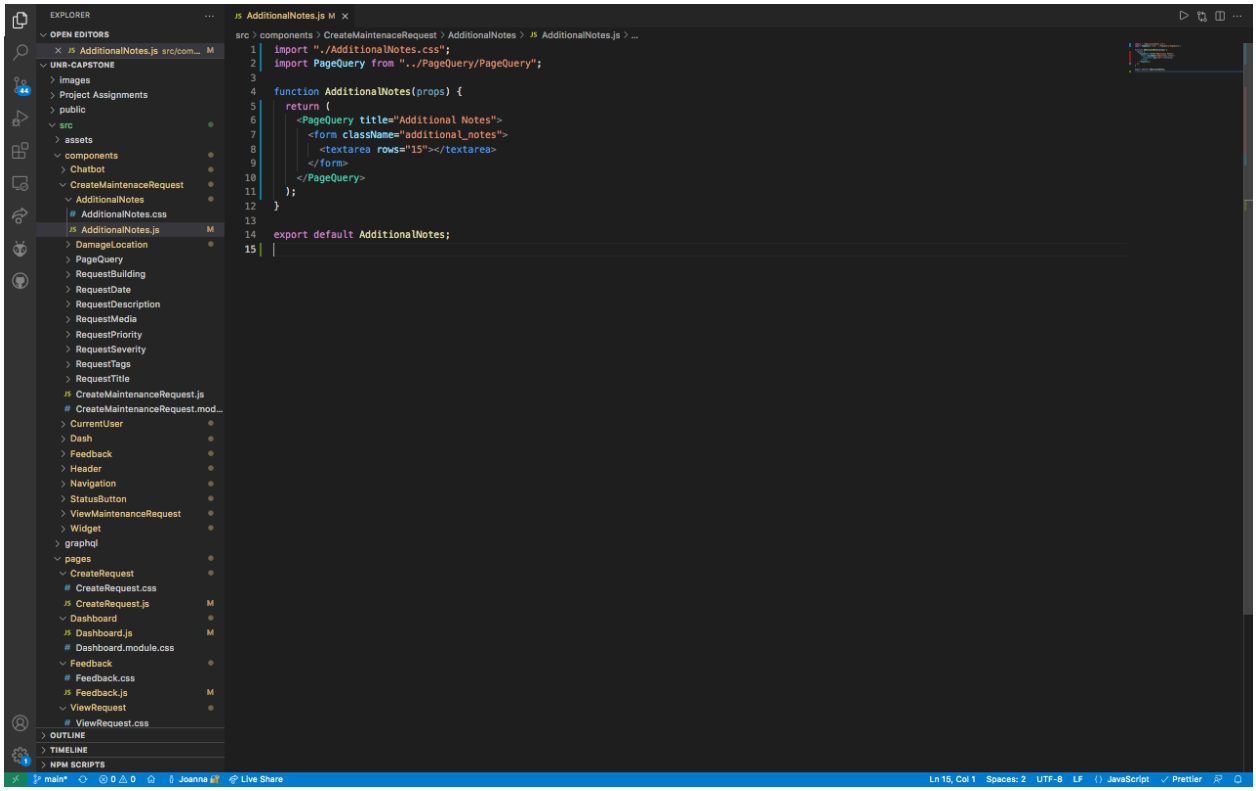


Fig. 7.1.10: A snapshot showing the AdditionalNotes.js file.

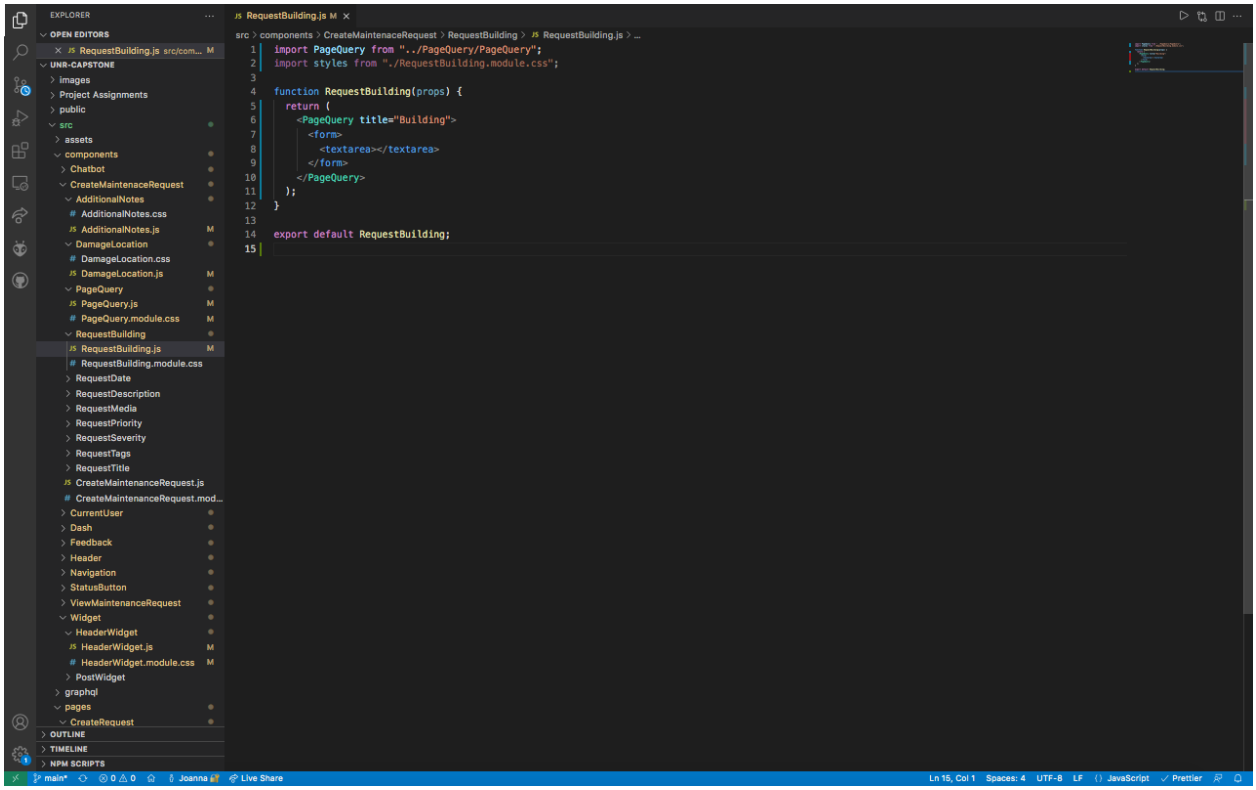


Fig. 7.1.11: A snapshot showing the RequestBuilding.js file.

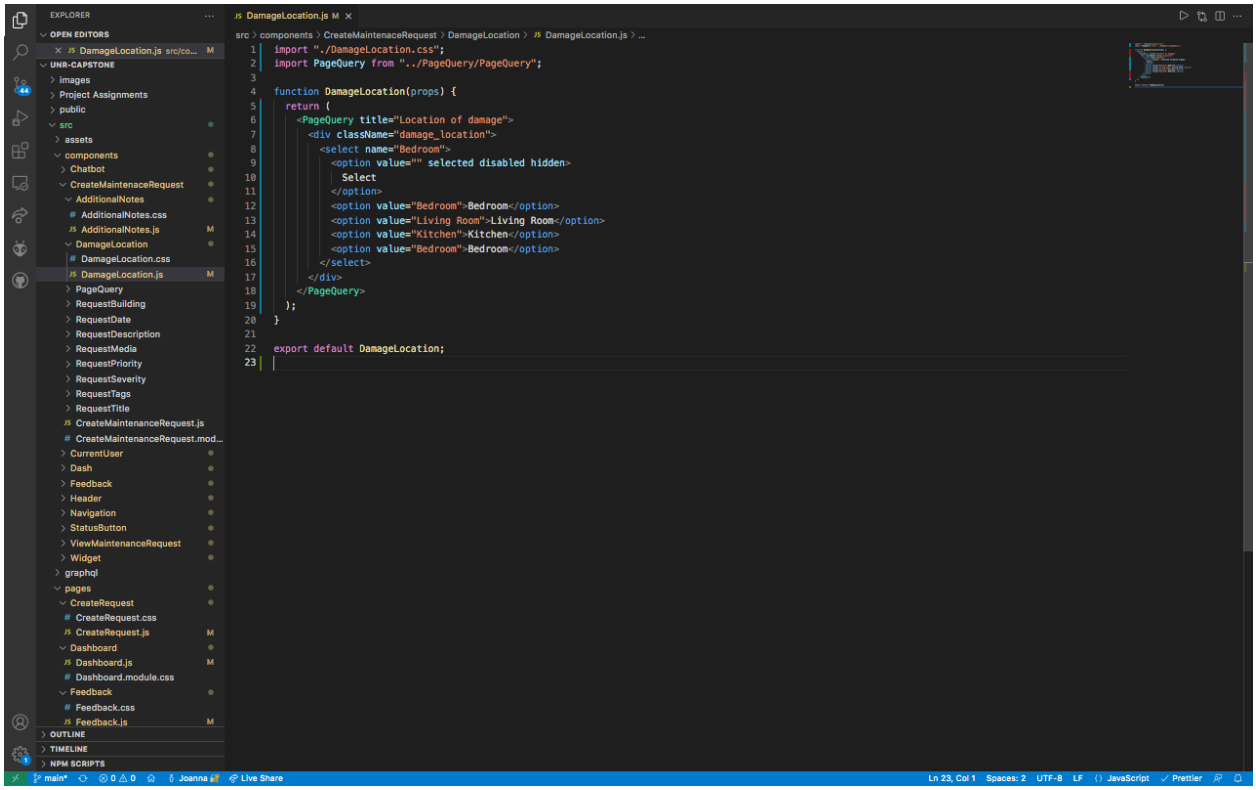


Fig 7.1.12: A snapshot showing the DamageLocation.js file.

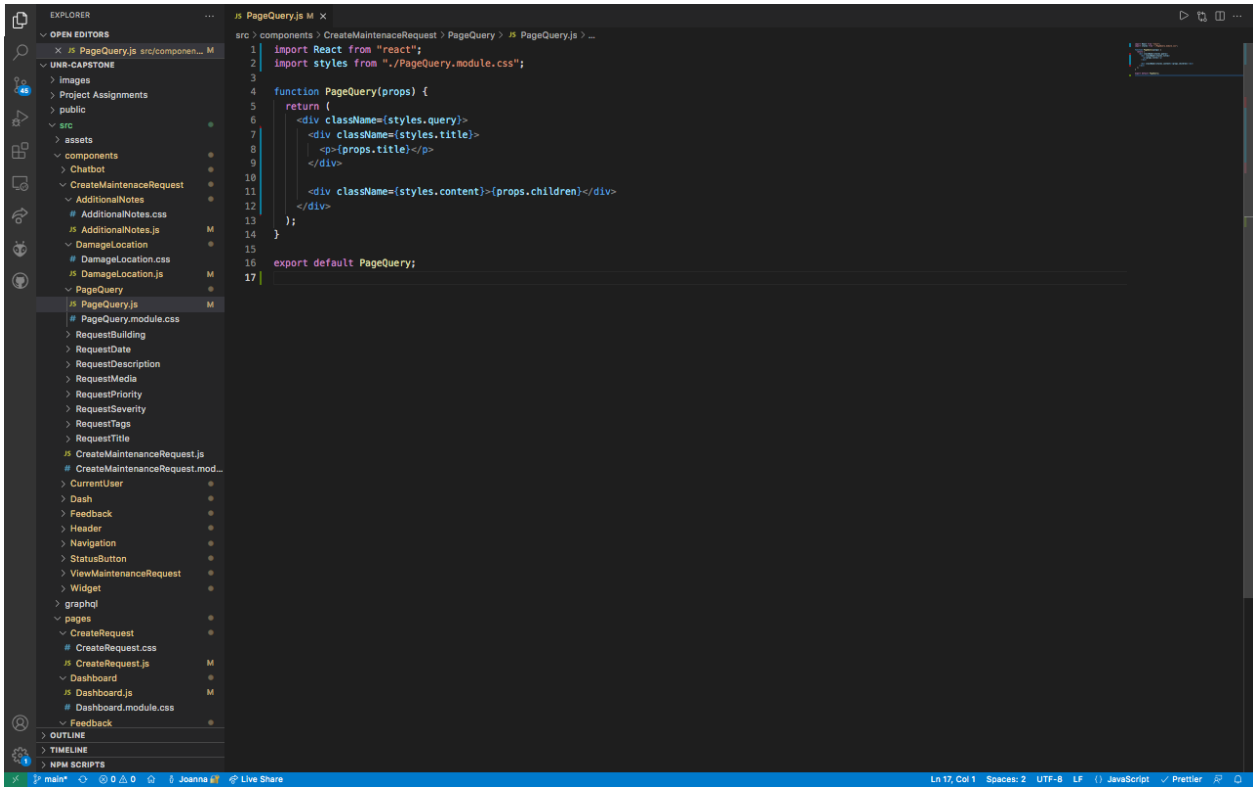


Fig. 7.1.13: A snapshot showing the PageQuery.js file.

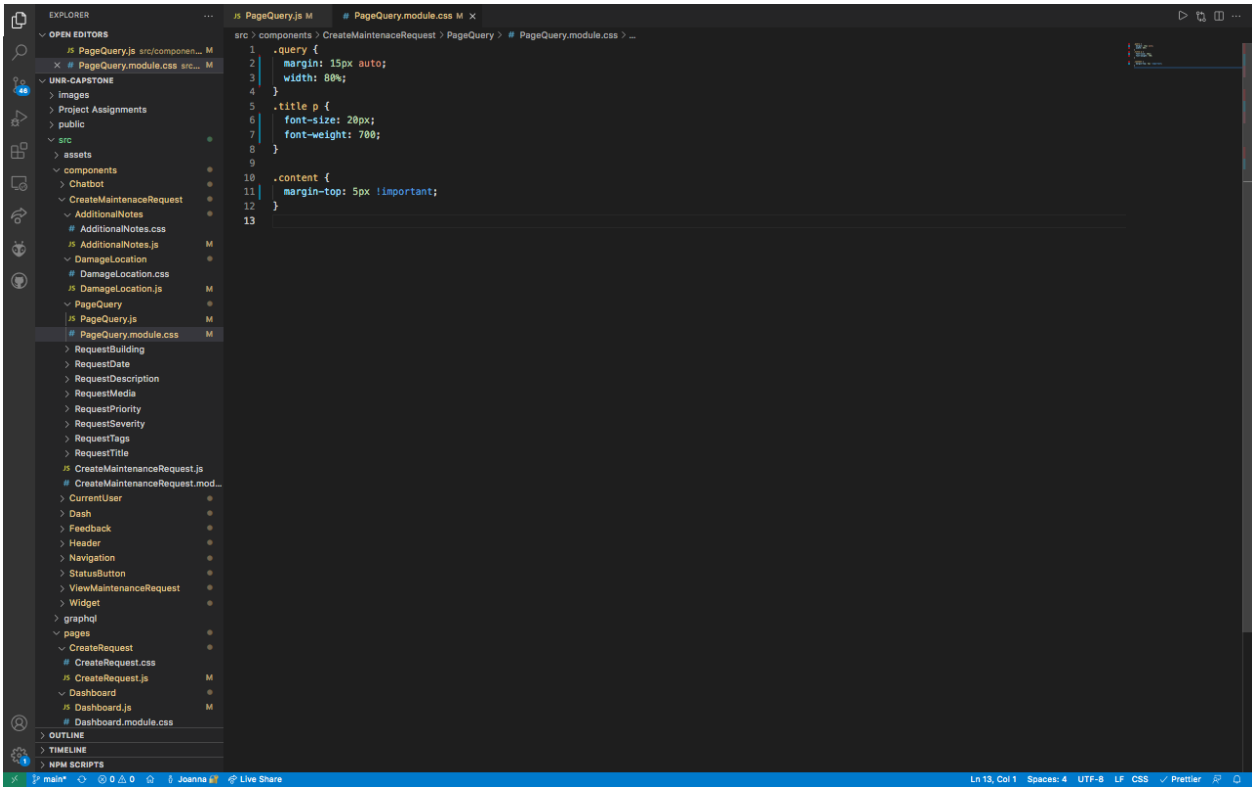
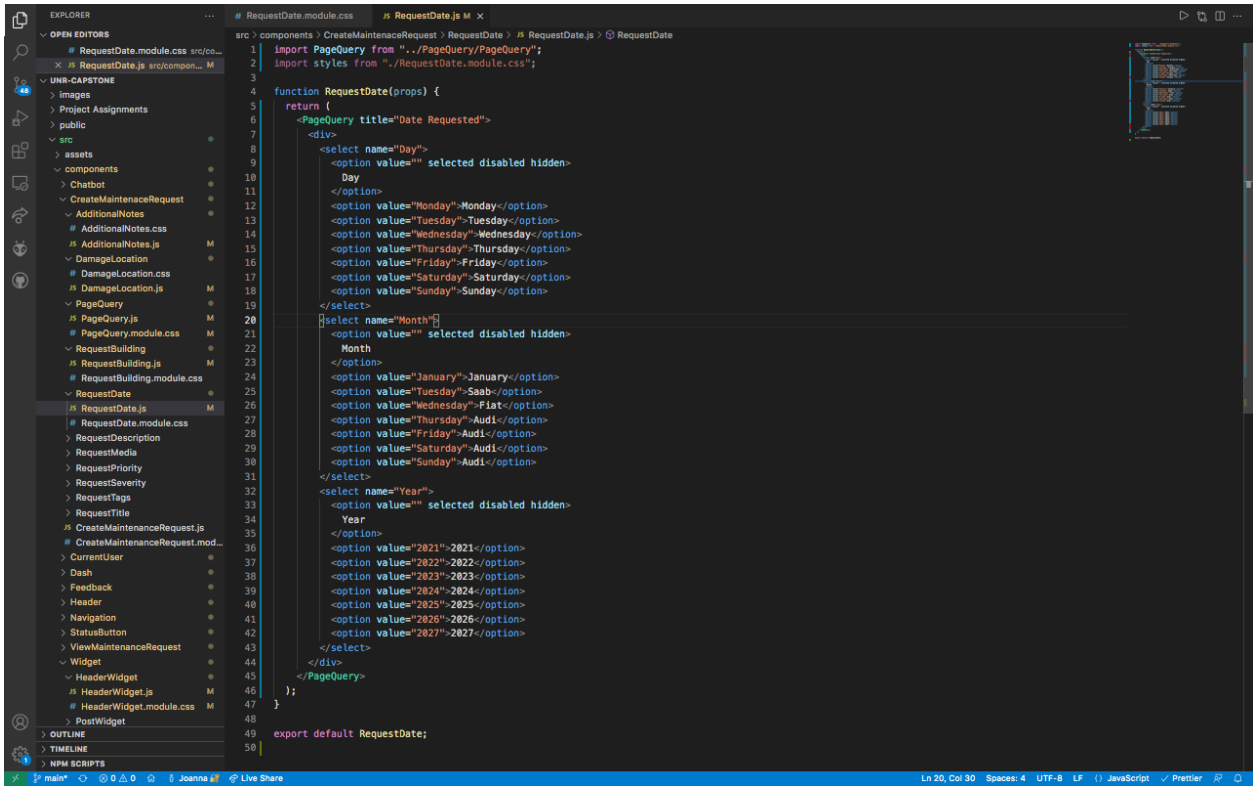


Fig 7.1.14: A snapshot showing the PageQuery.module.css file.

