Maintenance Manual

Project name: Lilac Ventures Website

Team name: Algo Avengers

David Enzler
Prasad Prabhu
Nathaniel Cruz
Justin Roome
Robert Mancuso
Ryan McCormick
Francisco Avalos
Sir Christian Orefer Casals

Date: 12/1/2023

Table of Contents

1.0 Introduction

Description of the application:

Lilac Ventures requires a software application to fulfill the business needs of Gail Tateyama's financial coaching service. The application, a user-friendly website, must be ADA-compliant and mobile-friendly, featuring an interactive customer account portal. This portal allows clients to submit forms, manage appointments through a calendar system, and communicate easily with the client. The website also incorporates a secure payment processing system, prioritizing the safety of financial data. Additionally, a dedicated company email address is set up, and a separate data management system captures website traffic analytics and generates reports for Gail Tateyama's exclusive access. The software aims to streamline financial coaching services, enhance customer engagement, and ensure the secure management of sensitive information.

2.0 Implementation tools and technologies

Programming languages Used, libraries, frameworks SDK's

- IDE Used
    - Visual Studio Code
- Programming Languages
    - Typescript, React, Express.js, Node.js
- Technologies, libraries and frameworks
    - Tailwind CSS
- Web Services
    - AWS Beanstalk and Amplify

Database

- MongoDB
    - Username: REDACTED
    - Password: REDACTED

Payment Services or other API's

- Stripe - Payment API
    - Username: REDACTED
    - Password: REDACTED
- AWS
    - Username: REDACTED
    - Password: REDACTED
- GoDaddy - Domain registration
    - Username: REDACTED
    - Password: REDACTED

Some Words on Transitioning from Javascript to Typescript:

The purpose of this section is to assist an experienced web developer who may be well-versed in Javascript, but might not have much (if any) experience with Typescript. We have compiled this brief guide to how the code on the site pages functions. Via an example page, we will explain maintaining and modifying the AdminTable component, a React component written in Typescript. It focuses on Typescript-specific features and best practices for managing customer data display and interactions.

Typescript vs Javascript

Type Annotations: Typescript extends Javascript by adding type annotations. For example, in this page the *Client* interface defines the shape of client objects. When working with Typescript, it is critical that you ensure variables and function parameters align with these defined types.

Interfaces: Unlike Javascript, Typescript allows declaring interfaces (e.g., *Client*, *AdminTableProps*). They describe object shapes, enforcing structure and providing IntelliSense in the Visual Studio Code IDE this project utilized.

Type Safety in Functions: Typescript checks static types at compile time - *this is very different from Javascript*. For example, *fetchClients* is typed to return Promise<Client[]>, ensuring the function resolves to an array of *Client*. While adding more work up front, this difference helps with error catching and debugging later in the development process.

Error Handling: Typescript's strict typing procedures help identify potential type-related errors during development, unlike Javascript's runtime errors, which may cause errors to go unnoticed until something breaks, or returns the wrong output, etc., which can make debugging more difficult or cause errors in deployment or afterwards.

Maintenance Tasks

Modifying Table Rows:

To change displayed client attributes, update the *Client* interface and modify the table's <thead> and <tbody> sections in the JSX/TSX.
Whenever this is done, make sure to implement these changes in the *fetchClients* function; to ensure the backend provides necessary data.

```
11
12  interface Client {
13    id: string;
14    firstName: string;
15    lastName: string;
16    city: string,
17    street: string,
18    state: string,
19    zip: string,
20    phone: string;
21    email: string;
22    marital: string;
23    employment: string;
24    progress: string;
25    accountLink: string;
26  }
27
```

```
112       <table className="clients-table">
113         <thead>
114           <tr>
115             <th>First Name</th>
116             <th>Last Name</th>
117             <th>Address</th>
118             <th>Mobile Number</th>
119             <th>Email</th>
120             <th>Marital Status</th>
121             <th>Employment Status</th>
122             <th>Progress Step</th>
123           </tr>
124         </thead>
125         <tbody>
126           {clients.map(client => (
127             <tr key={client.id}>
128               <td>{client.firstName}</td>
129               <td>{client.lastName}</td>
130               <td>{formatAddress(client.street, client.city, client.state, client.zip)}</td>
131               <td>{client.phone}</td>
132               <td>{client.email}</td>
133               <td>{client.marital}</td>
134               <td>{client.employment}</td>
135               <td>
136                 {client.progress}
137                 <button onClick={() => handleIncrease(client.email, client.progress)} className="green-button-admi
138                 <button onClick={() => handleDecrease(client.email, client.progress)} className="red-button-adminT
139               </td>
140             </tr>
141           ))}
142         </tbody>
```

Updating Row Contents:

Adjust the *Client* interface to add or remove properties.
In the JSX/TSX, map these properties to table cells.

Increment/Decrement Buttons:

In our code, *handleIncrease* and *handleDecrease* control these actions. Modify them, if necessary, to change any logic needed, ensuring continued compliance with the Typescript types used in the project.
For UI changes, update the corresponding JSX/TSX in the map function within <tbody>.

```
68  const handleIncrease = async (email: string, progress: string) => {
69    //console.log(`Increased for email: ${email}`);
70    try {
71      const id = await getClientIDByEmail(email);
72      //console.log(id);
73
74      const currentProgress = progress;
75      await axios.put(`${baseURL}/customerProgress/${id}`, { "progress": currentProgress + 1 });
76      setRefreshFlag(prevFlag => !prevFlag);
77    } catch (error) {
78      // Handle errors here
79      console.error('Error in handleIncrease:', error);
80    }
81  };
82
```

Backend Database Integration:

Modify *fetchClients*, *getClientIDByEmail*, *handleIncrease*, and *handleDecrease* for any backend changes. These functions interact with other pages/files on the site - be aware you might need to refer to those for making certain changes to the code on this page.

```
45    const fetchClients = async (): Promise<Client[]> => {
46      try {
47        const response = await axios.get(`${baseURL}:8080/clientInfoUpdate/`);
48        return response.data;
49      } catch (error) {
50        console.error('Error fetching clients:', error);
51        throw new Error('Failed to fetch clients');
52      }
53    };
54
55    async function getClientIDByEmail(email: any) {
56      try {
57        const response = await axios.get(`${baseURL}/customerProgress/getID/${email}`);
58        const id = response.data.id; // Assuming the response contains an "id" property
59        //console.log(id)
60        return id;
61      } catch (error) {
62        // Handle errors here
63        console.error('Error fetching client ID:', error);
64        //throw error; // Re-throw the error to be handled by the calling function
65      }
66    }
```

Ensure the data types in these functions align with the backend responses and the Client interface.

Error Handling and Debugging:

Typescript's static typing analysis helps catch type-related issues early, earlier than one might be used to in Javascript. Use this alongside traditional debugging methods - one common method we used often were ‘try-catch’ statements.

```
try {
  // Enter your function or other code here
} catch (error) {
  if (error instanceof Error) {
    // log statement of the mistake, helpful for debugging:
    console.error('Error:', error.message);
  }
}
```