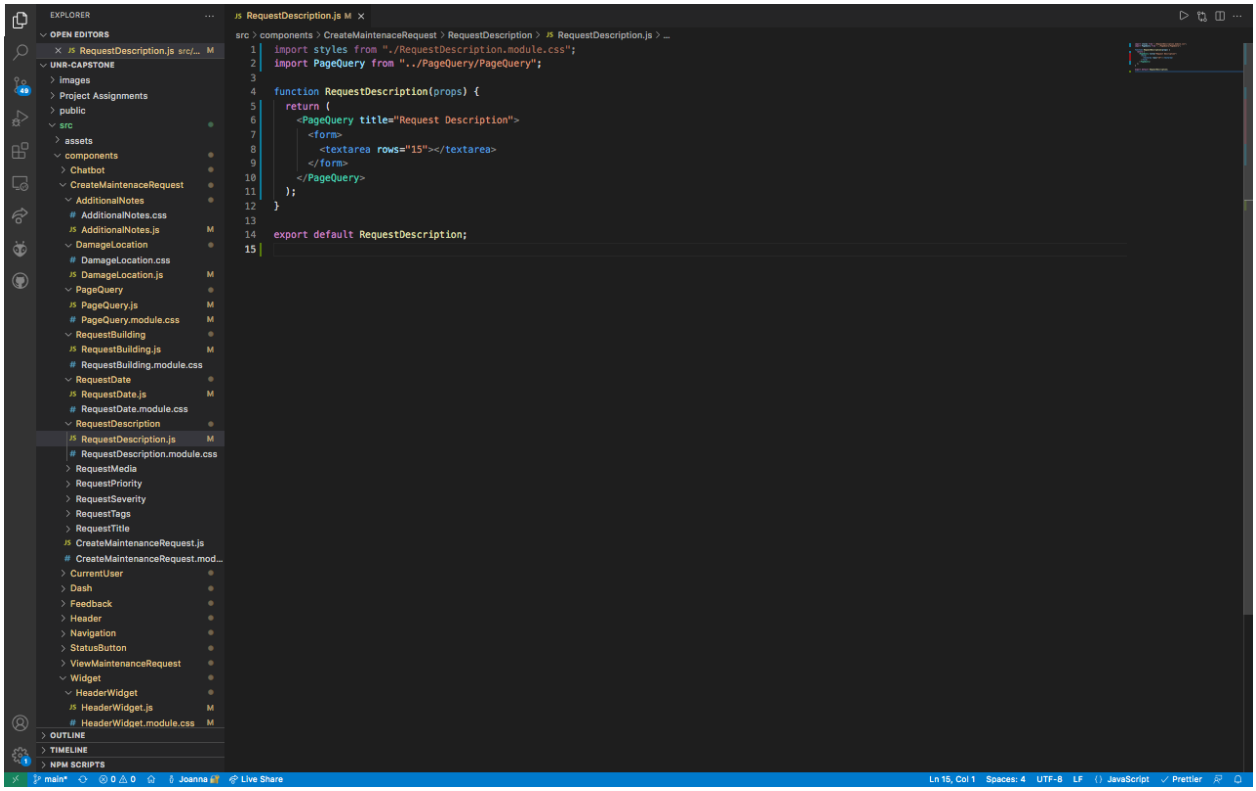


```
import PageQuery from "../PageQuery/PageQuery";
import styles from "./RequestDate.module.css";

function RequestDate(props) {
  return (
    <PageQuery title="Date Requested">
      <div>
        <select name="Day">
          <option value="" selected disabled hidden>
            Day
          </option>
          <option value="Monday">Monday</option>
          <option value="Tuesday">Tuesday</option>
          <option value="Wednesday">Wednesday</option>
          <option value="Thursday">Thursday</option>
          <option value="Friday">Friday</option>
          <option value="Saturday">Saturday</option>
          <option value="Sunday">Sunday</option>
        </select>
        <select name="Month">
          <option value="" selected disabled hidden>
            Month
          </option>
          <option value="January">January</option>
          <option value="Tuesday">Saab</option>
          <option value="Wednesday">Fiat</option>
          <option value="Thursday">Audi</option>
          <option value="Friday">Audi</option>
          <option value="Saturday">Audi</option>
          <option value="Sunday">Audi</option>
        </select>
        <select name="Year">
          <option value="" selected disabled hidden>
            Year
          </option>
          <option value="2021">2021</option>
          <option value="2022">2022</option>
          <option value="2023">2023</option>
          <option value="2024">2024</option>
          <option value="2025">2025</option>
          <option value="2026">2026</option>
          <option value="2027">2027</option>
        </select>
      </div>
    </PageQuery>
  );
}

export default RequestDate;
```

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



```
import styles from "./RequestDescription.module.css";
import PageQuery from "../PageQuery/PageQuery";

function RequestDescription(props) {
  return (
    <PageQuery title="Request Description">
      <form>
        <textarea rows="15"></textarea>
      </form>
    </PageQuery>
  );
}

export default RequestDescription;
```

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

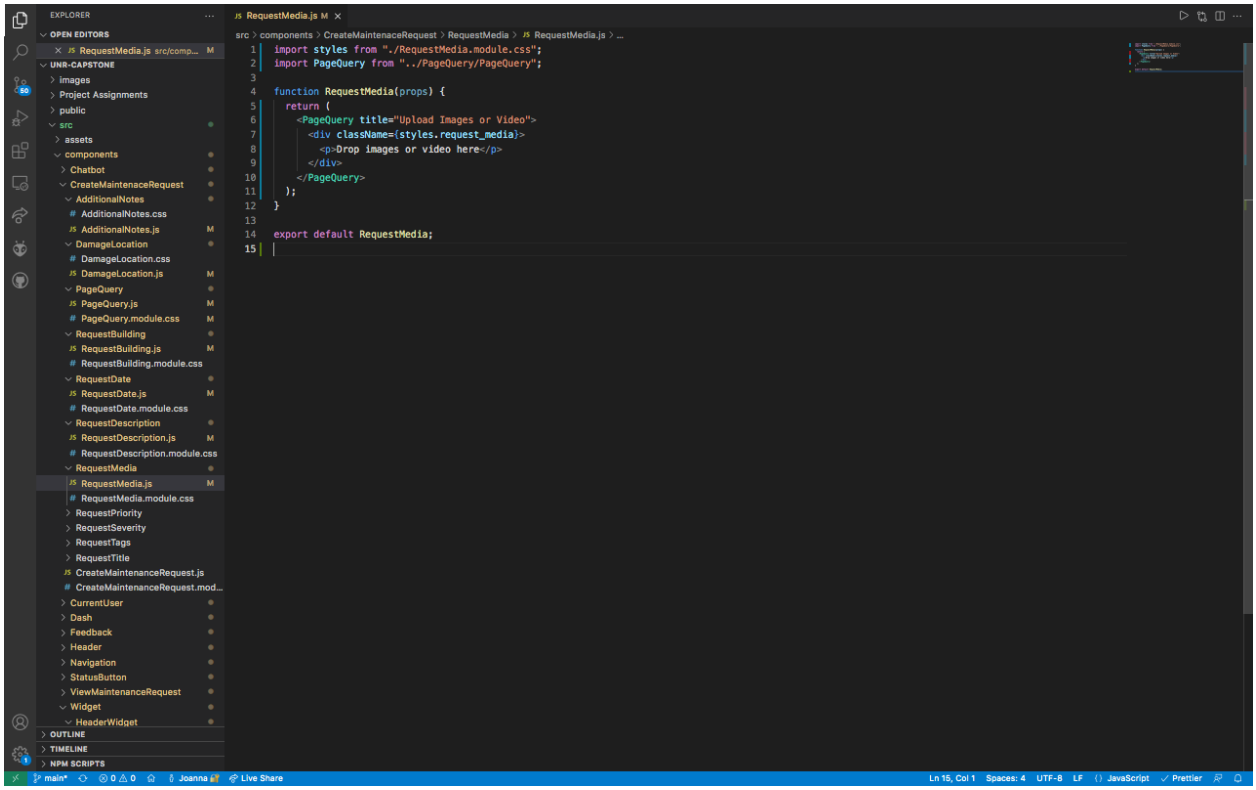


Fig. 7.1.17: A snapshot showing the RequestMedia.js file.

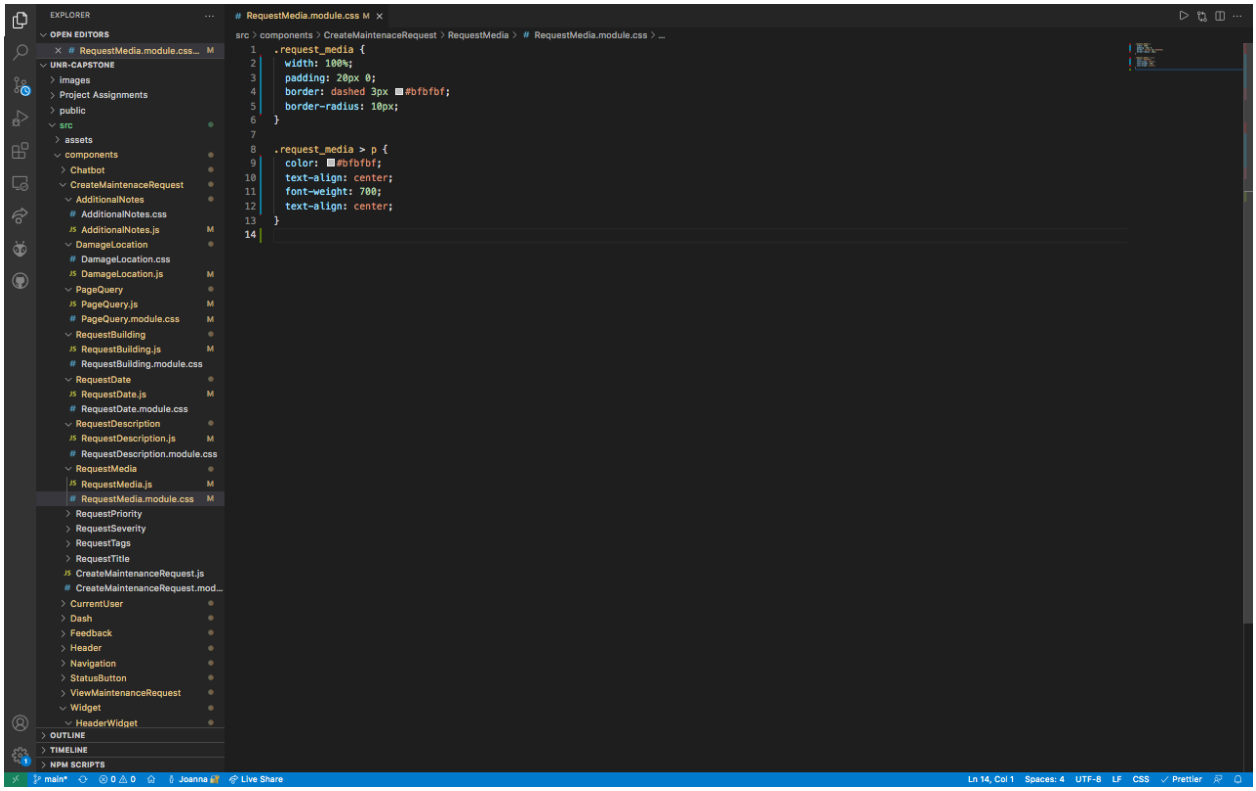


Fig. 7.1.18: A snapshot showing the RequestMedia.module.css file.

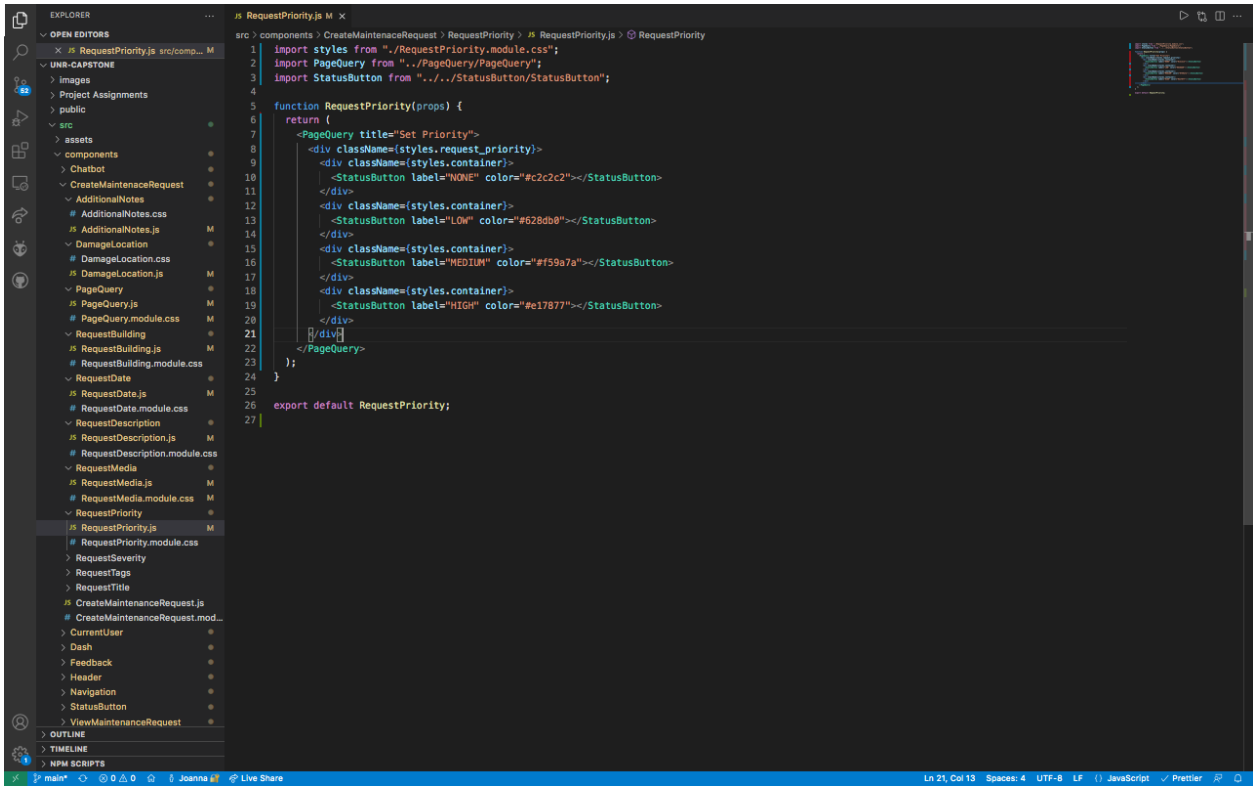


Fig. 7.1.19: A snapshot showing the RequestPriority.js file.

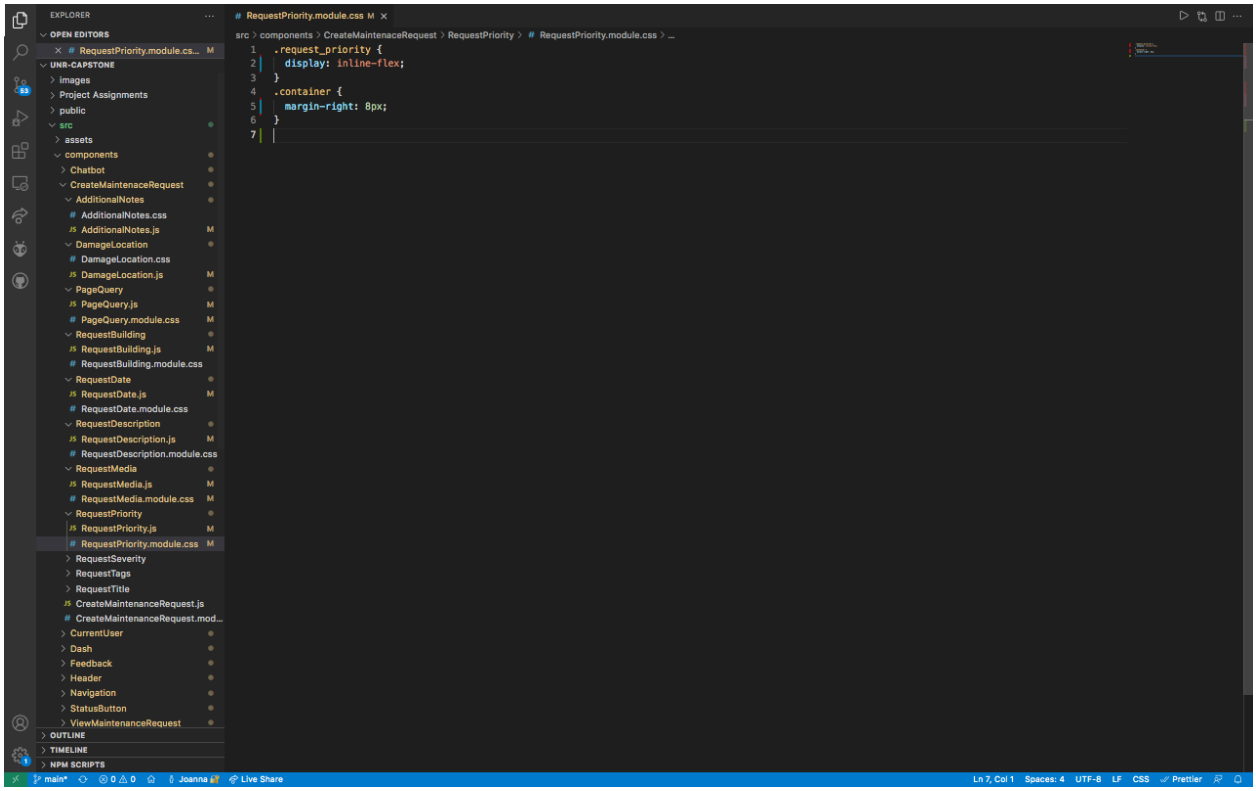


Fig. 7.1.20: A snapshot showing the RequestPriority.module.css file.

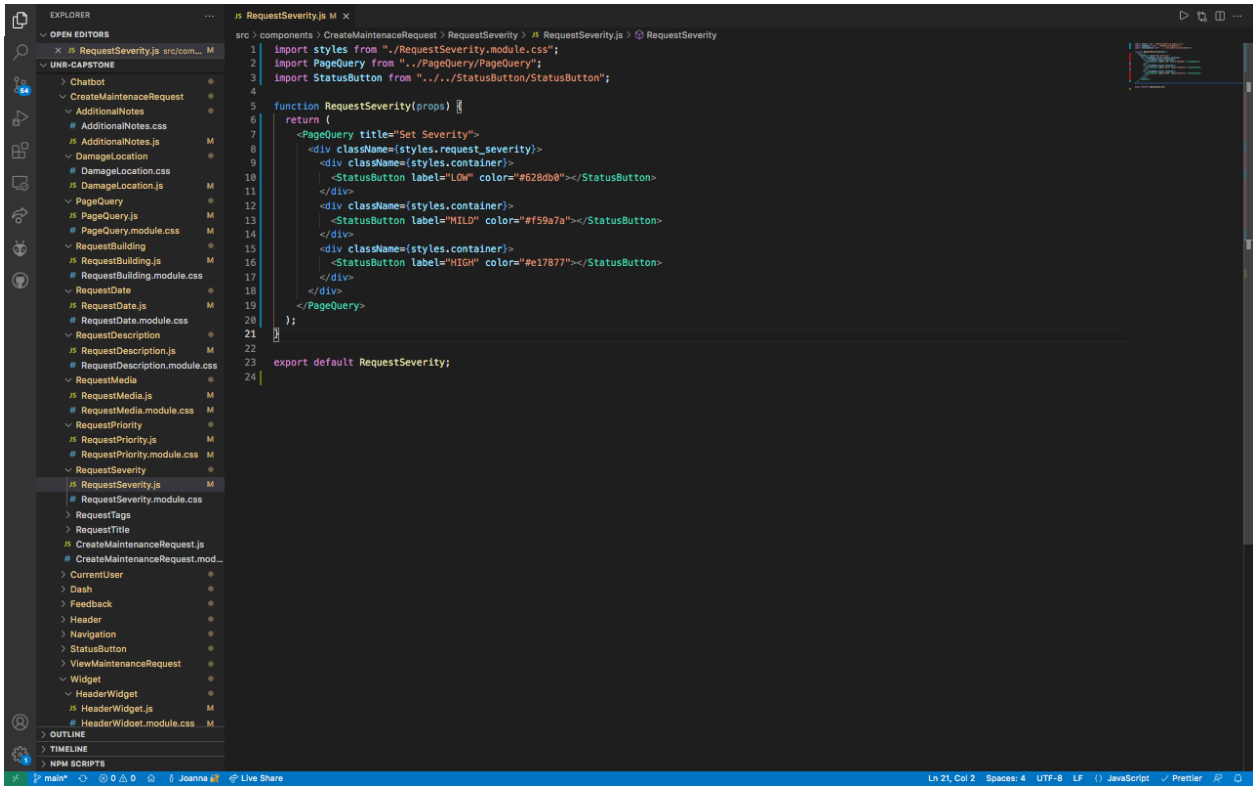


Fig. 7.1.21: A snapshot showing the RequestSeverity.js file.

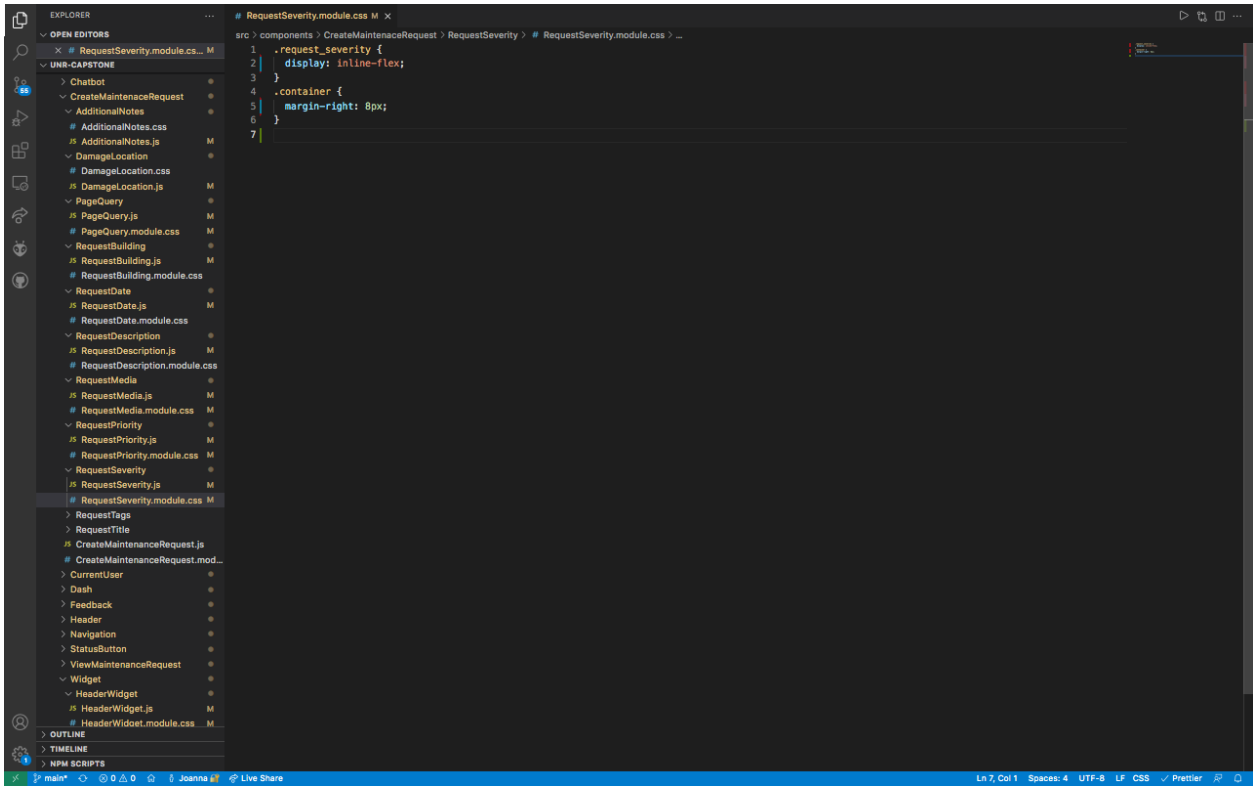


Fig. 7.1.22: A snapshot showing the RequestSeverity.module.css file.

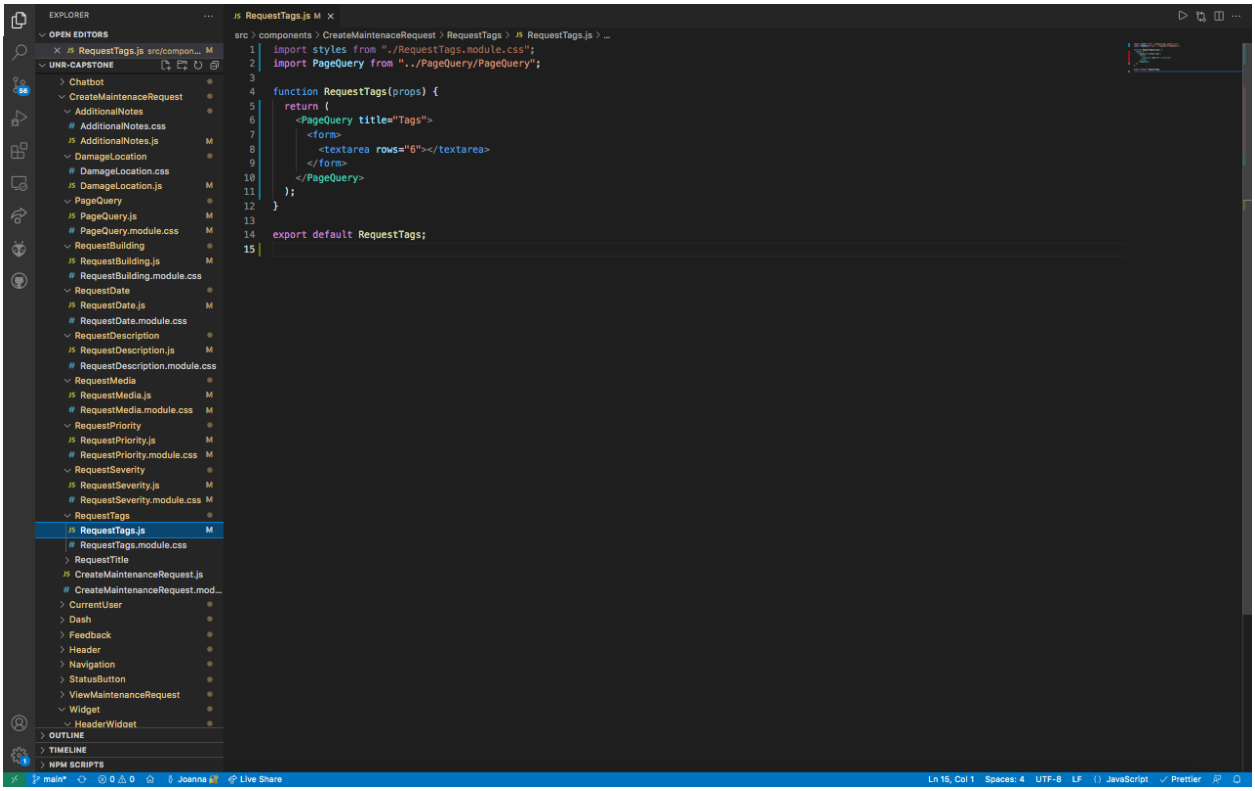


Fig 7.1.23: A snapshot showing the RequestTags.js file.

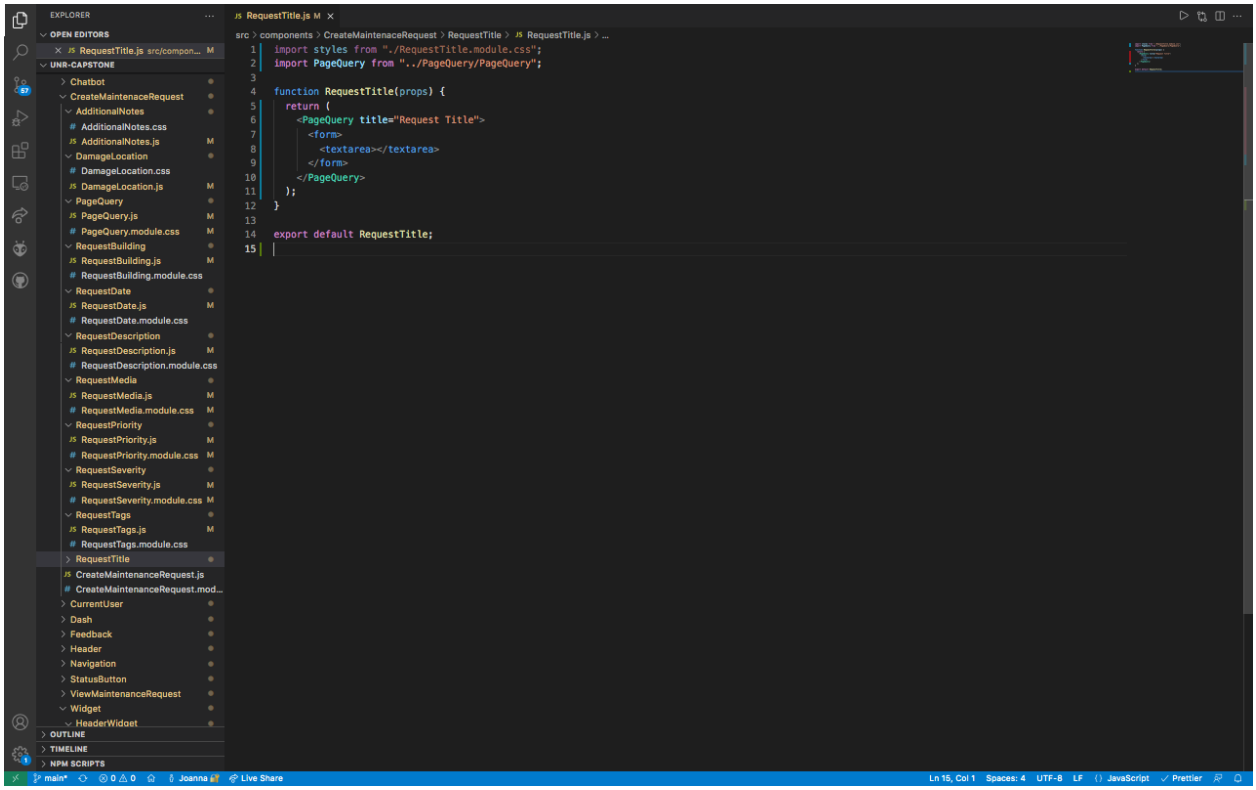


Fig. 7.1.24: A snapshot showing the RequestTitle.js file.

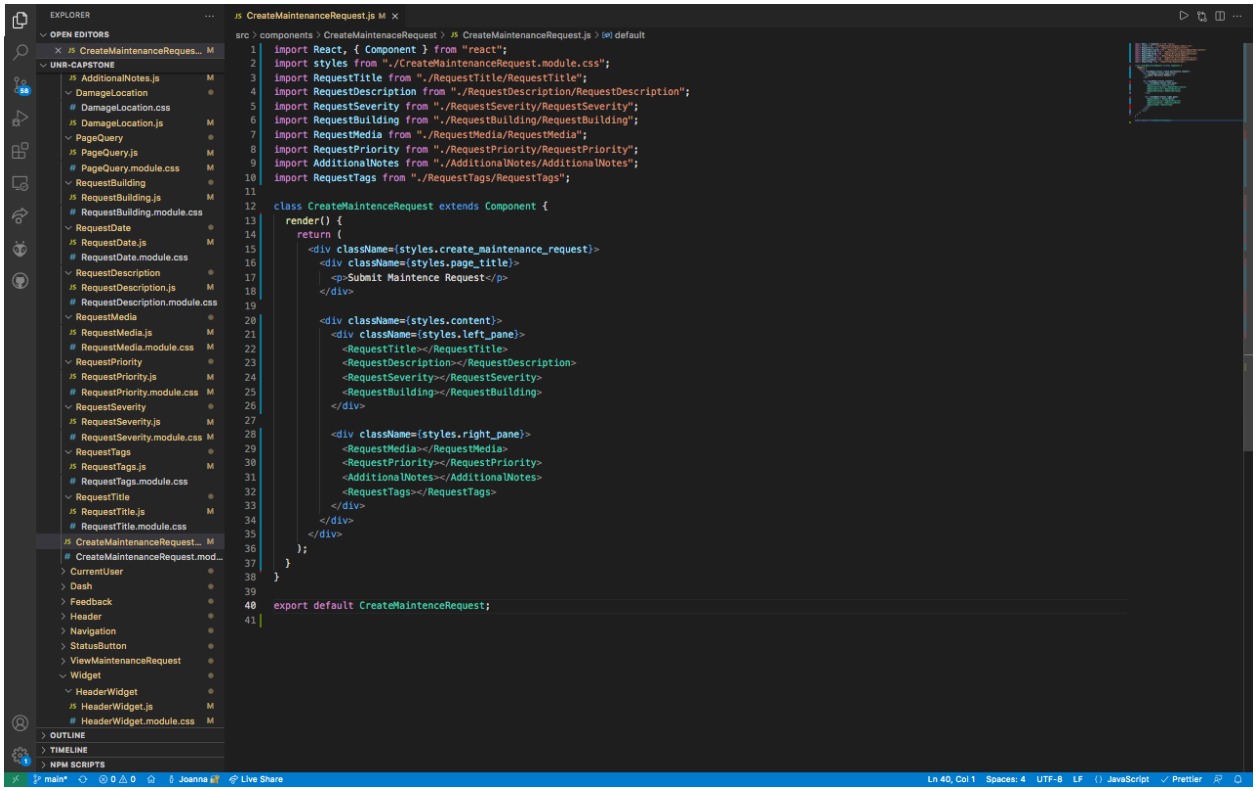


Fig. 7.1.25: A snapshot showing the CreteMaintemanceRequest.js file.

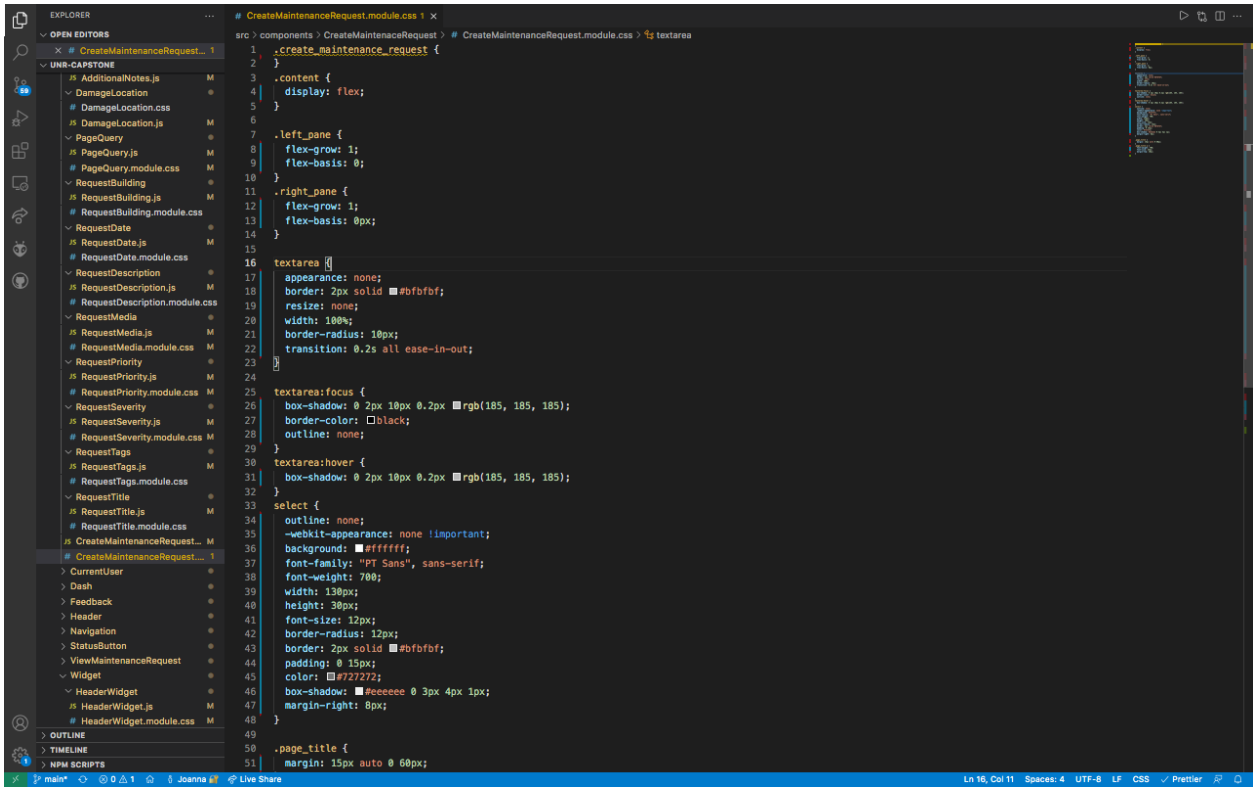


Fig. 7.1.26: A snapshot showing the CreateMaintenanceRequest.css file.

7.2. Joanna Lopez

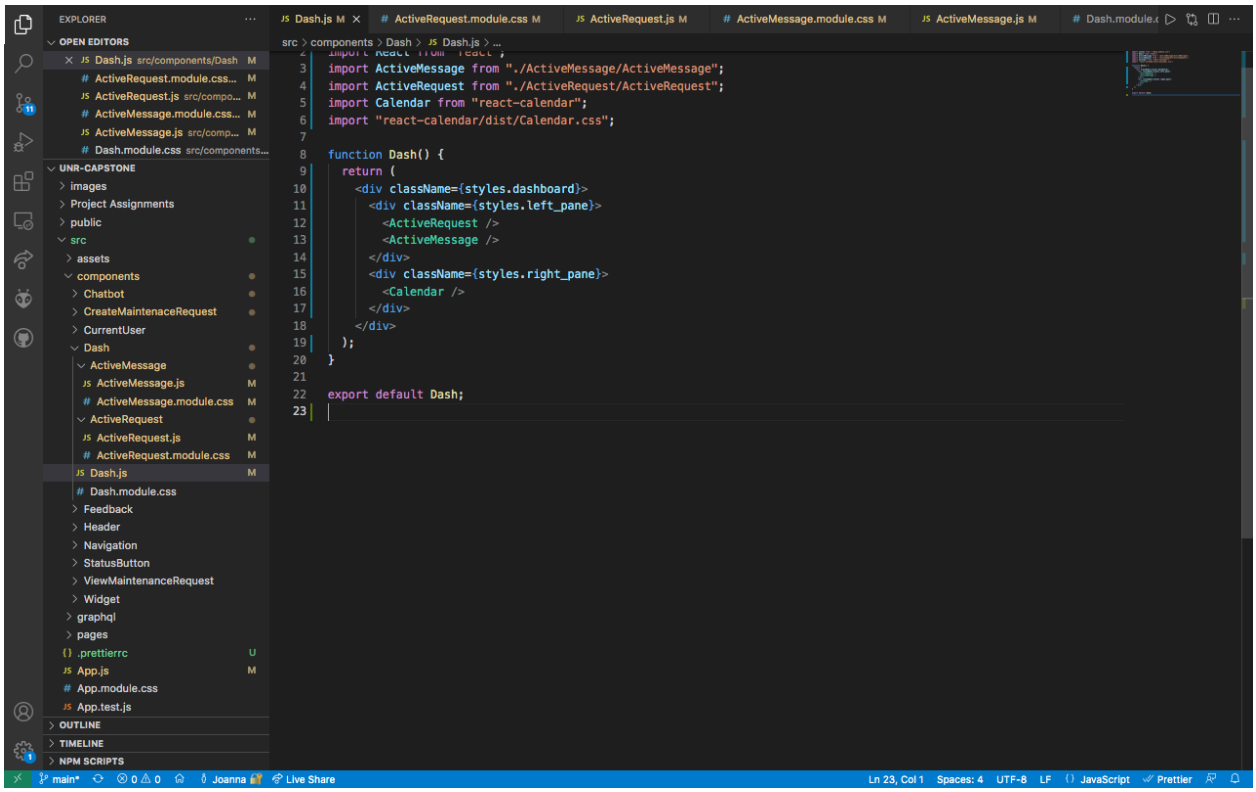


Fig. 7.2.1: A snapshot showing the Dash.js file.