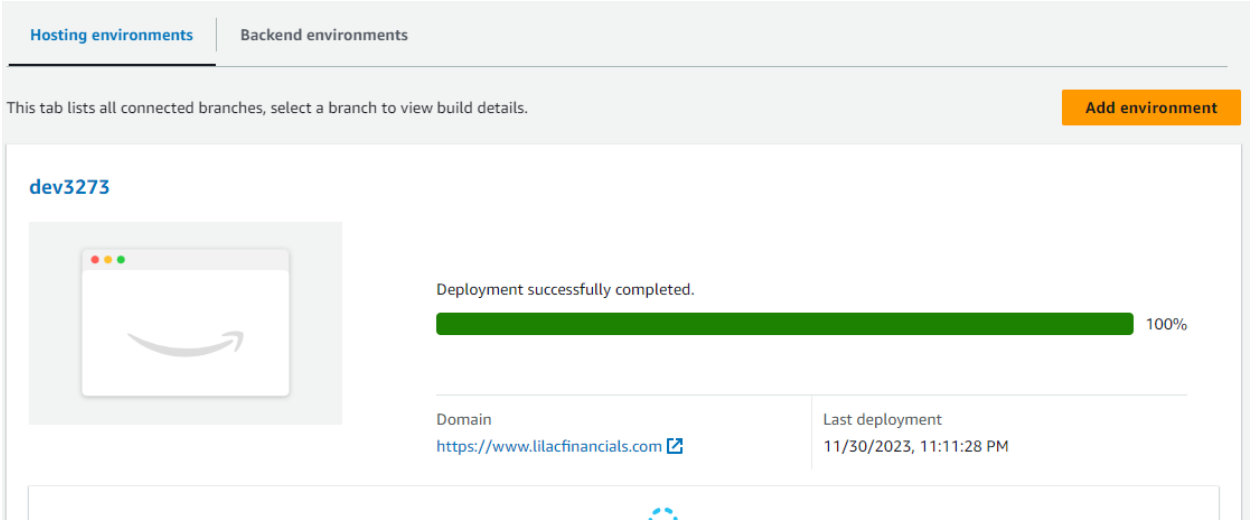
3.0 Runbook

List what to do if services goes down

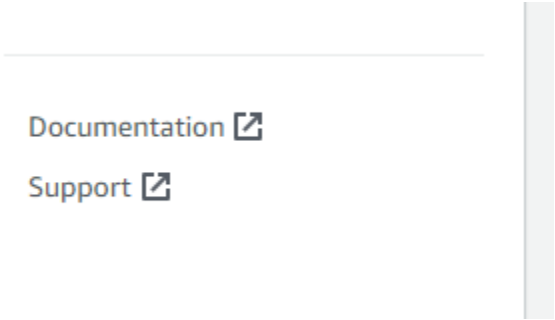
Note: More information on accessing the consoles for both AWS Amplify and AWS Beanstalk can be found in the 4.0 Deployment section.

Amplify

1. If AWS Amplify goes down, then you can check the portal for Amplify to check the health of the instance.

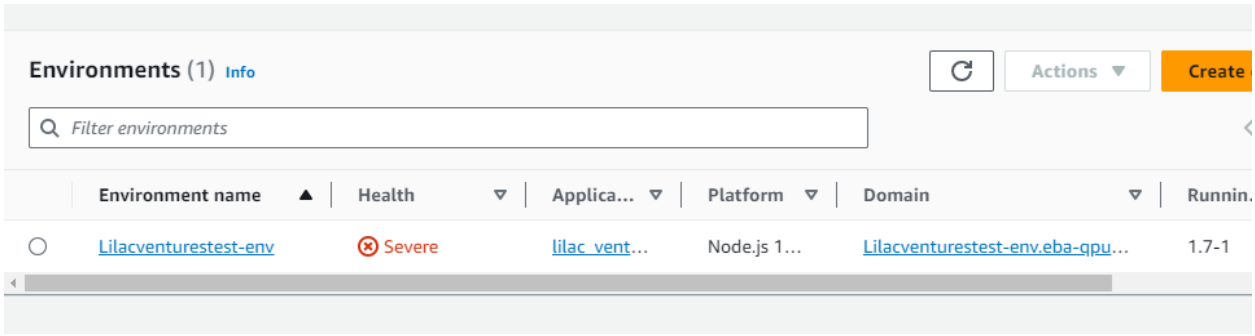


2. However, if there are issues with AWS, then you should reach out using the support link in the left side of the AWS Amplify console

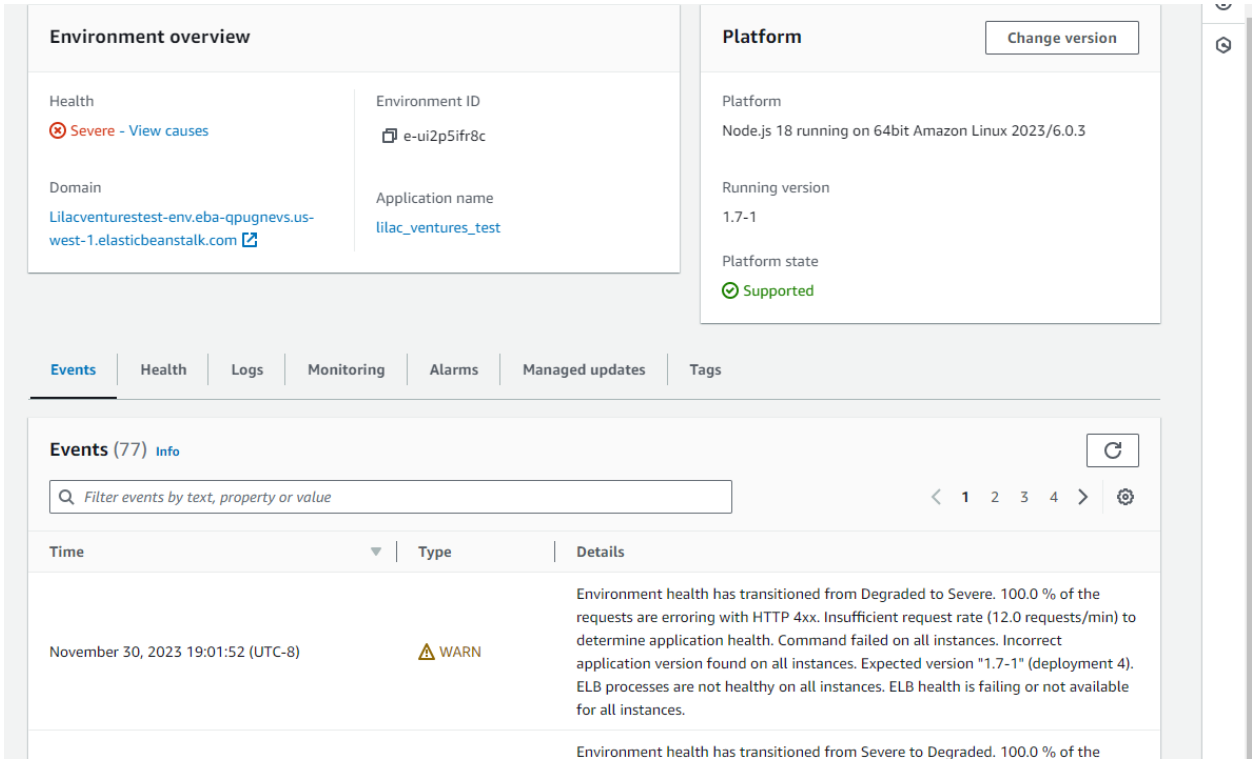


AWS Beanstalk

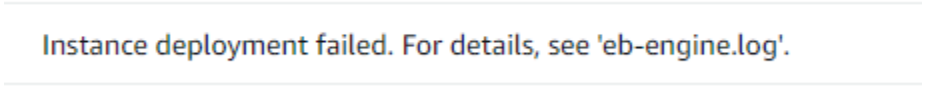
1. AWS Beanstalk deploys 2 machines: a load balancer and an EC2 instance. The status of both can be seen from the environment tab



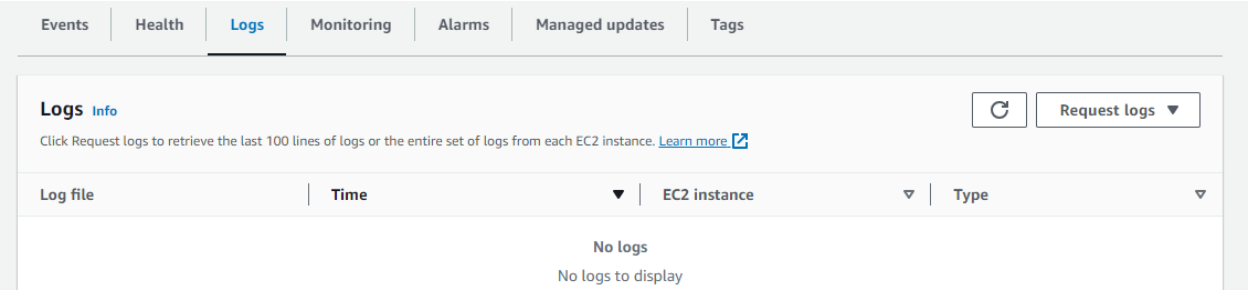
2. The above instance is listed as `Severe`
3. Click on the environment name to get more information



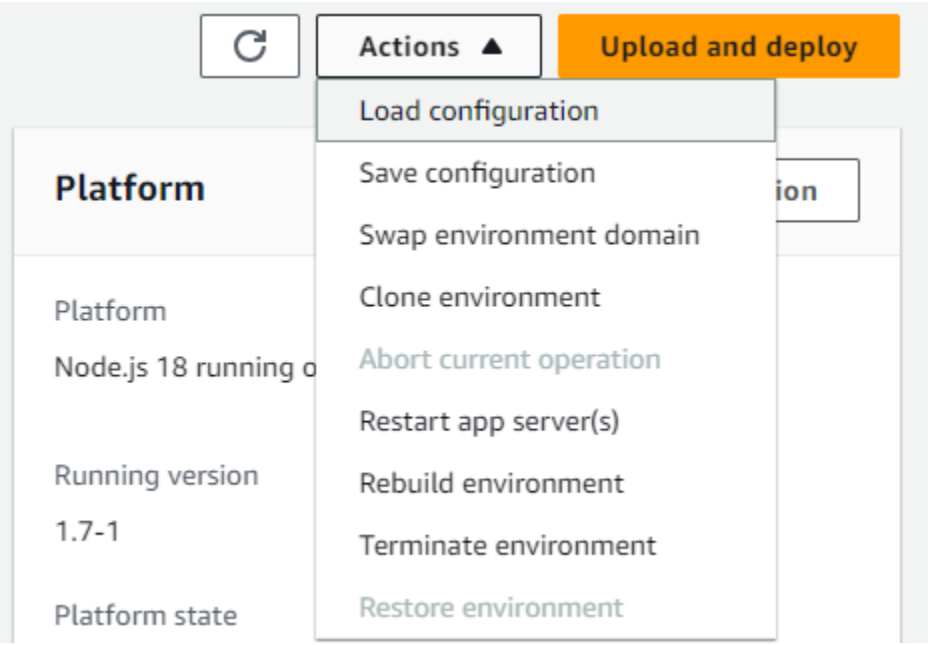
4. Information can be seen in the Events tab. Additionally, security events are found in the Monitoring and Alarms tabs.
    a. In this example, an application failed to deploy.



5. Download and check logs from the Logs tabe



6. You can request either the full logs or just the last 100 lines. I recommend starting with the last 100 lines, and then getting the full logs if not enough information is provided.
7. The Events tab will usually provide a recommended log to check depending on what service failed.
8. The environment can be restarted by selecting `Actions` from the environment overview and clicking `Restart app server`



Database becomes corrupt

Currently, a backup is not set up for the database. However, MongoDB Cloud offers snapshot features that can be configured on their clusters.
More information on conjuring regular snapshots can be found on their site:
https://www.mongodb.com/docs/atlas/backup/cloud-backup/overview/

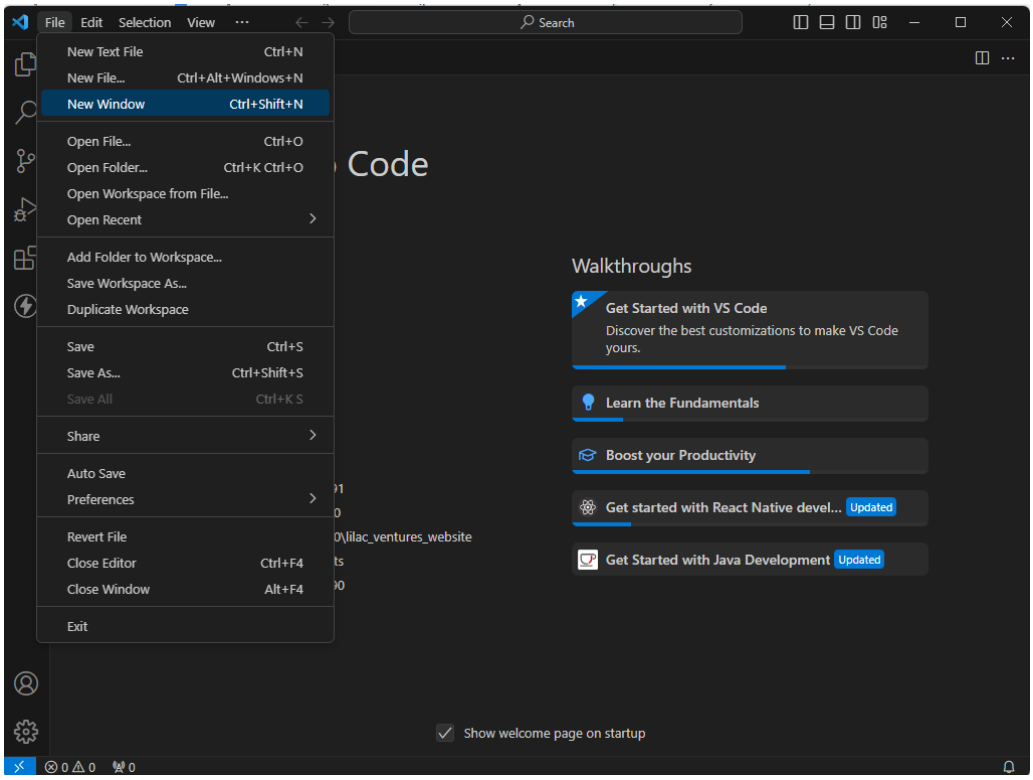
Third Party services go down (API's)

Stripe will list issues on their dashboard. Generally, if there are issues with any of the Stripe payment APIs, the responses to the endpoints will return 5xx statuses.

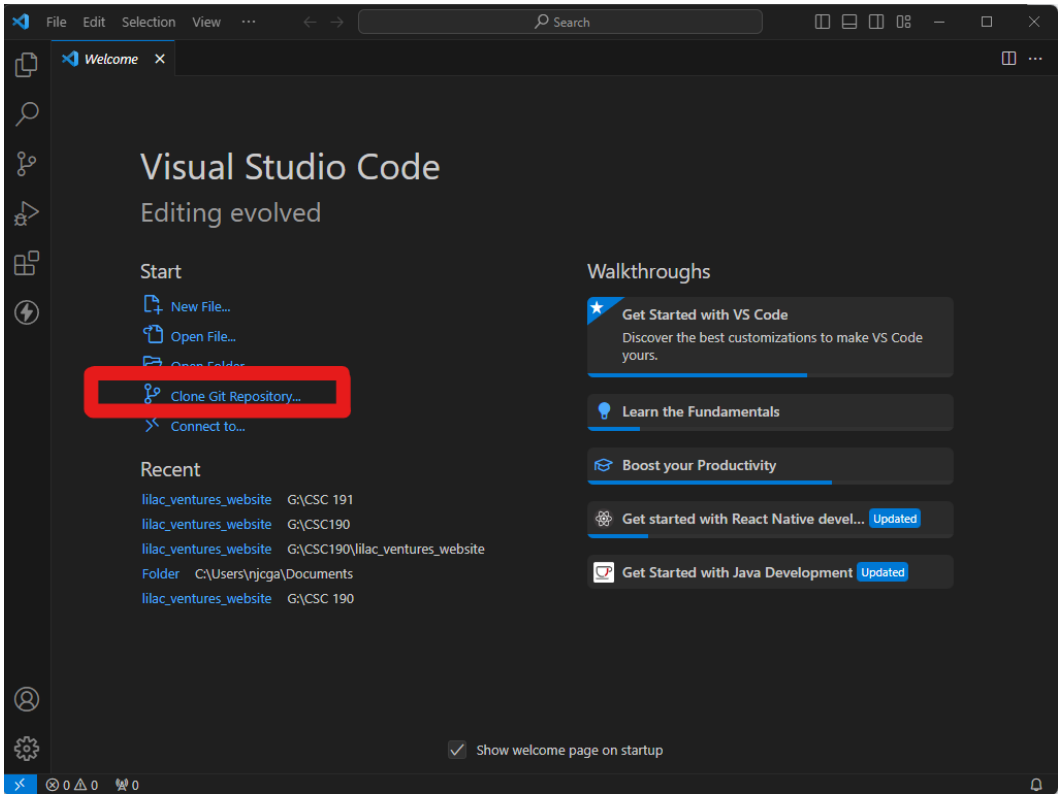
All endpoints using the Stripe API are under the `./customerBilling` endpoints. The routes file for the services is `./routes/customerBilling.ts`.
If there are issues with payments being processed or invoices not being created, then the first place to check is the Stripe Dashboard. They will list any service outages there. If it is not clear from the dashboard where the issues are stemming from, then set listeners on the endpoints. Generally, anything returning a 5xx status code indicates an issue on Stripe's servers. Generally, a status code 4xx indicates a configuration issue. Make sure to check the environmental variables of the NodeJS Elastic Beanstalk server to ensure all tokens are up to date.