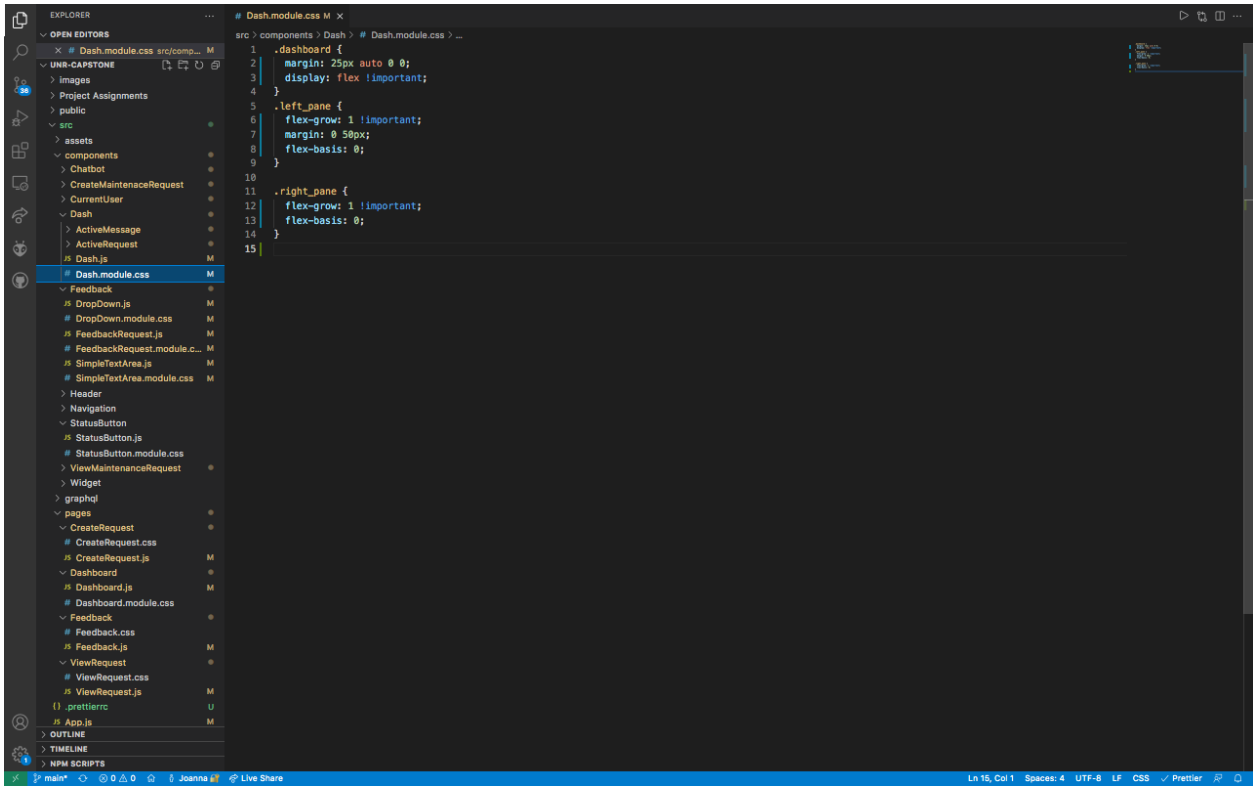


Fig. 7.2.2: A snapshot showing the Dash.module.css file.

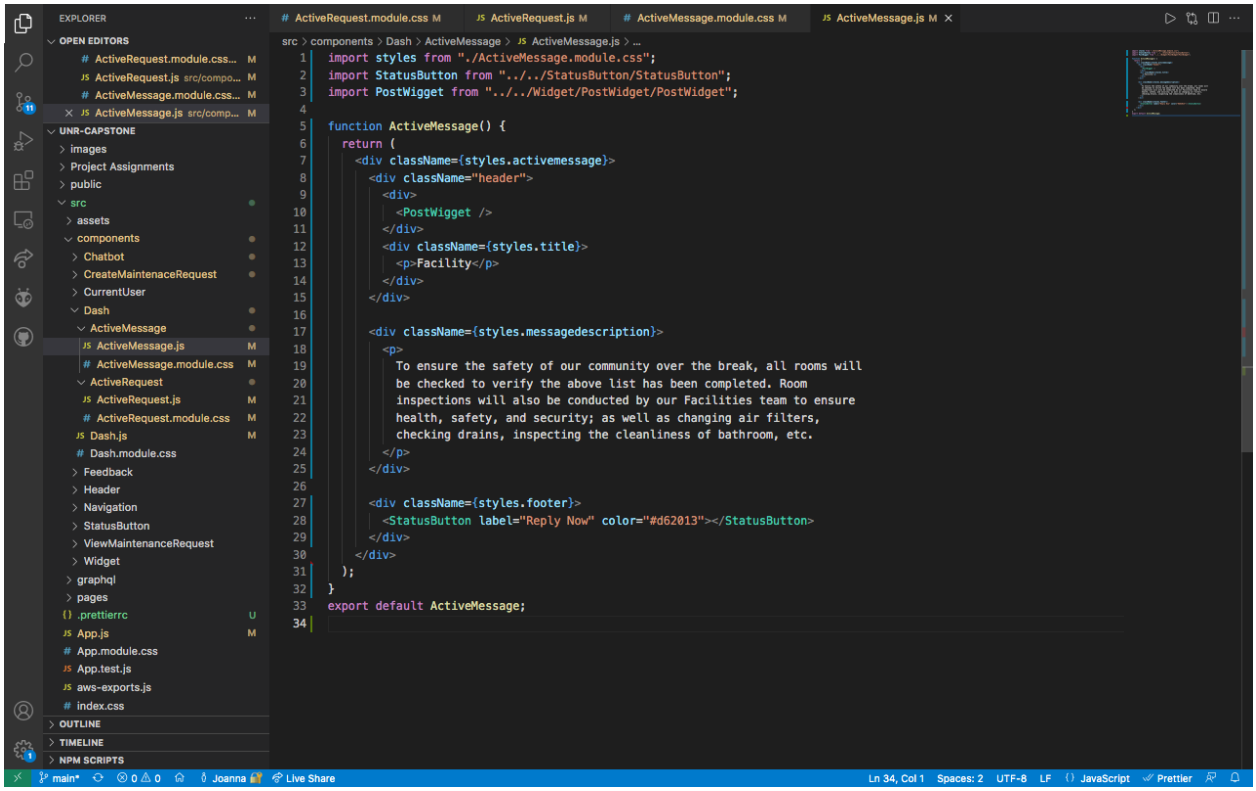


Fig. 7.2.3: A snapshot showing the ActiveMessages.js file.

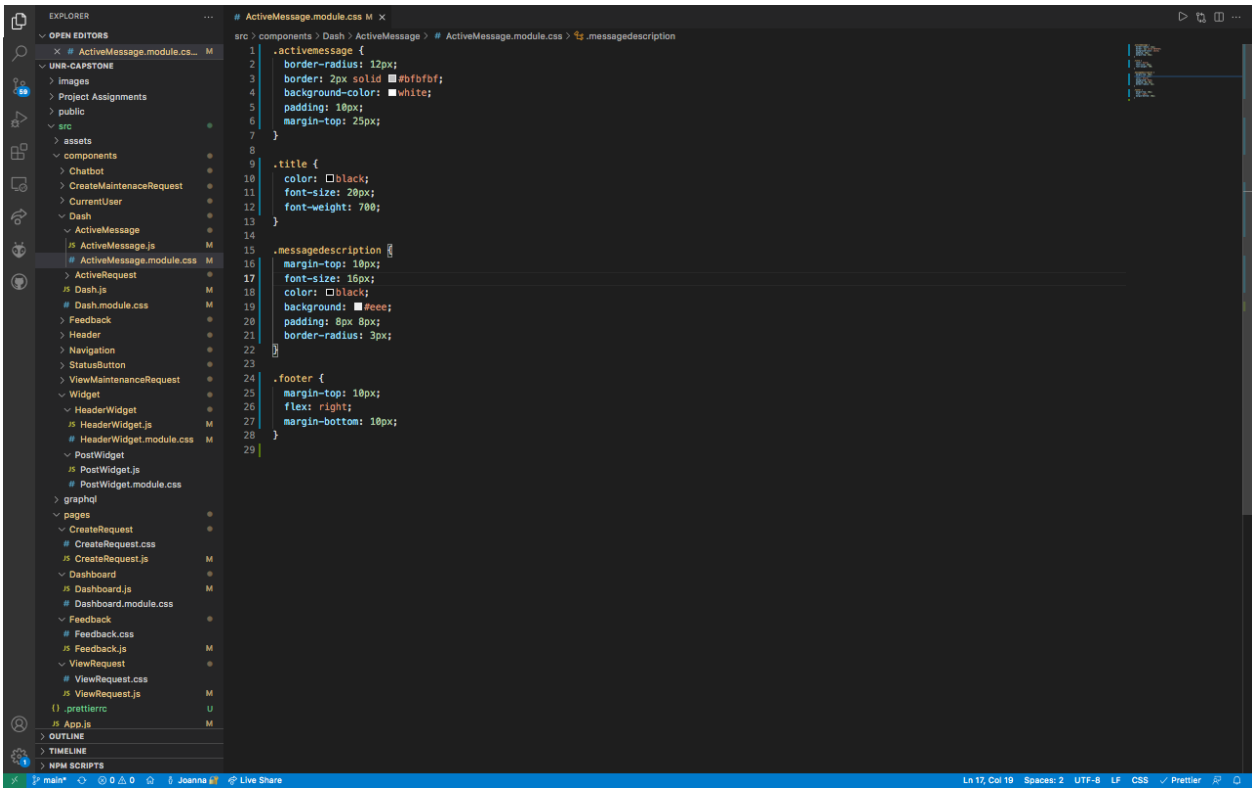


Fig. 7.2.3: A snapshot showing the ActiveMessage.module.css file.

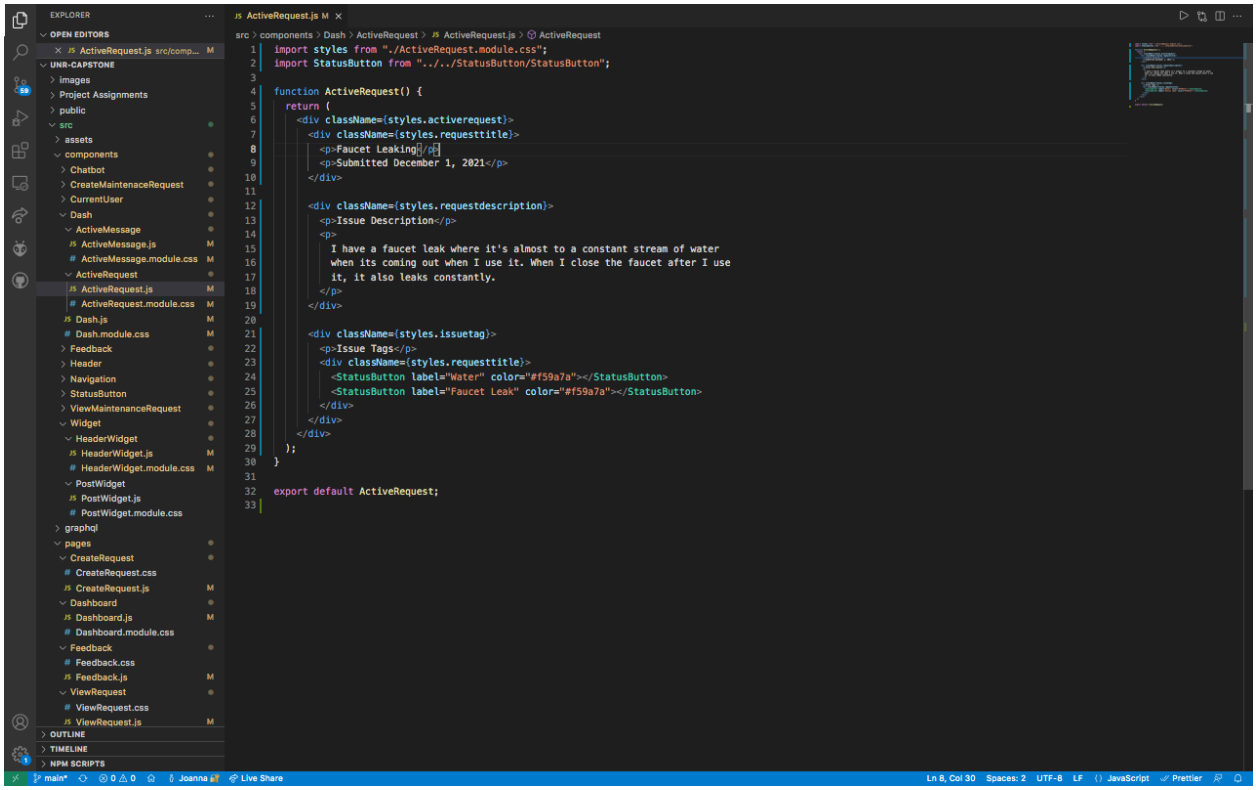


Fig. 7.2.4: A snapshot showing the ActiveRequest.js file.

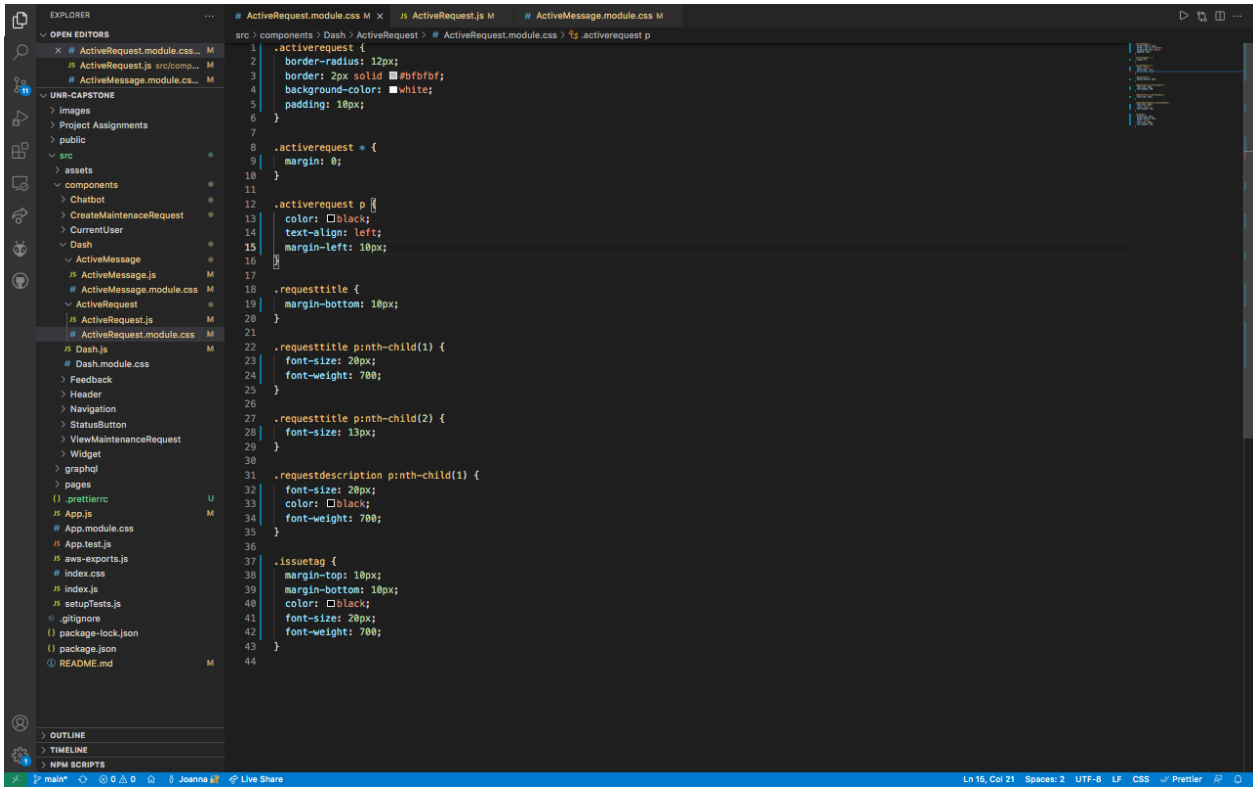


Fig.7.2.5: A snapshot showing the Active.module.css file.

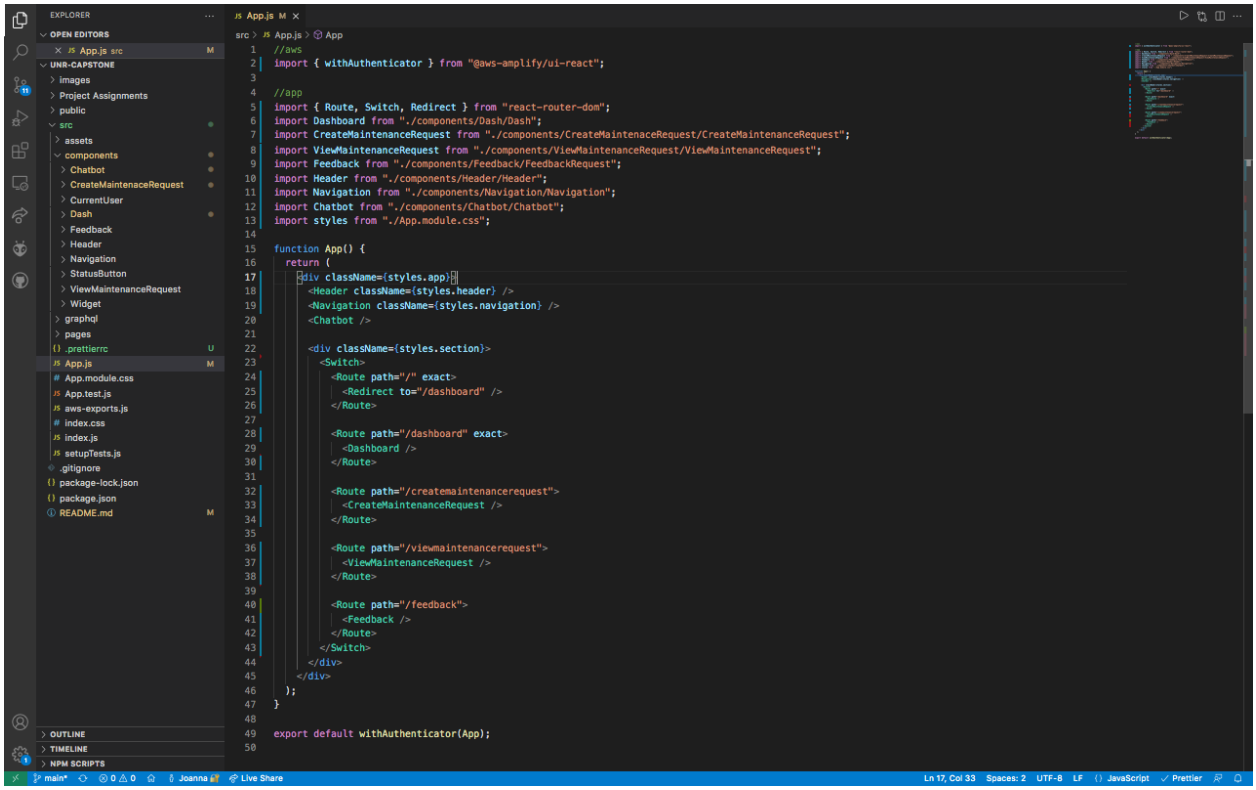


Fig. 7.2.6: A snapshot showing the App.js file.

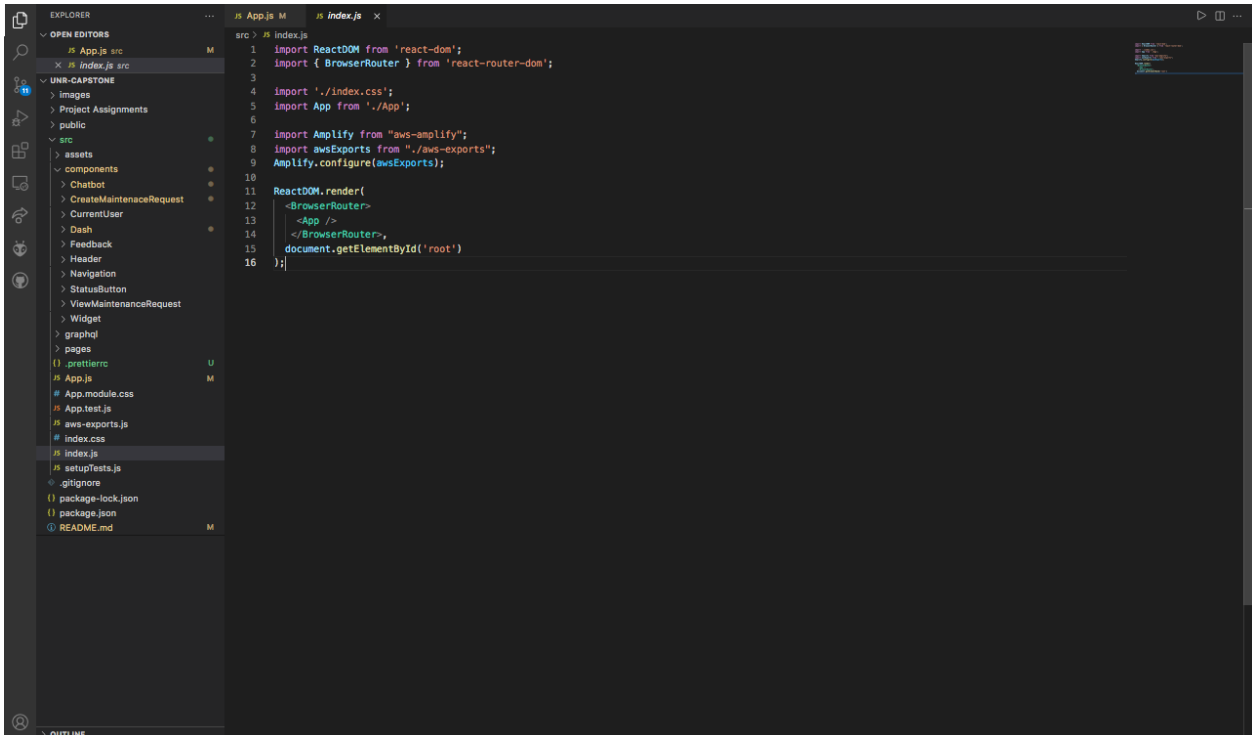


Fig. 7.2.7: A snapshot showing the index.js file.

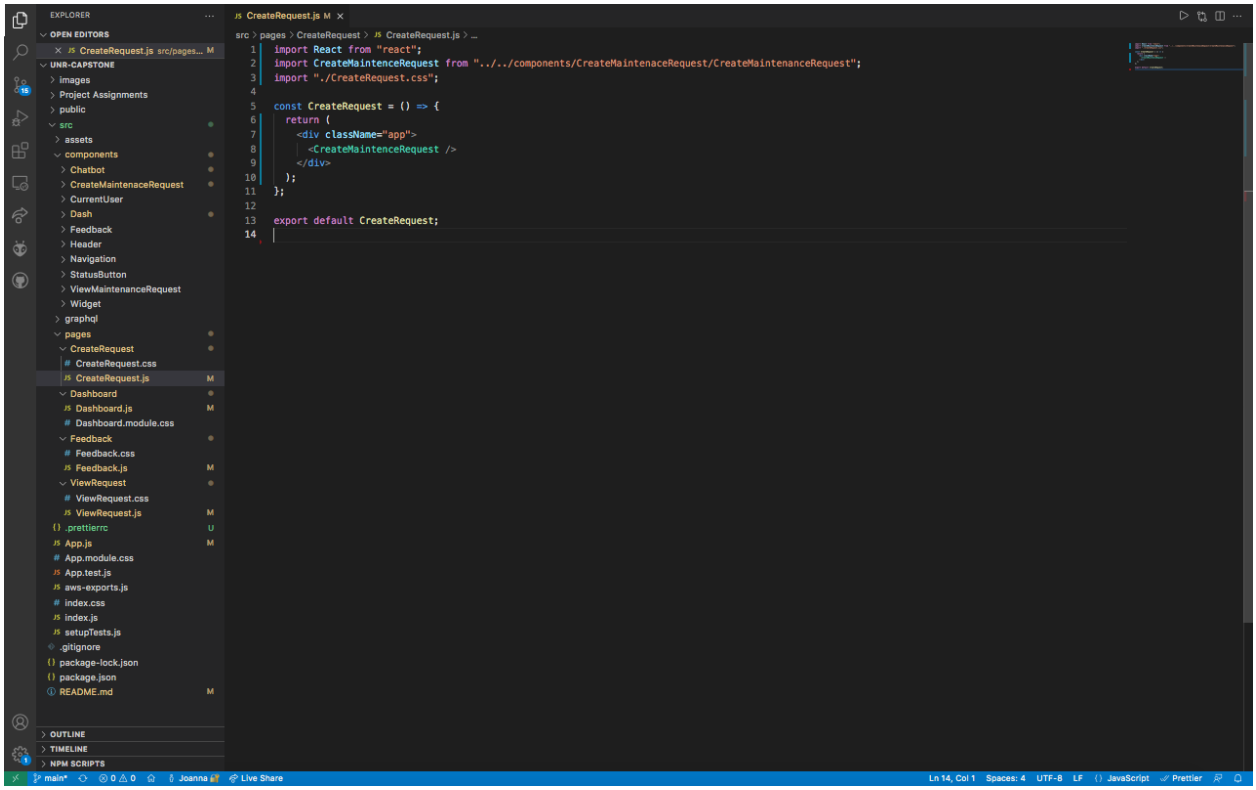


Fig. 7.2.8: A snapshot showing the CreateRequest.js file.

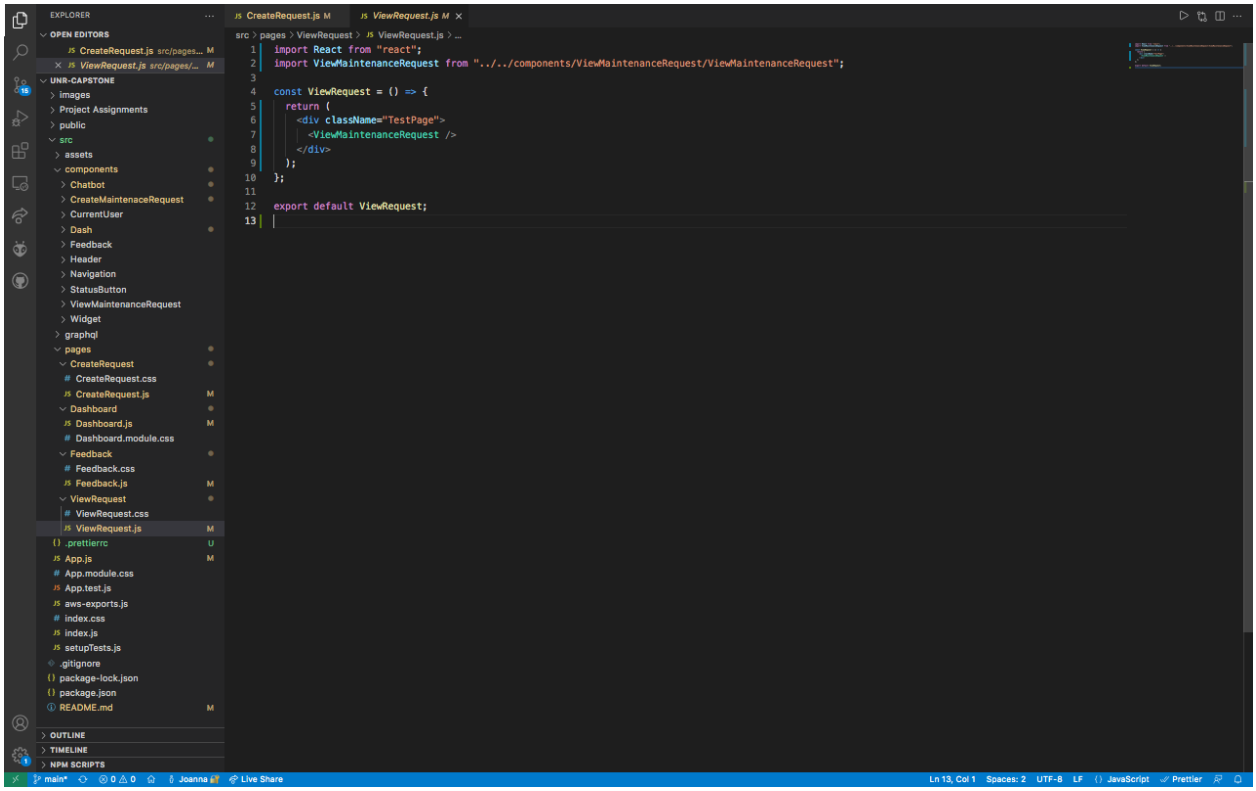


Fig. 7.2.9: A snapshot showing the viewRequest.js file.

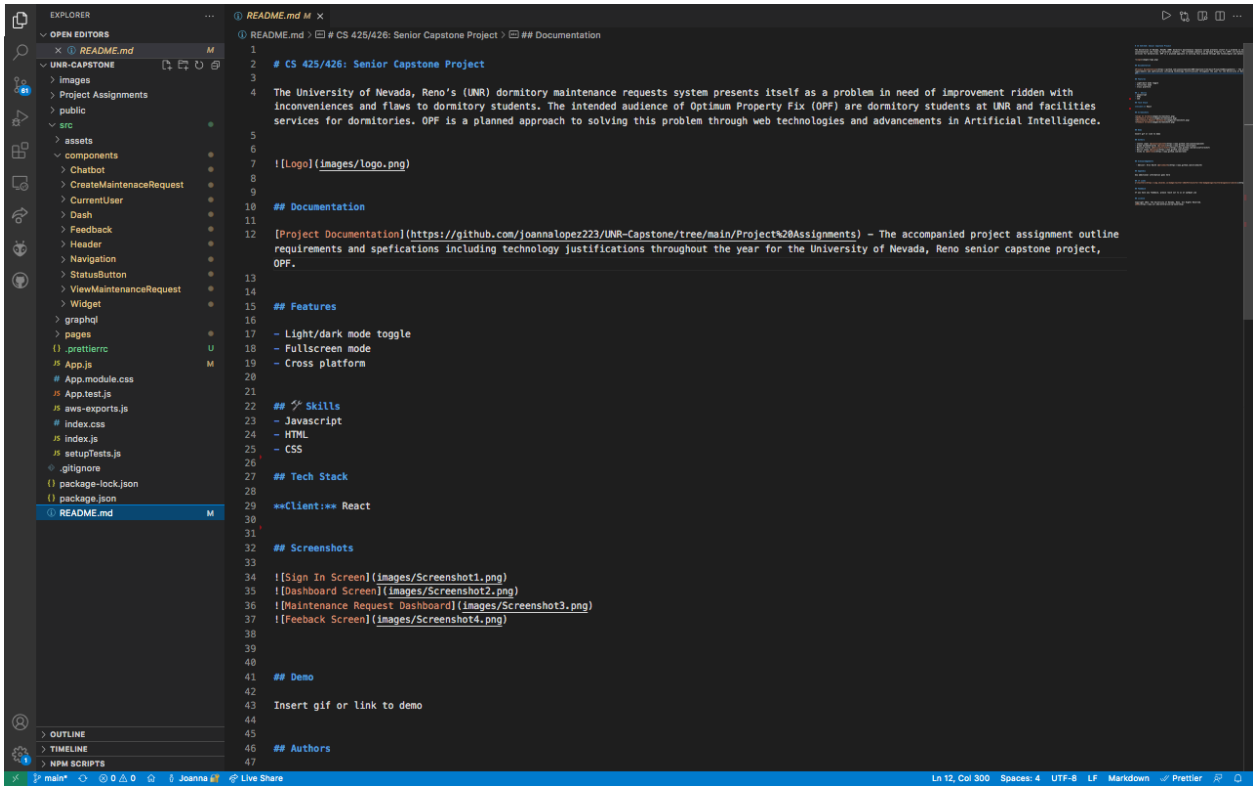


Fig. 7.2.10: A snapshot showing the README.md file.

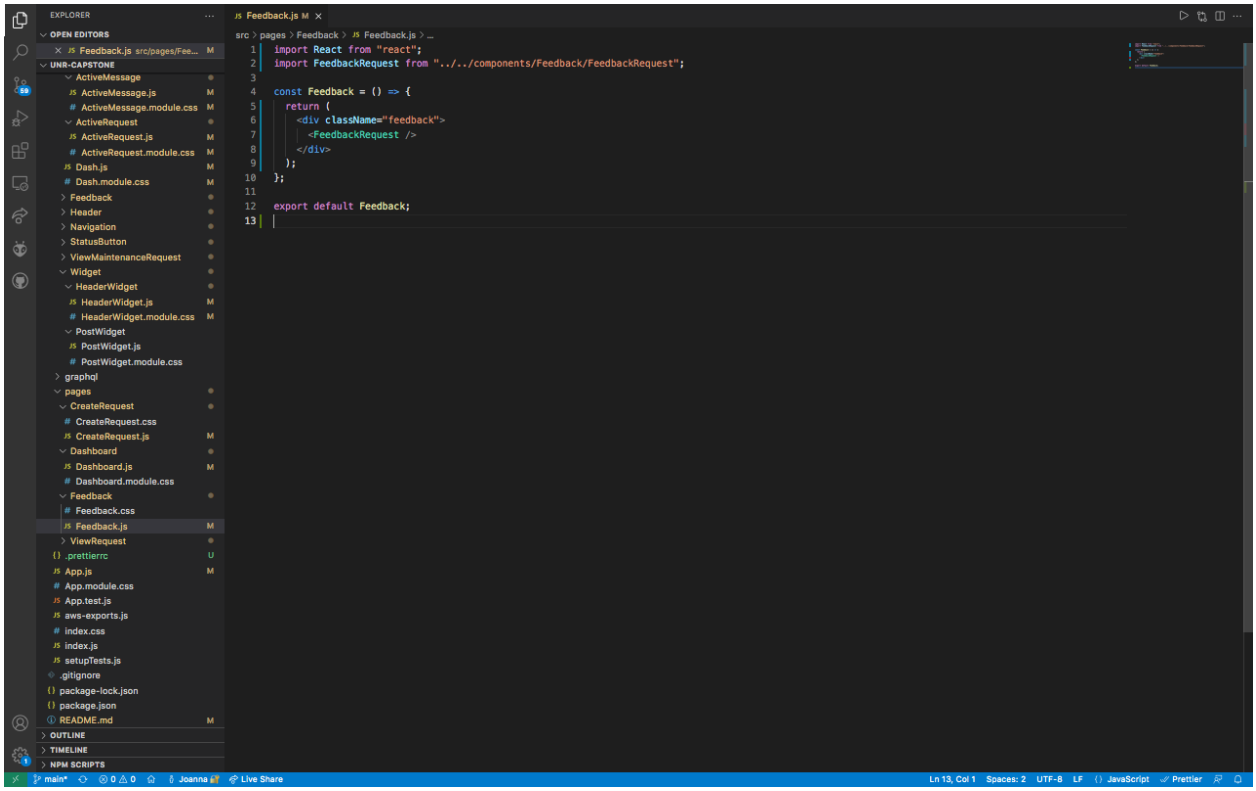


Fig 7.2.11: A snapshot showing the Feeback.js file.

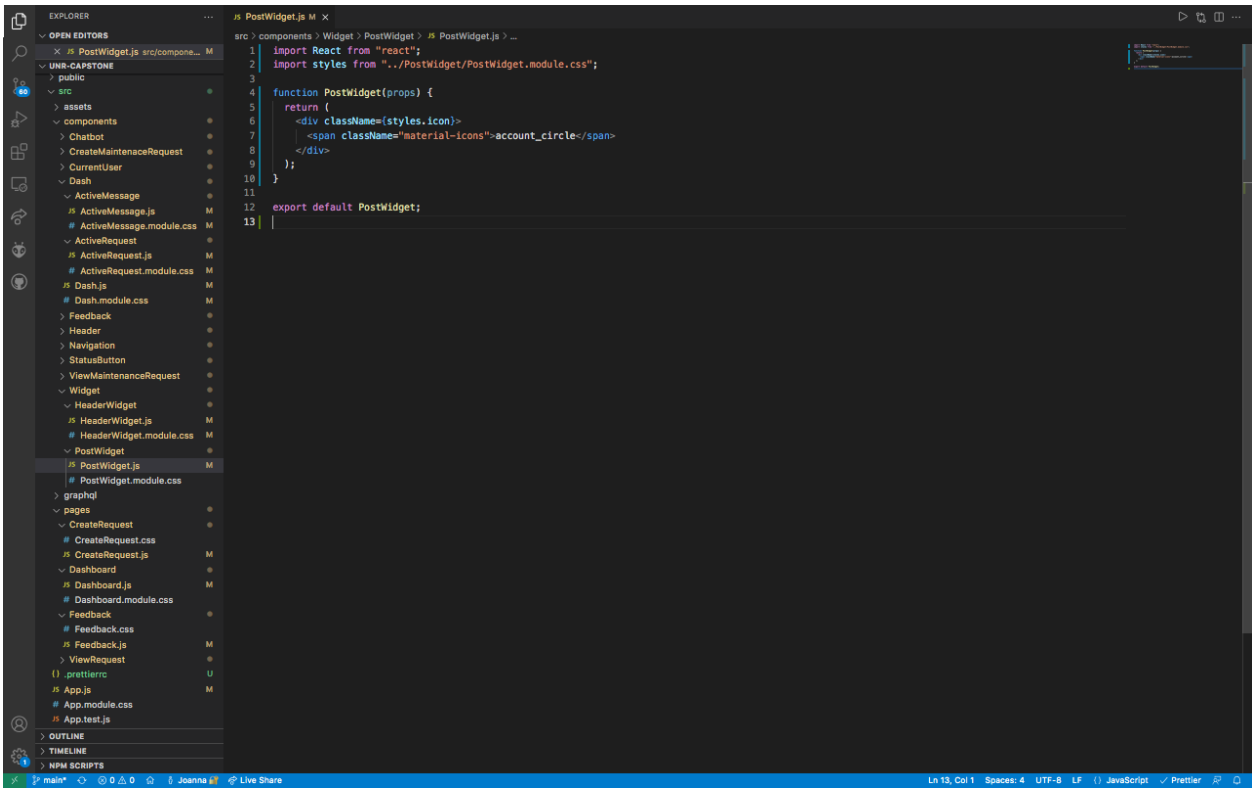


Fig. 7.2.12: A snapshot showing the PostWidget.js file.

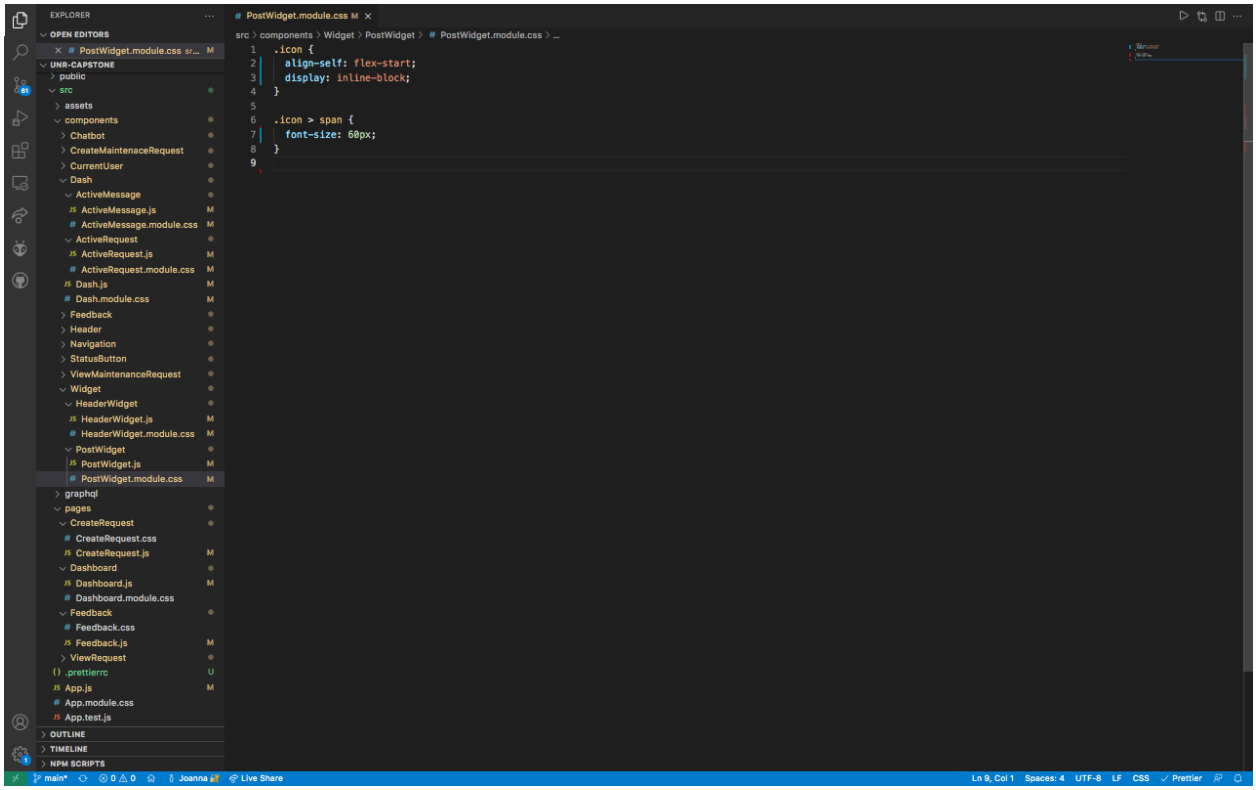


Fig. 7.2.13: A snapshot showing the PostWidget.module.css file.