How to configure a computer for development work

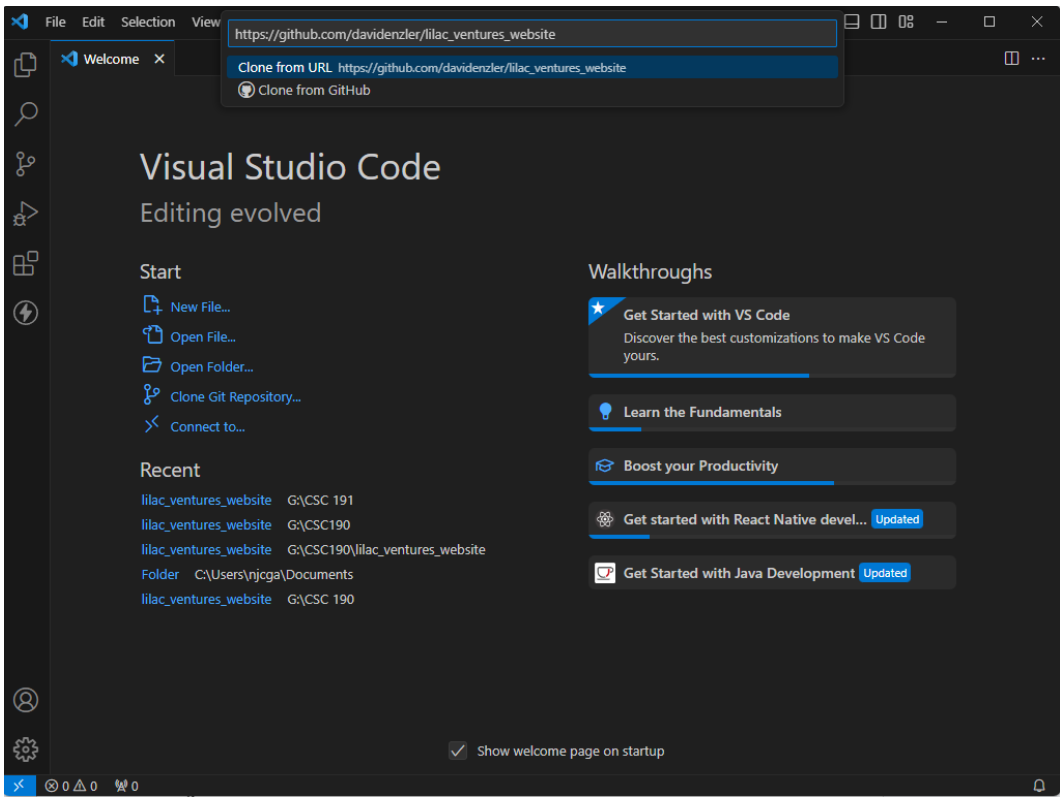
a. Install Visual Studio Code, Node.js, and Git.
b. In Visual Studio code, Click File in the top left and Open a new Window.



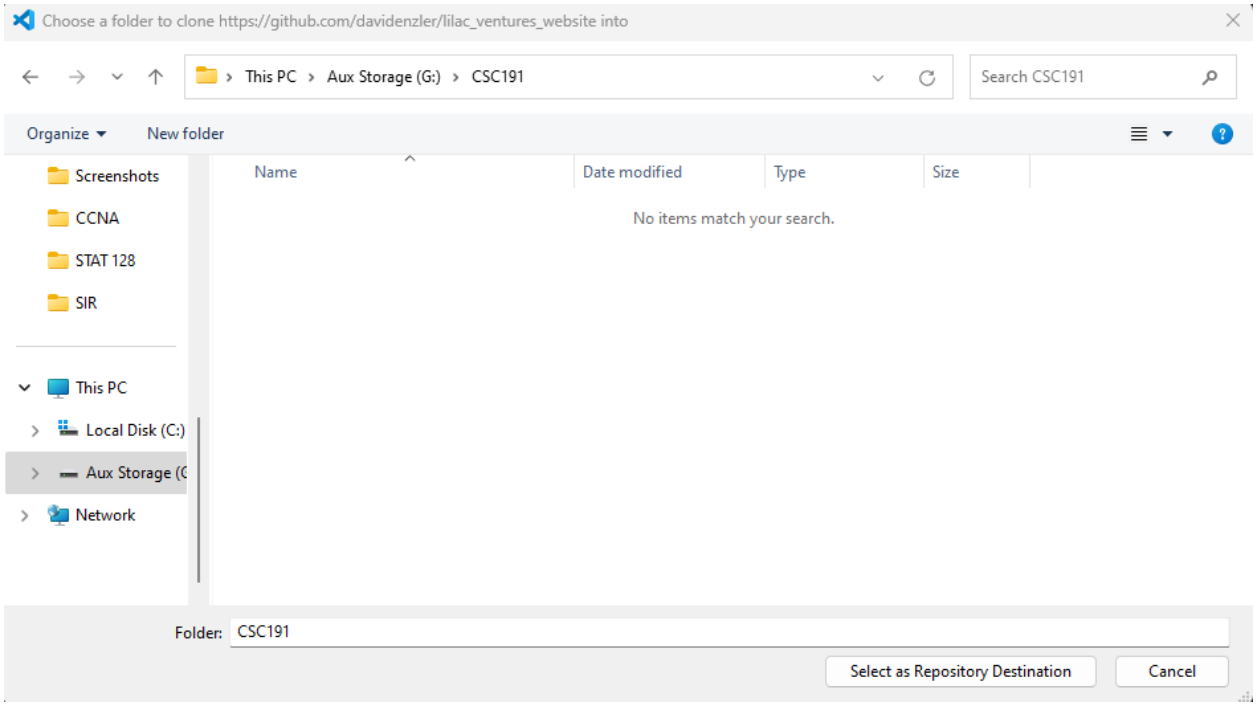
c. Select Clone Git Repository



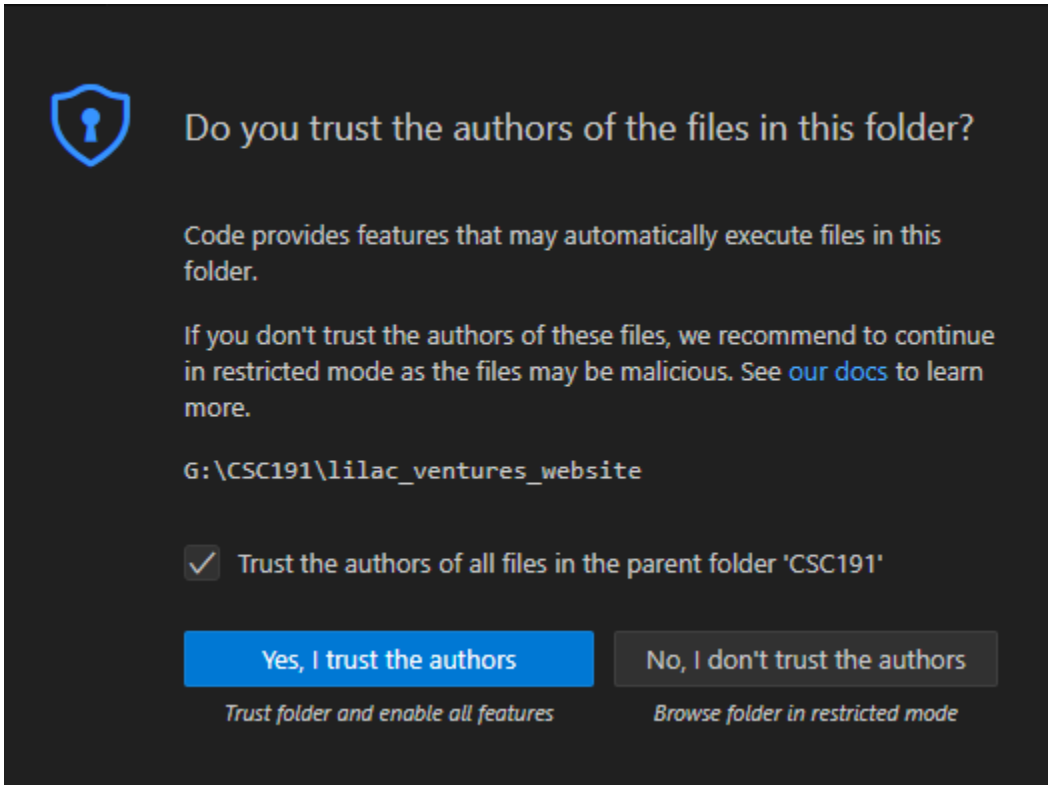
d. Enter the link for the for the github Lilac Ventures github repository



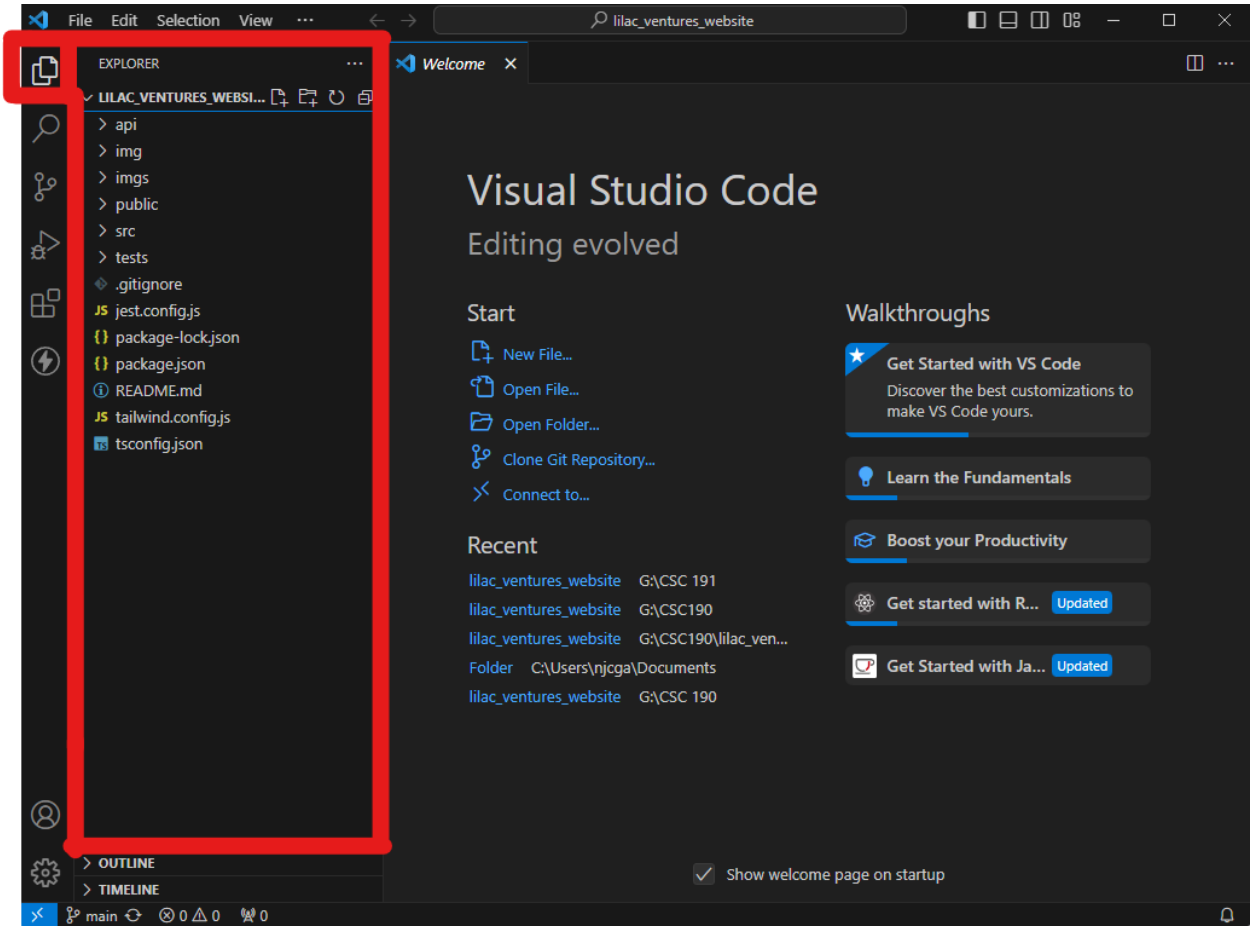
e. Select where to download the cloned repository



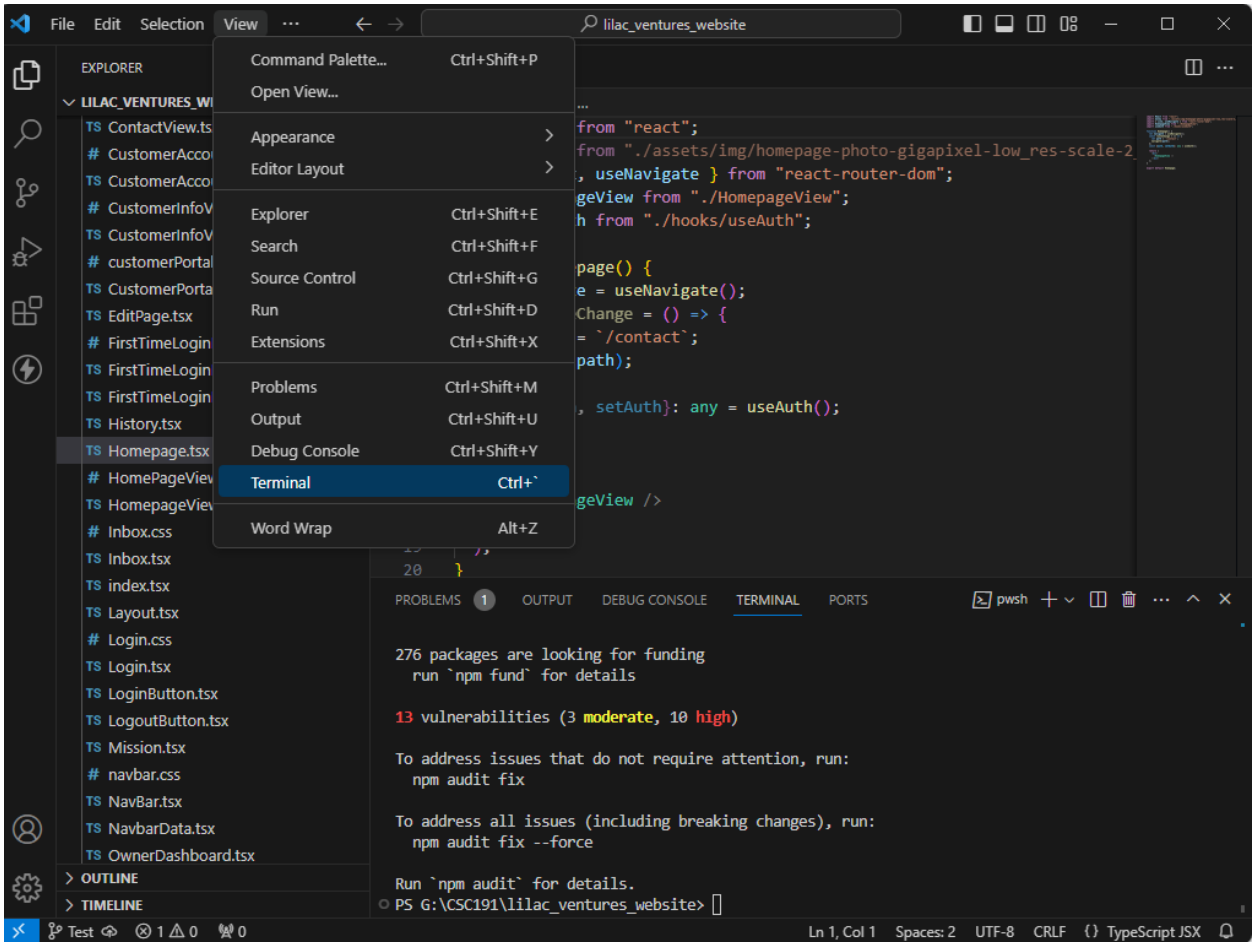
f. In the next popup, select Open to open the repository, then select Yes, I trust the authors