7.3. Melissa Flores

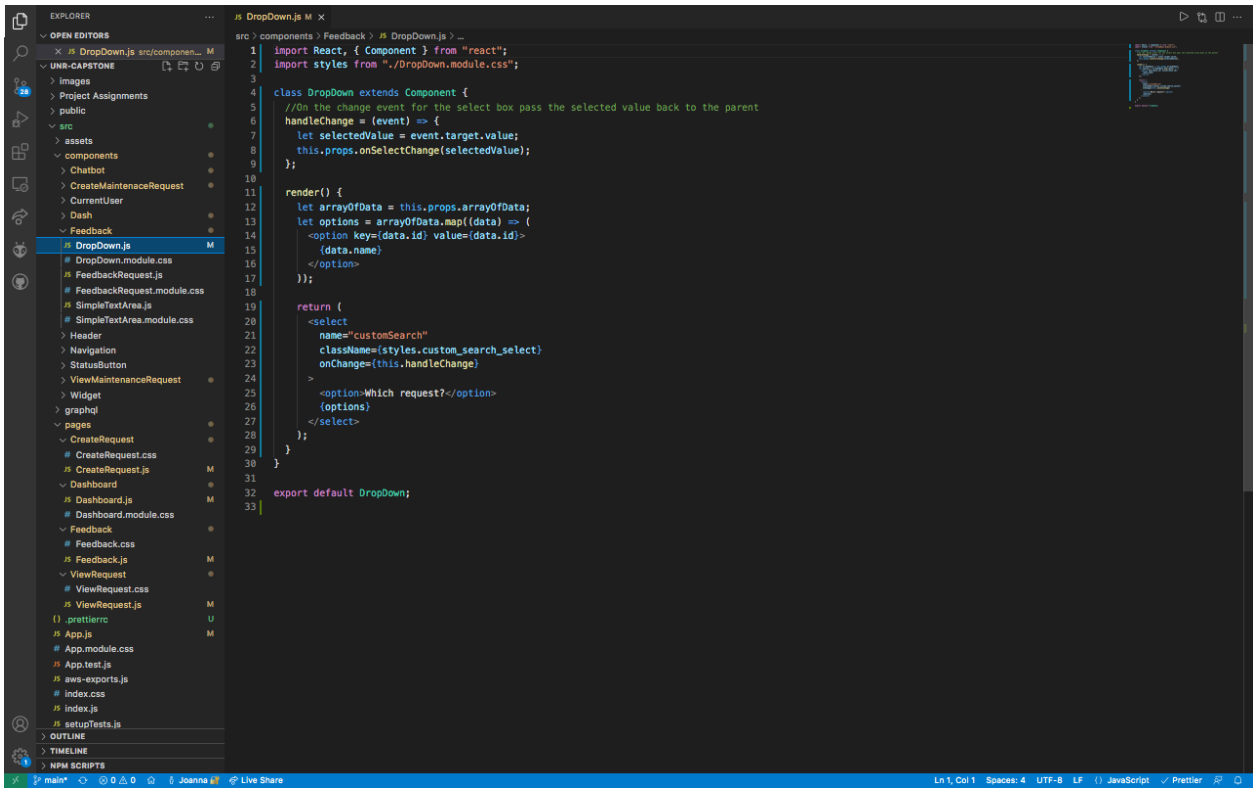


Fig. 7.3.1: A snapshot showing the Dropdown.js file.

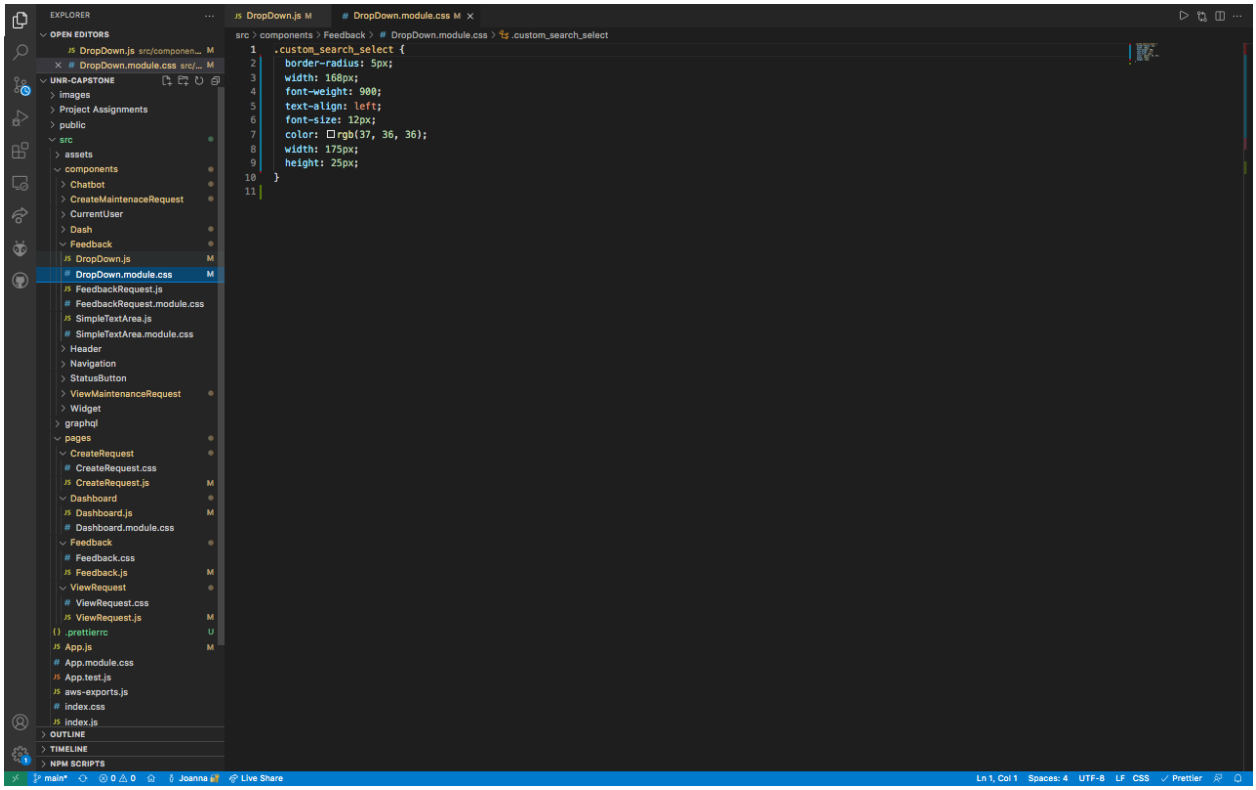


Fig. 7.3.2: A snapshot showing the DropDown.module.css file.

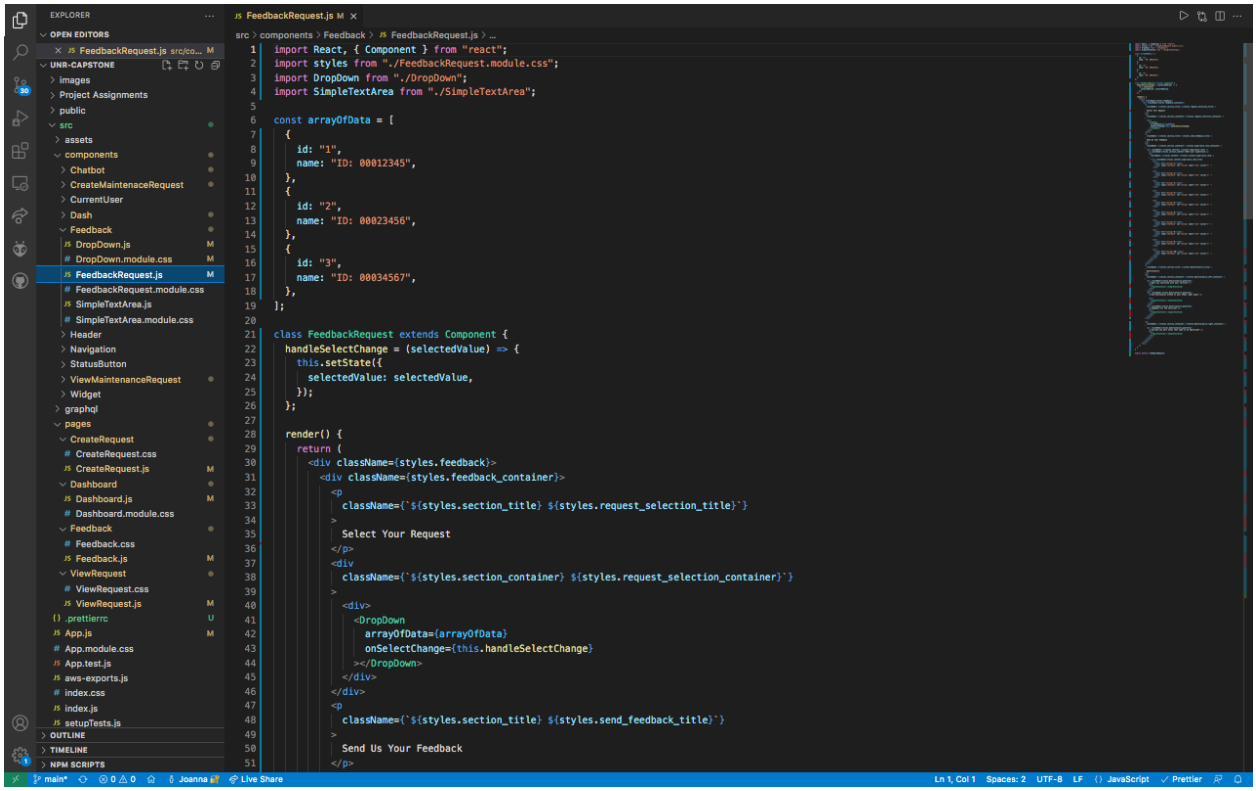


Fig. 7.3.3: A snapshot showing the FeedbackRequest.js file.

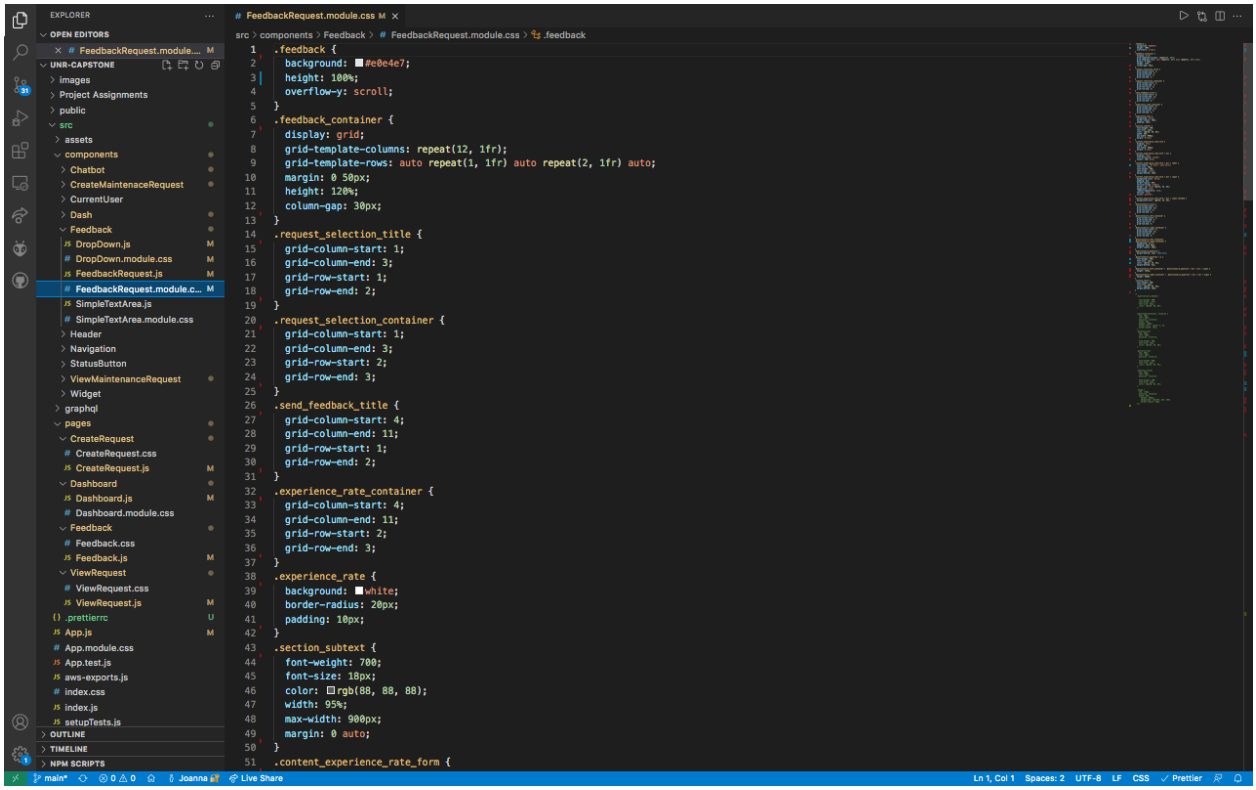


Fig. 7.3.4: A snapshot showing the FeedbackRequest.module.css file.

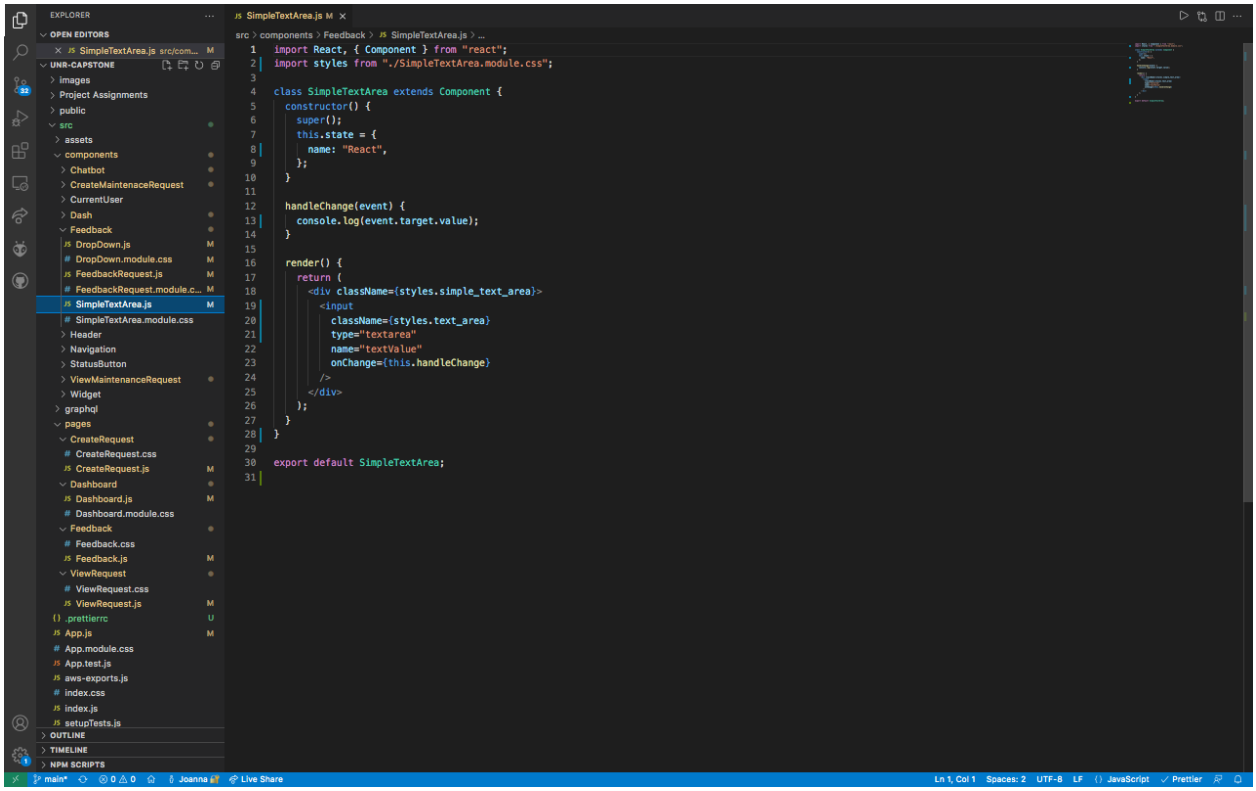


Fig 7.3.5: A snapshot showing the SimpleTextArea.js file.

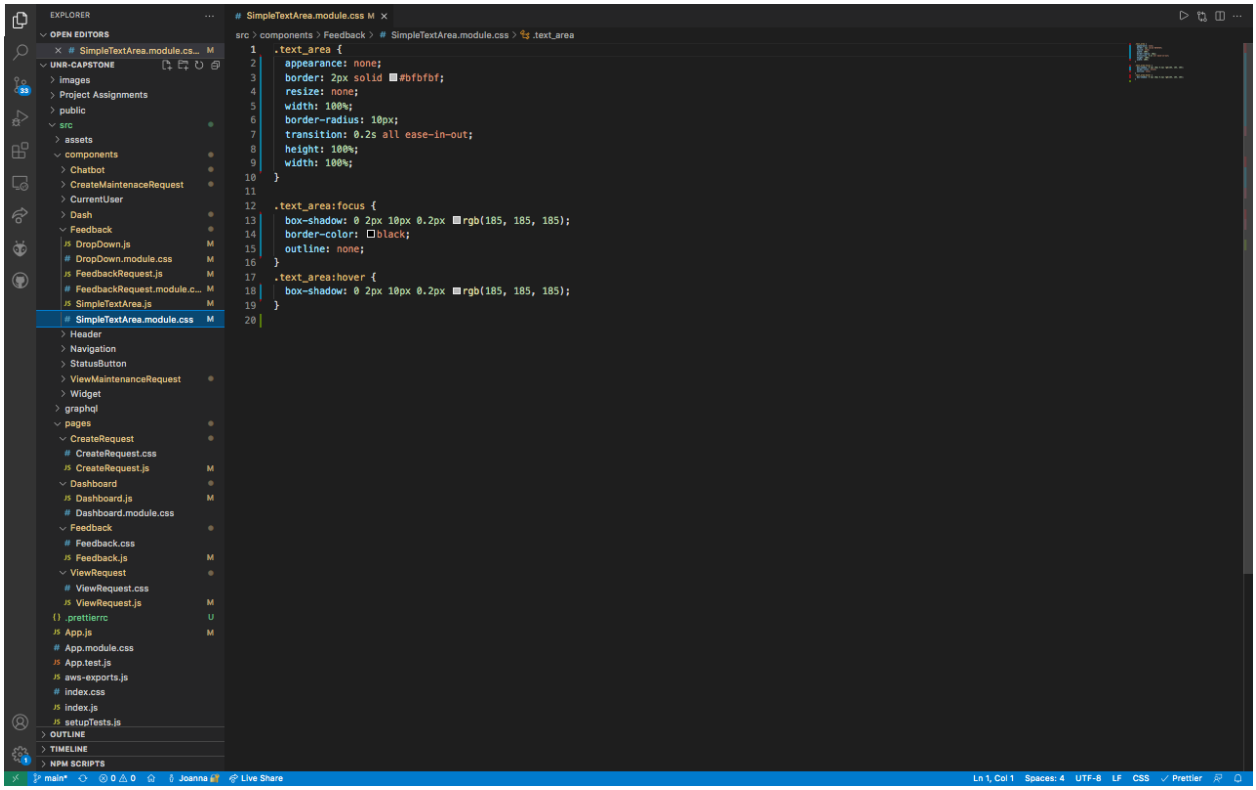


Fig. 7.3.6: A snapshot showing the SimpleTextArea.module.css file.