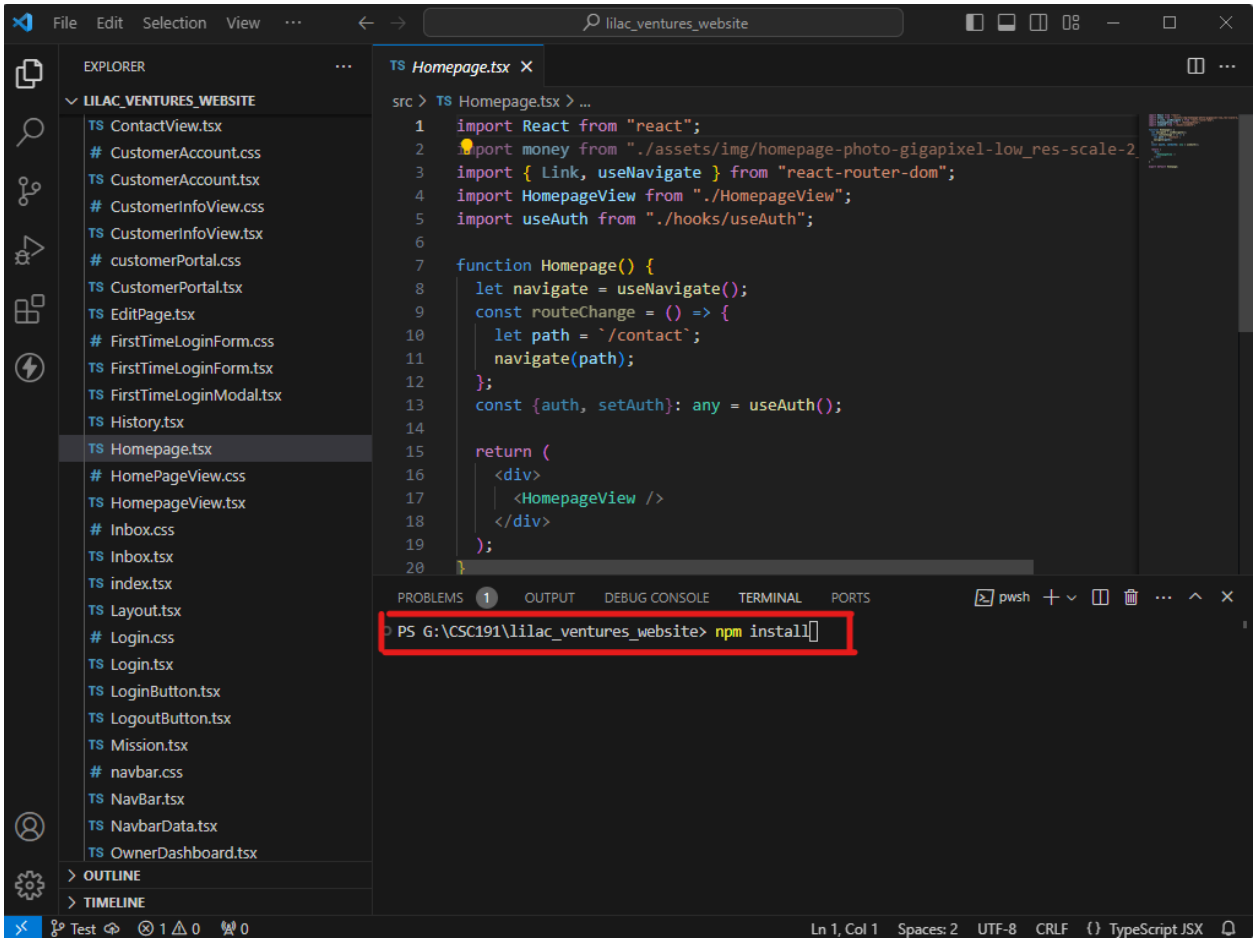
g. The files in the repository can now be opened and edited from the explorer tab