7.4. Nasrin Juana

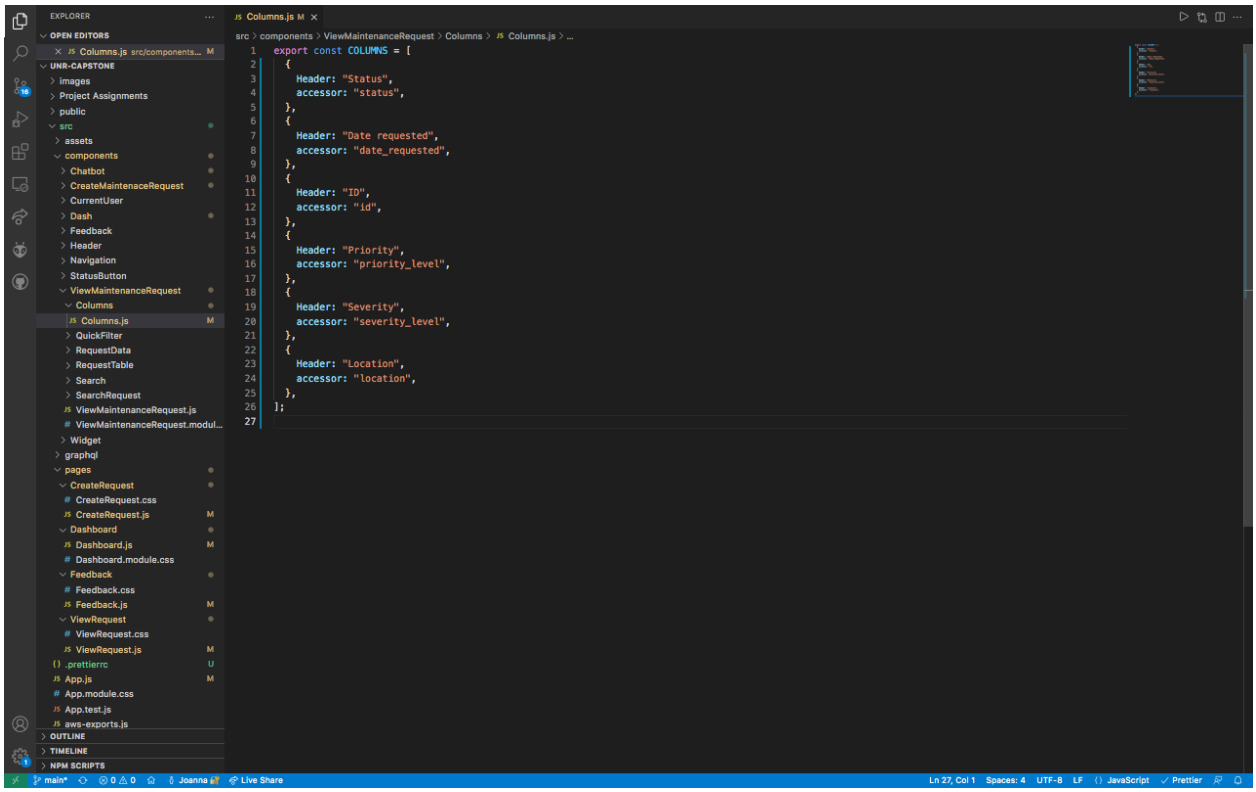


Fig. 7.4.1: A snapshot showing the Columns.js file.

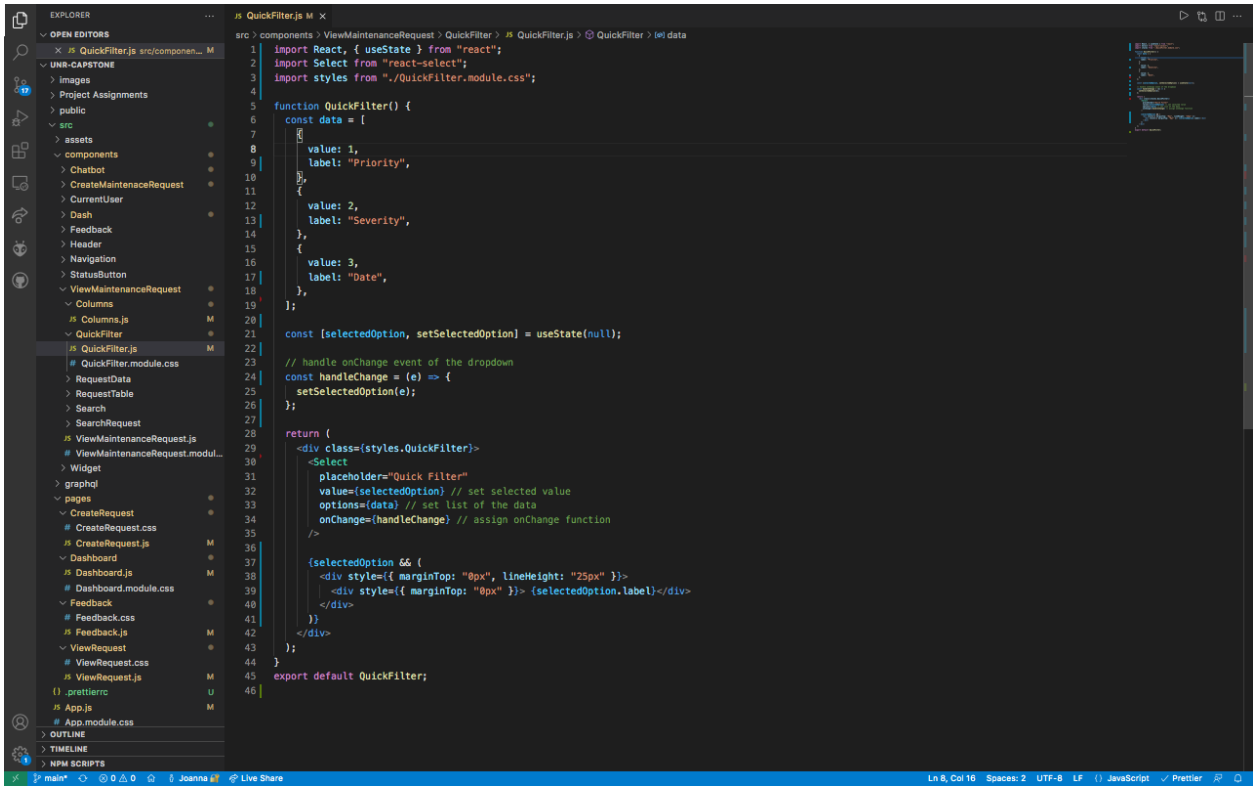


Fig. 7.4.2: A snapshot showing the QuickFilter.js file.

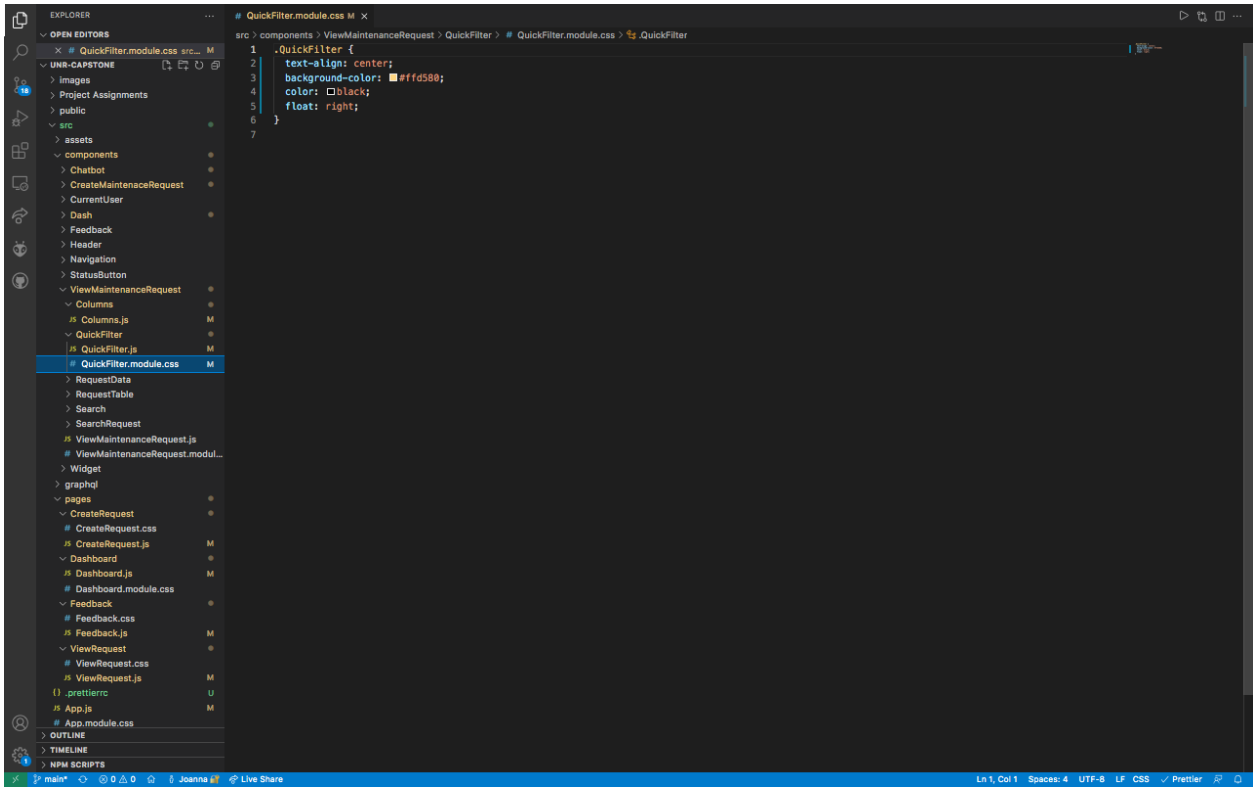


Fig. 7.4.3: A snapshot showing the QuickFilter.module.css file.

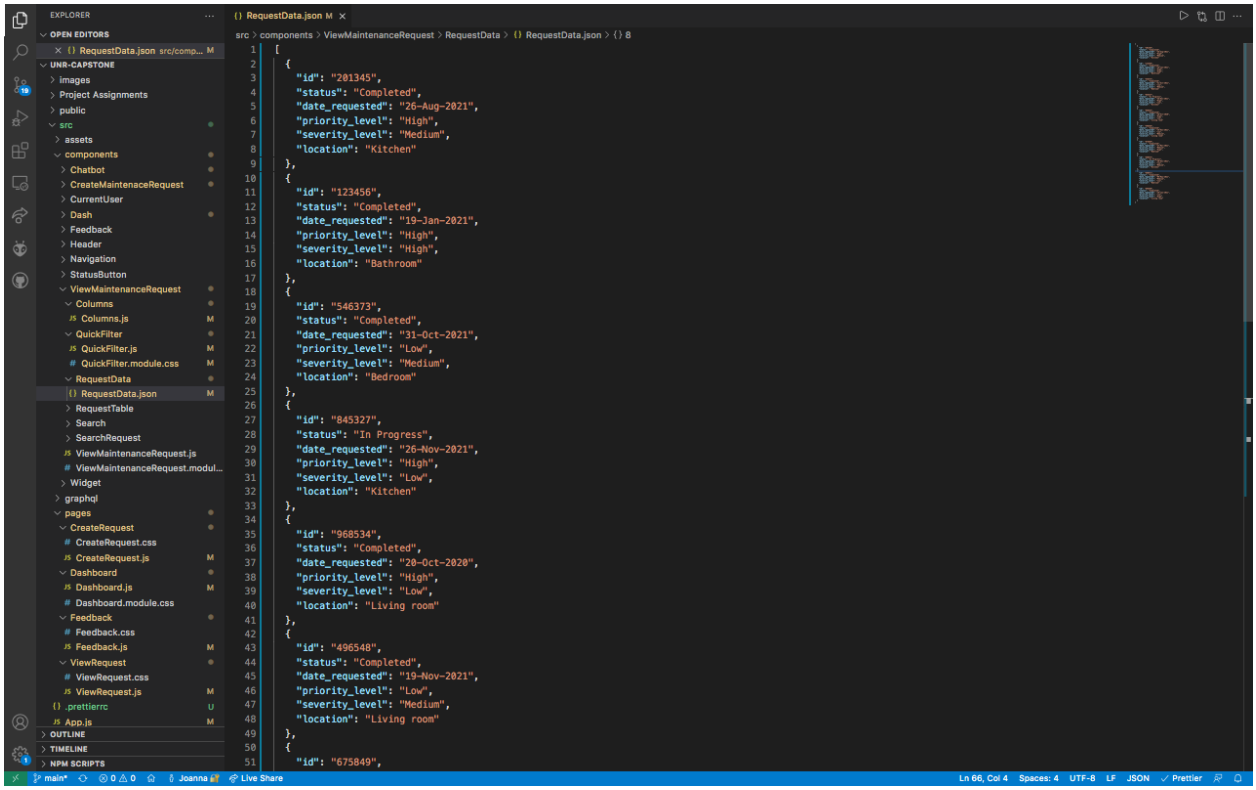


Fig. 7.4.4: A snapshot showing the RequestData.json file.

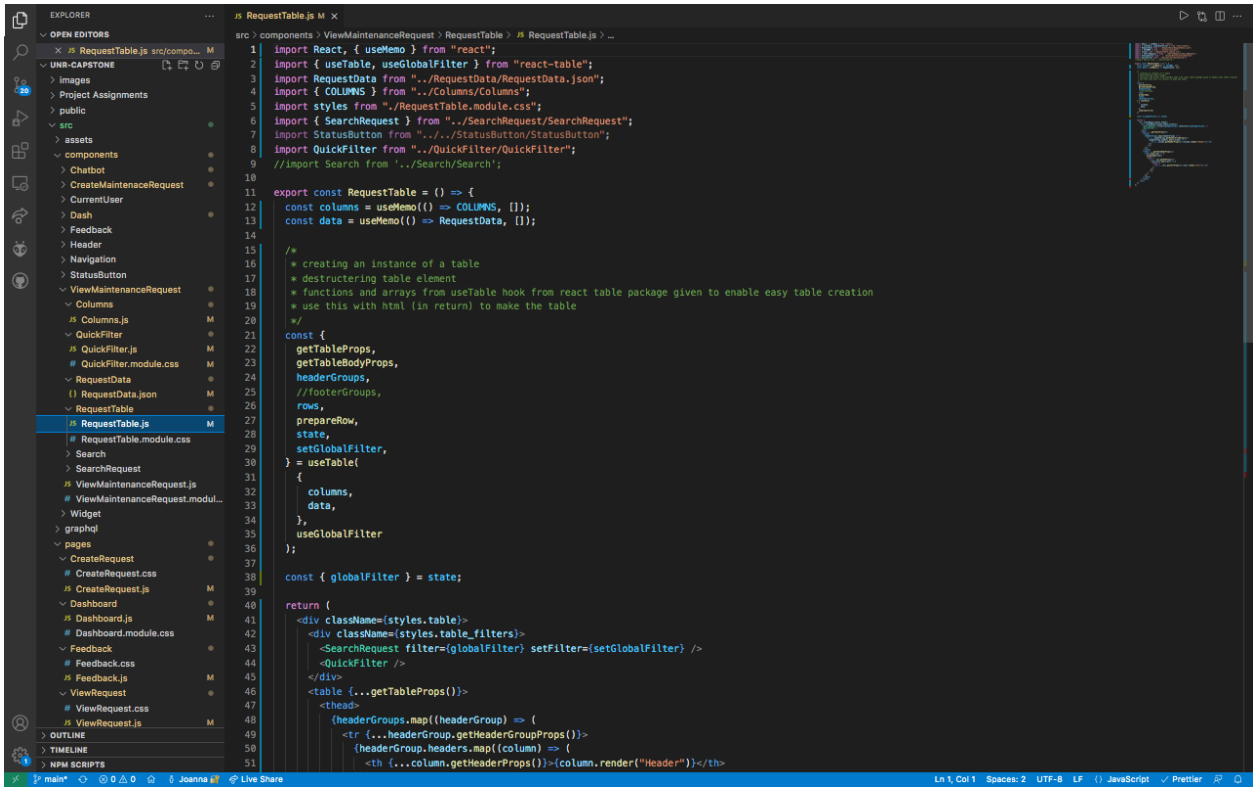


Fig. 7.4.5: A snapshot showing the RequestTable.js file.

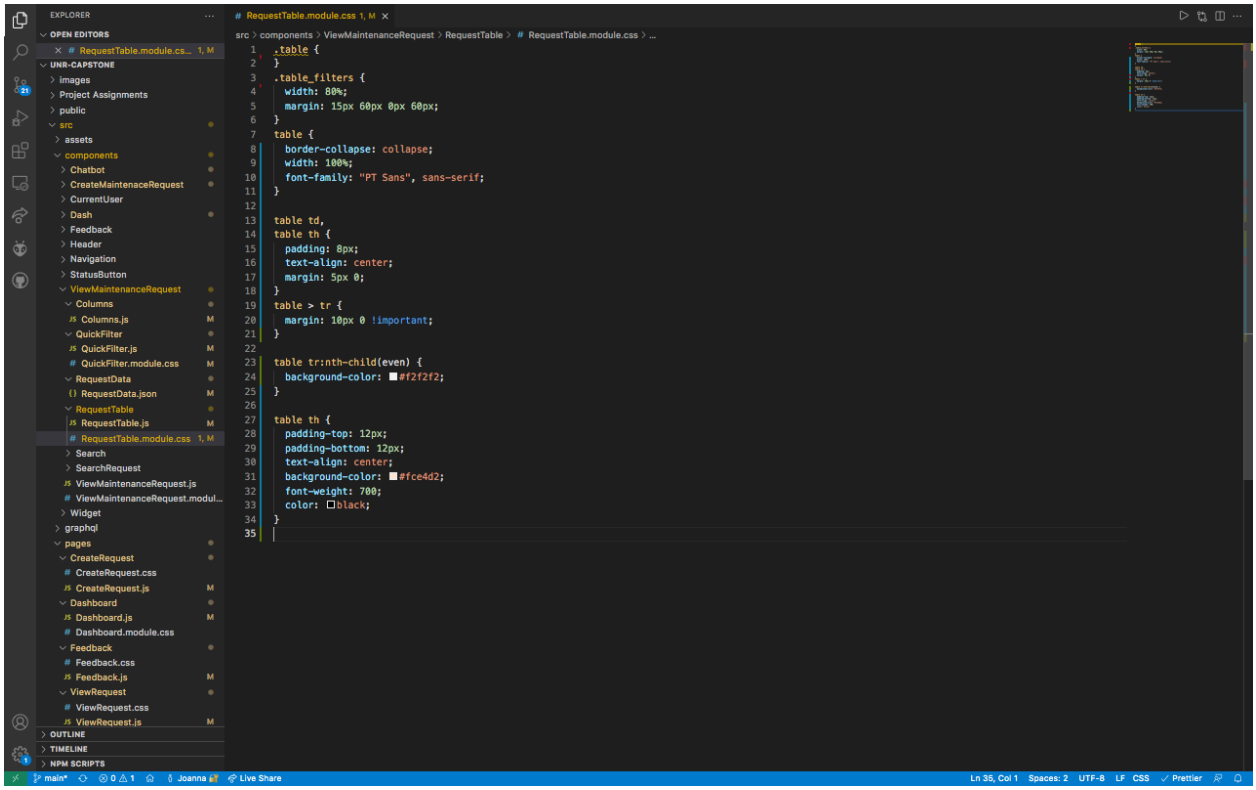


Fig. 7.4.6: A snapshot showing the RequestTable.module.css file.

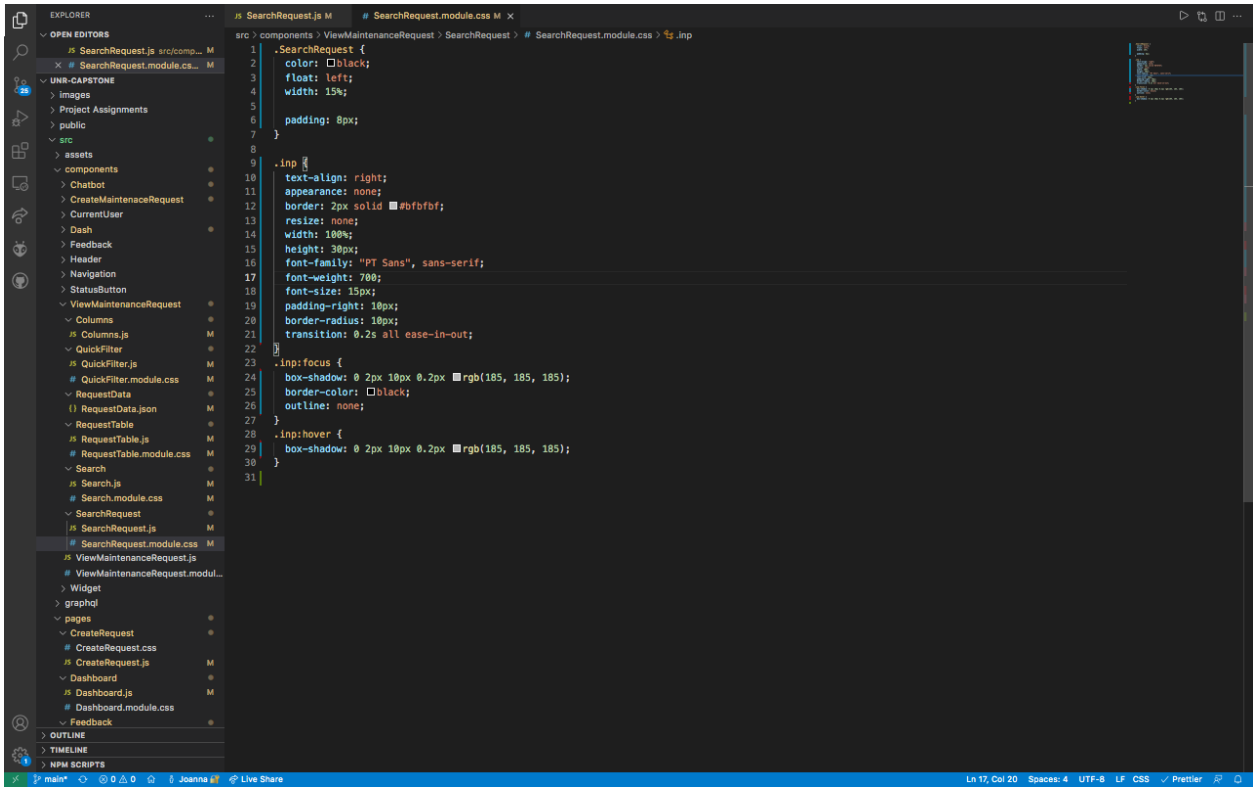


Fig. 7.4.7: A snapshot showing the SearchRequest.module.css file.

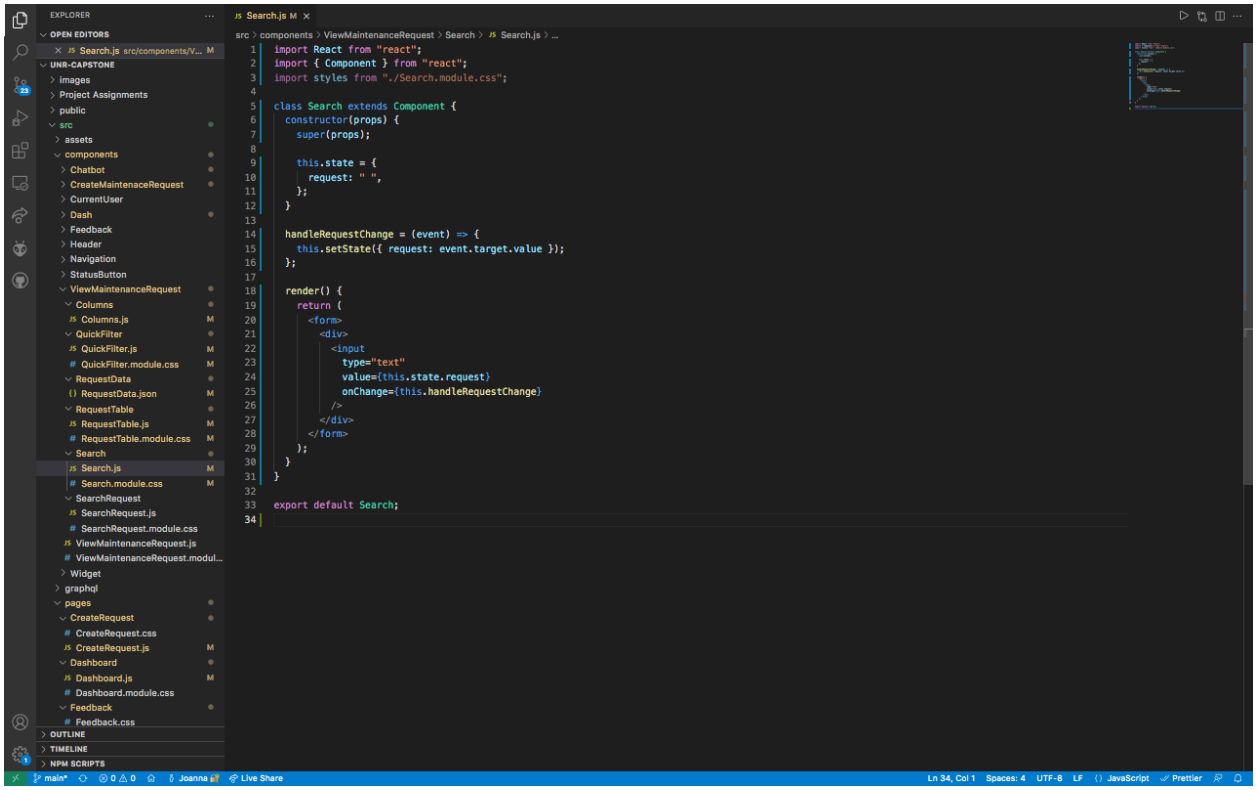


Fig. 7.4.8: A snapshot showing the Search.js file.

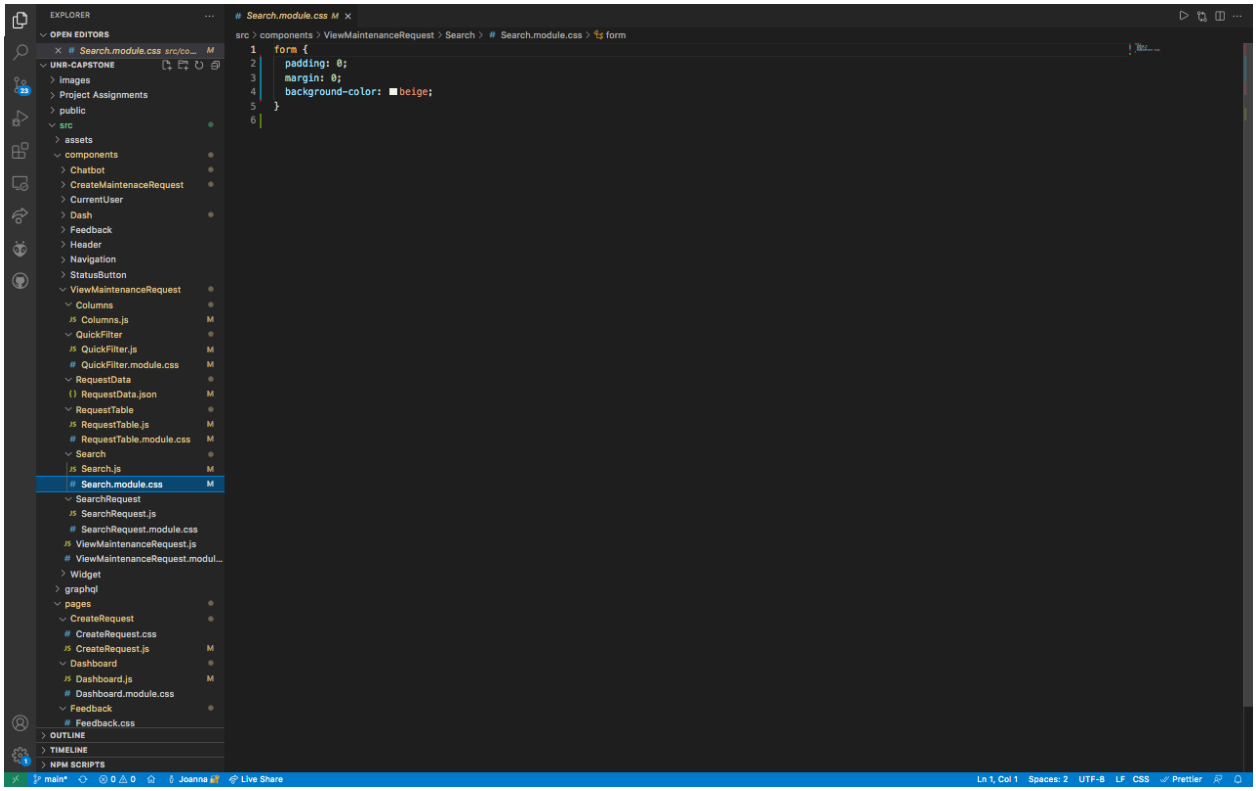


Fig. 7.4.9: A snapshot showing the Search.module.css file.

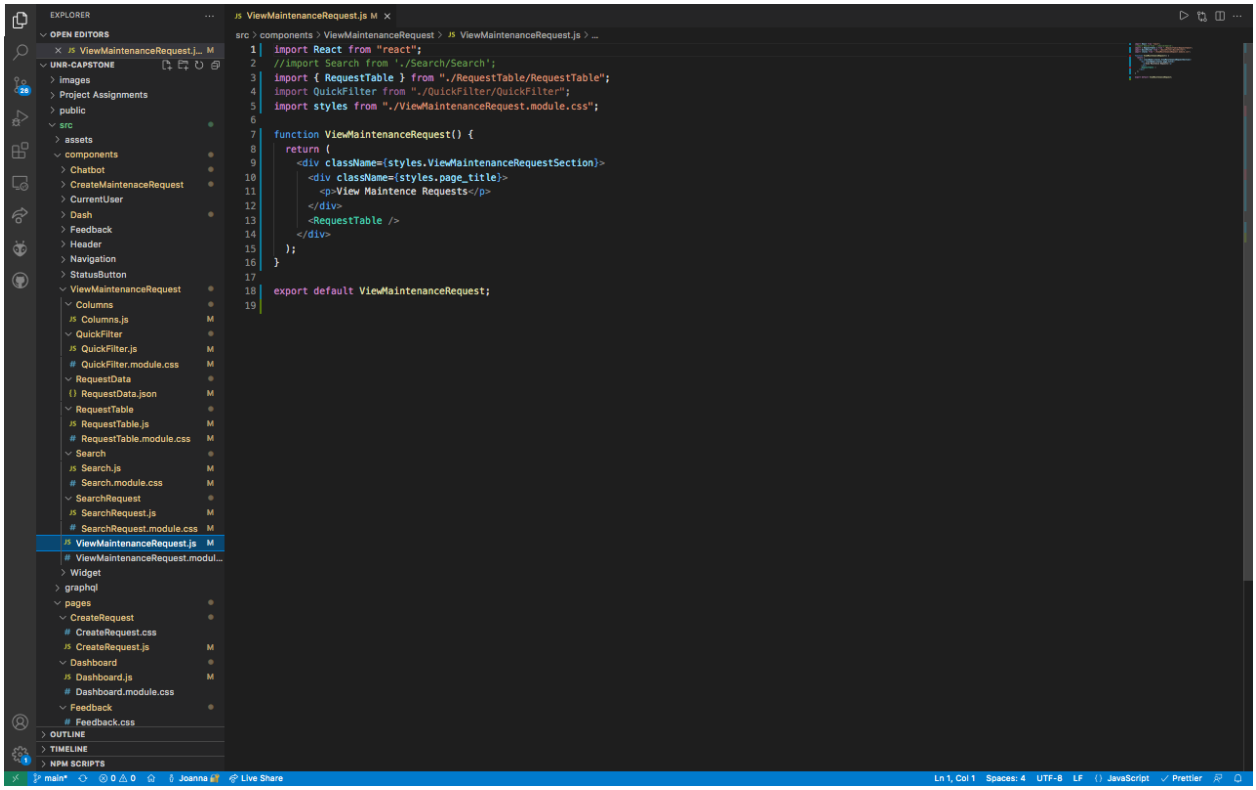


Fig. 7.4.10: A snapshot showing the ViewMaintenanceRequest.js file.

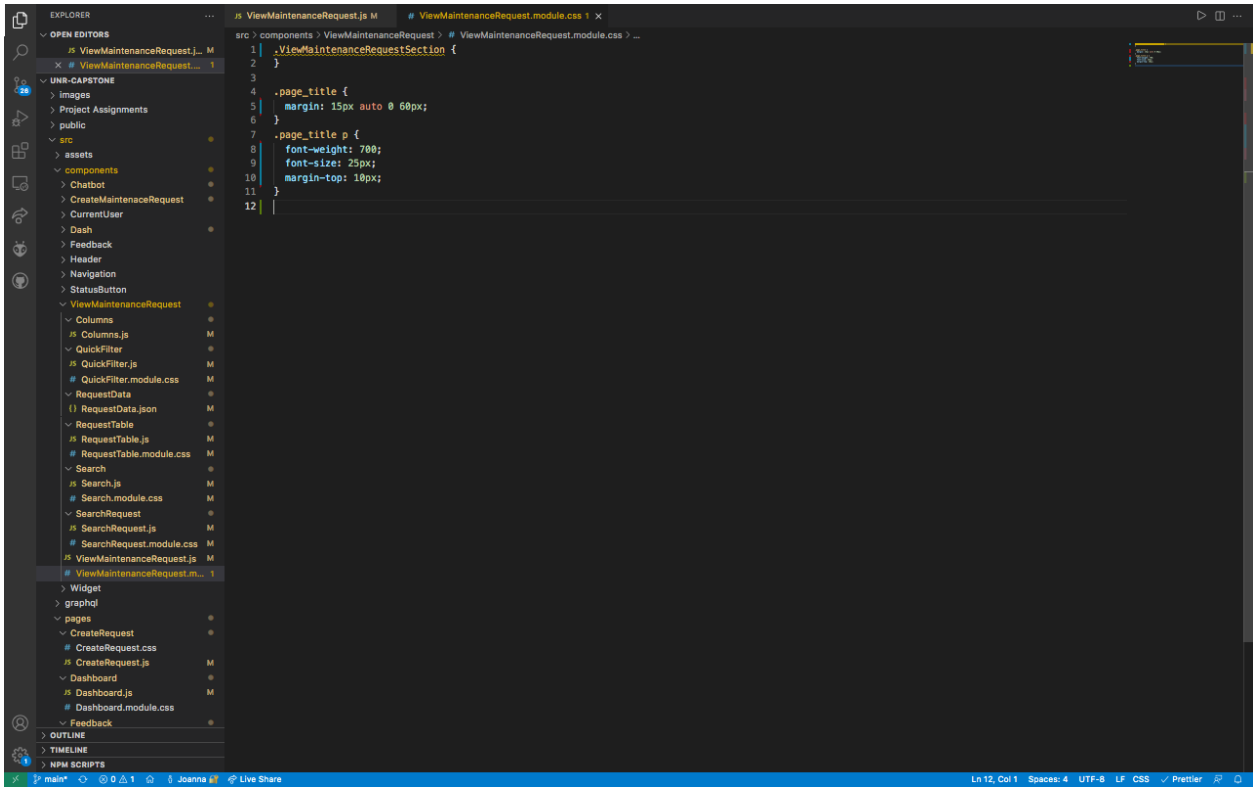


Fig. 7.4.11: A snapshot showing the ViewMaintenanceRequest.module.module.css file.

7.5. Aisha Co

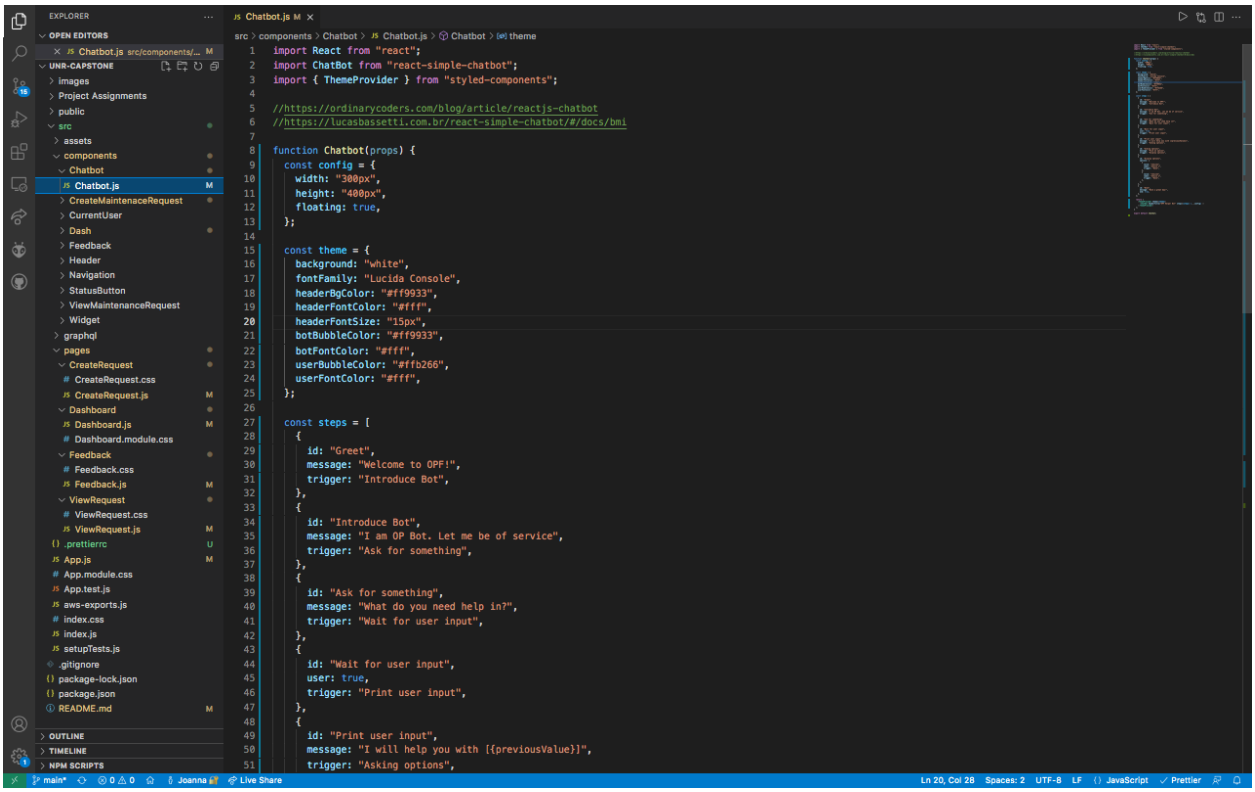


Fig. 7.5.1: A snapshot showing the Chatbot.js file which includes everything regarding the chatbot component.

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