h. Press Ctrl+` or go to view->terminal to open a new terminal tab

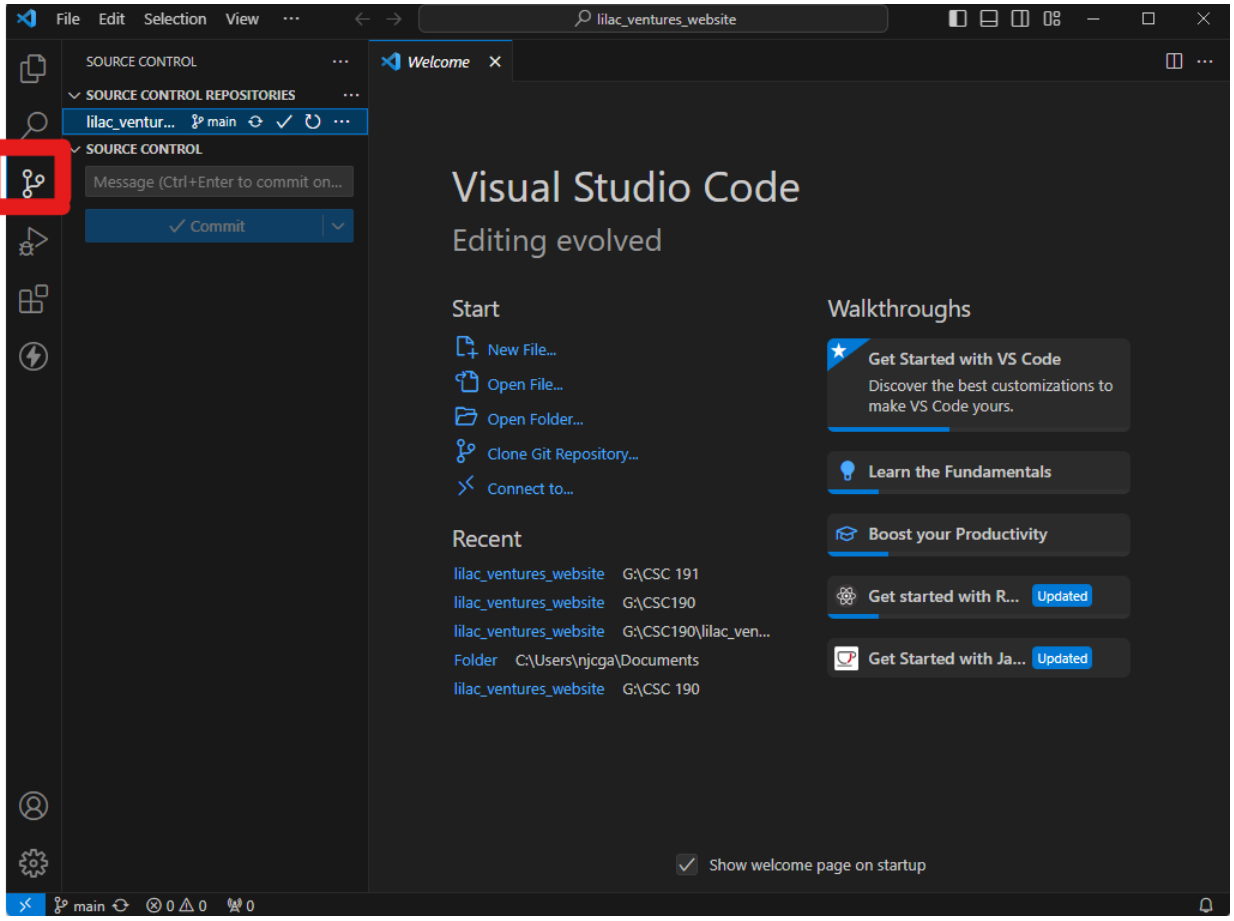


i. In the terminal that appears at the bottom of the screen type npm install and press enter to install any missing dependencies

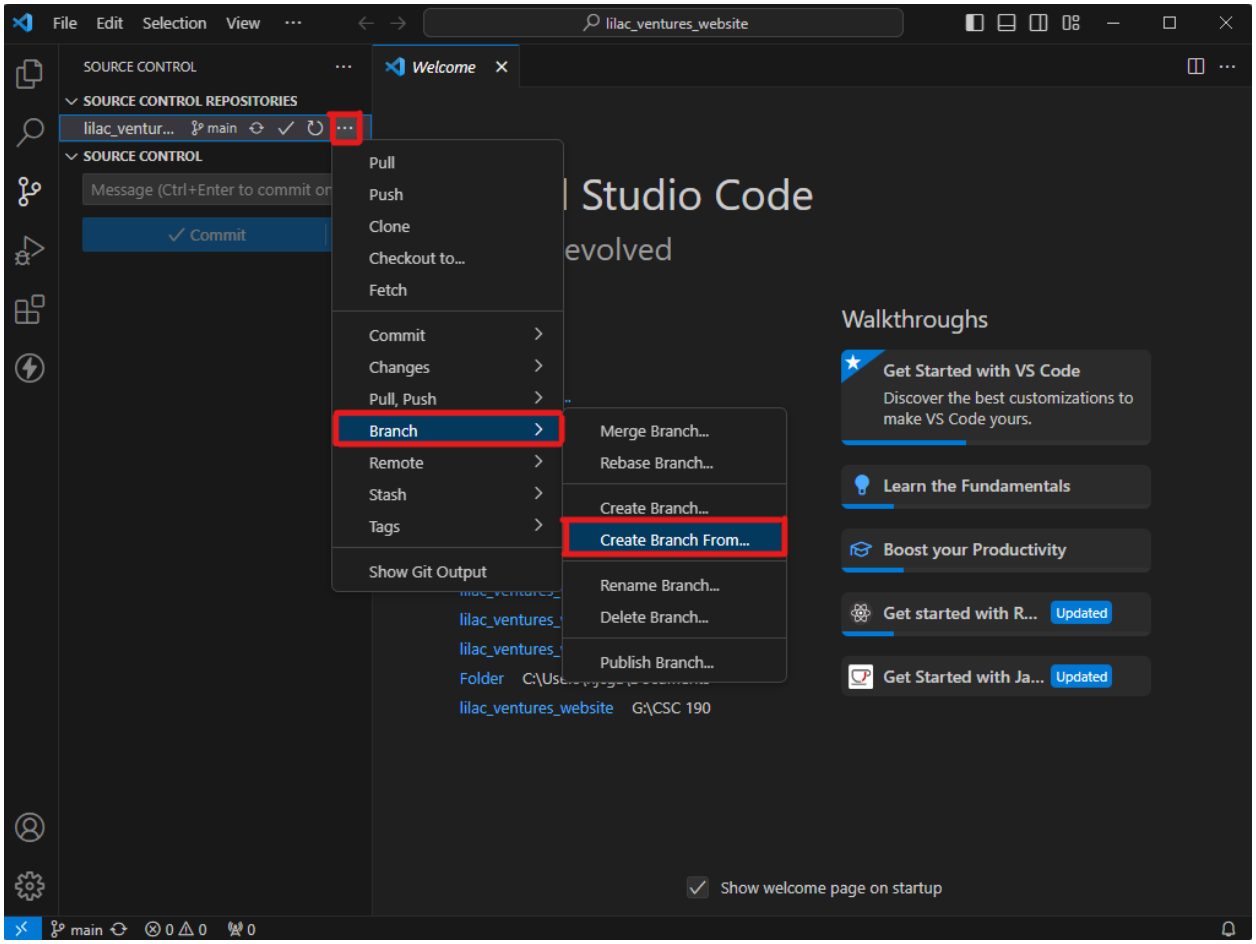


How to make a code change

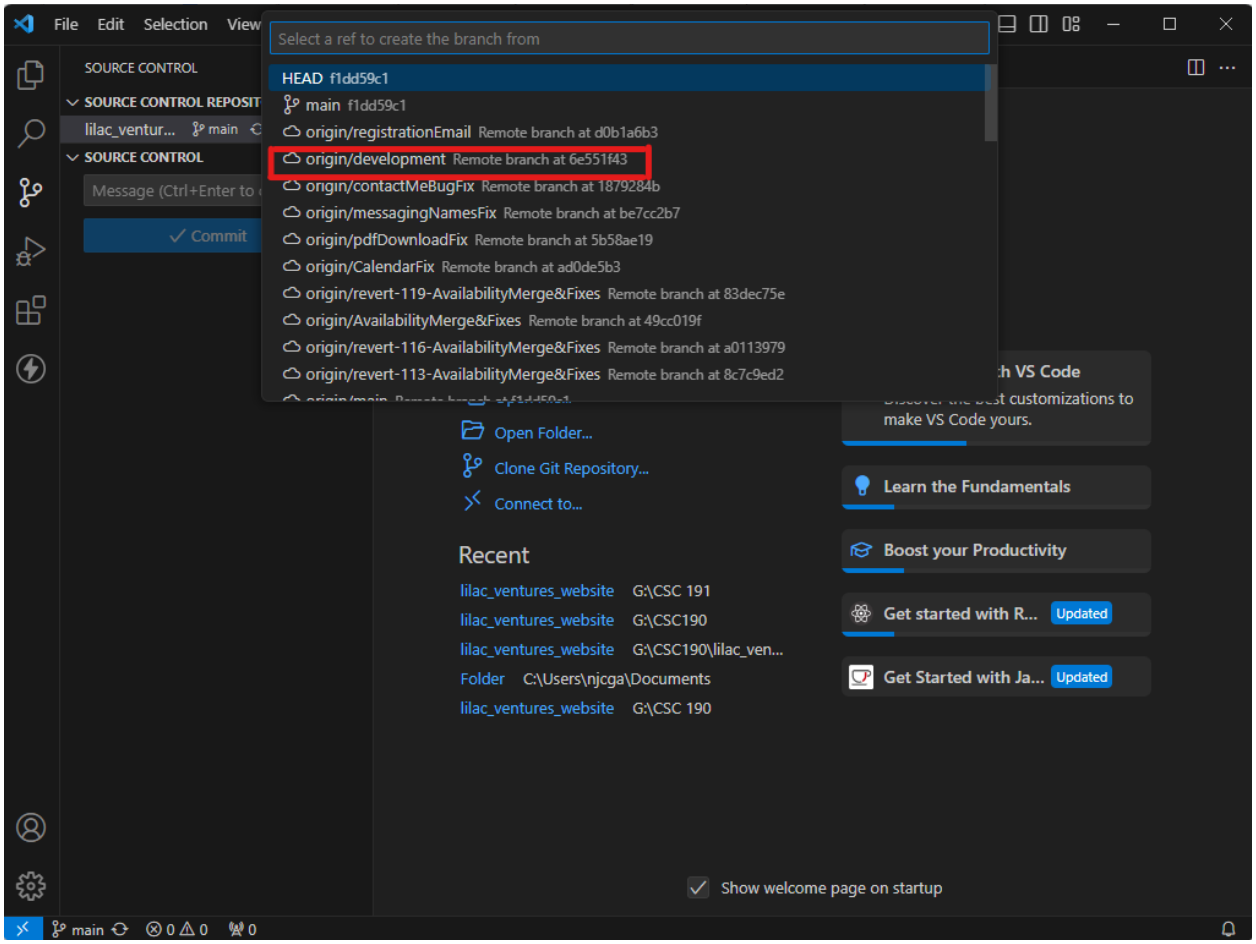
a. To make changes, first, you must create a new branch. This can be done from the source control tab.



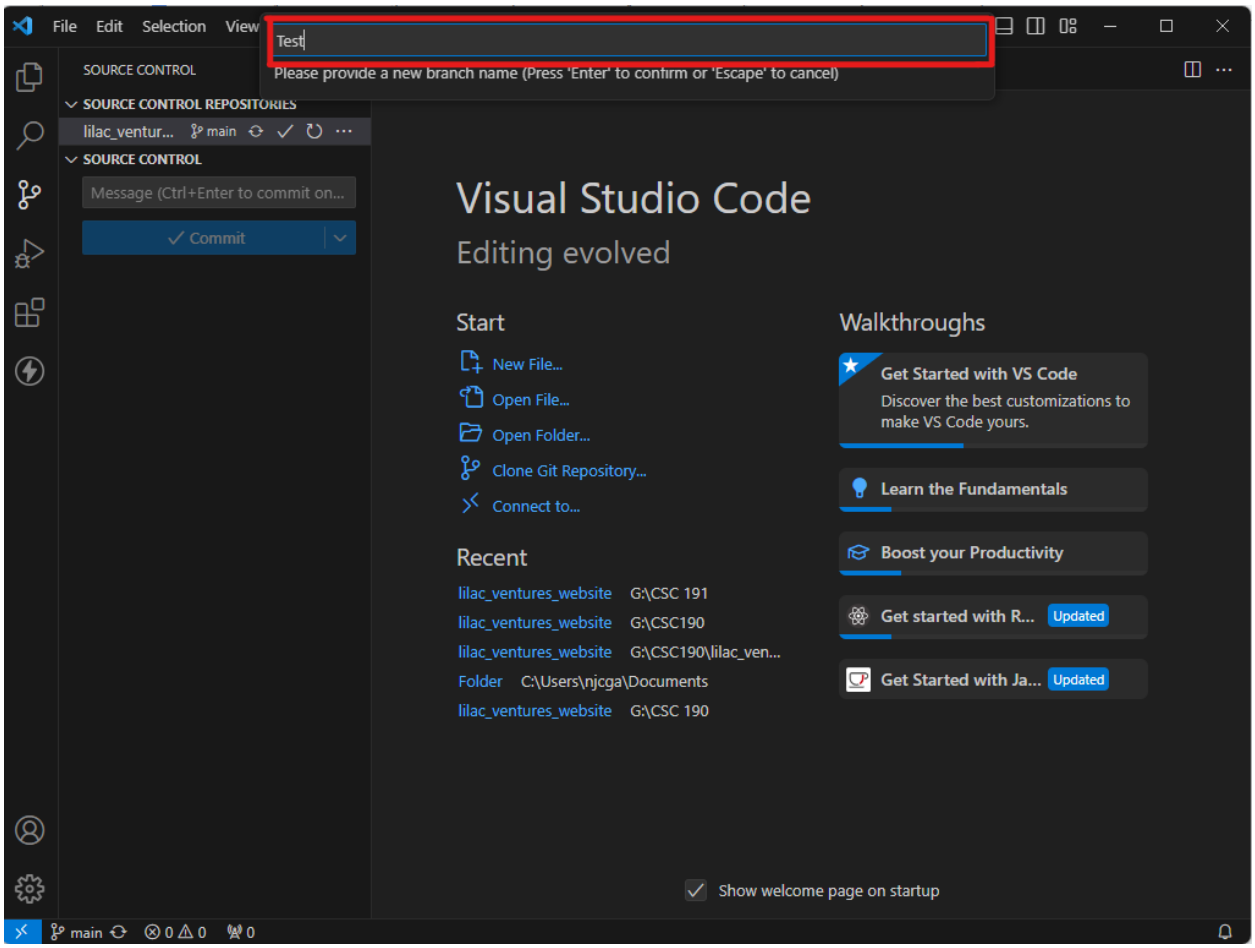
b. Click on the ellipsis on the line highlighted in blue, and from the drop-down menu, navigate to branch then select create branch from



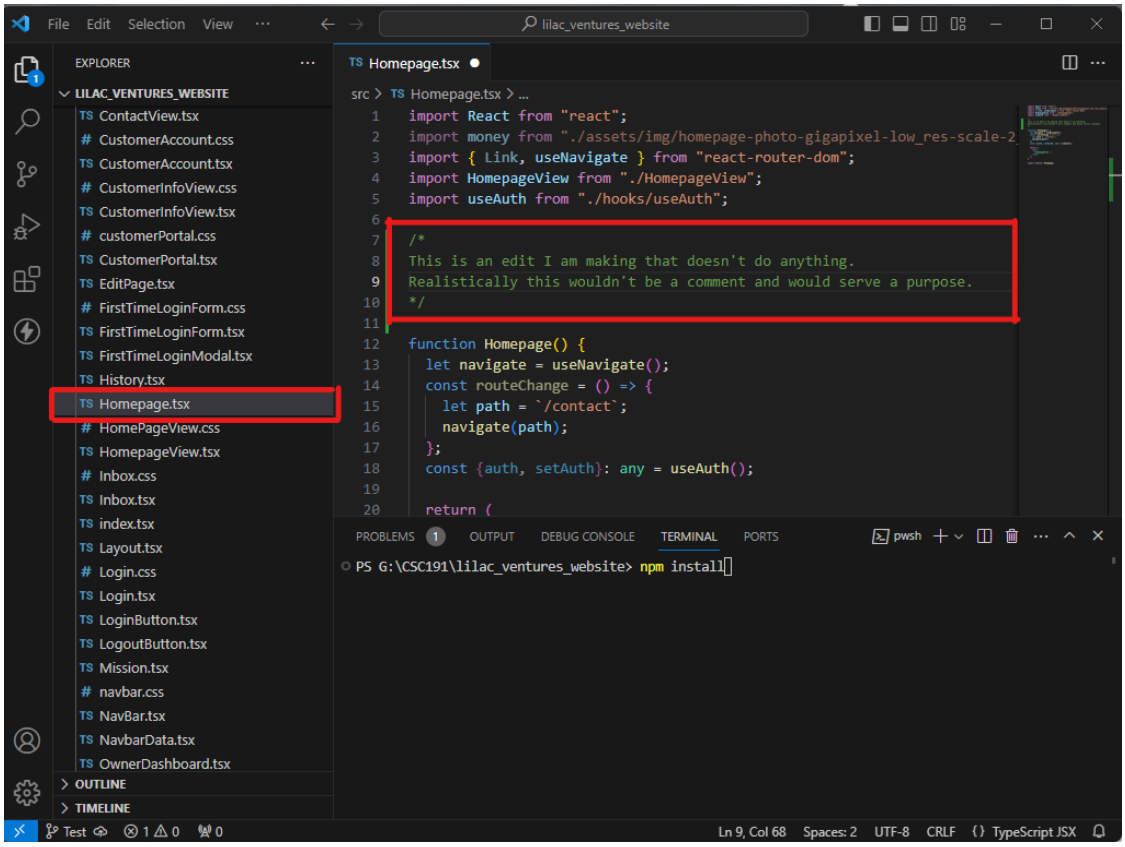
c. In the popup menu, select the origin/development branch

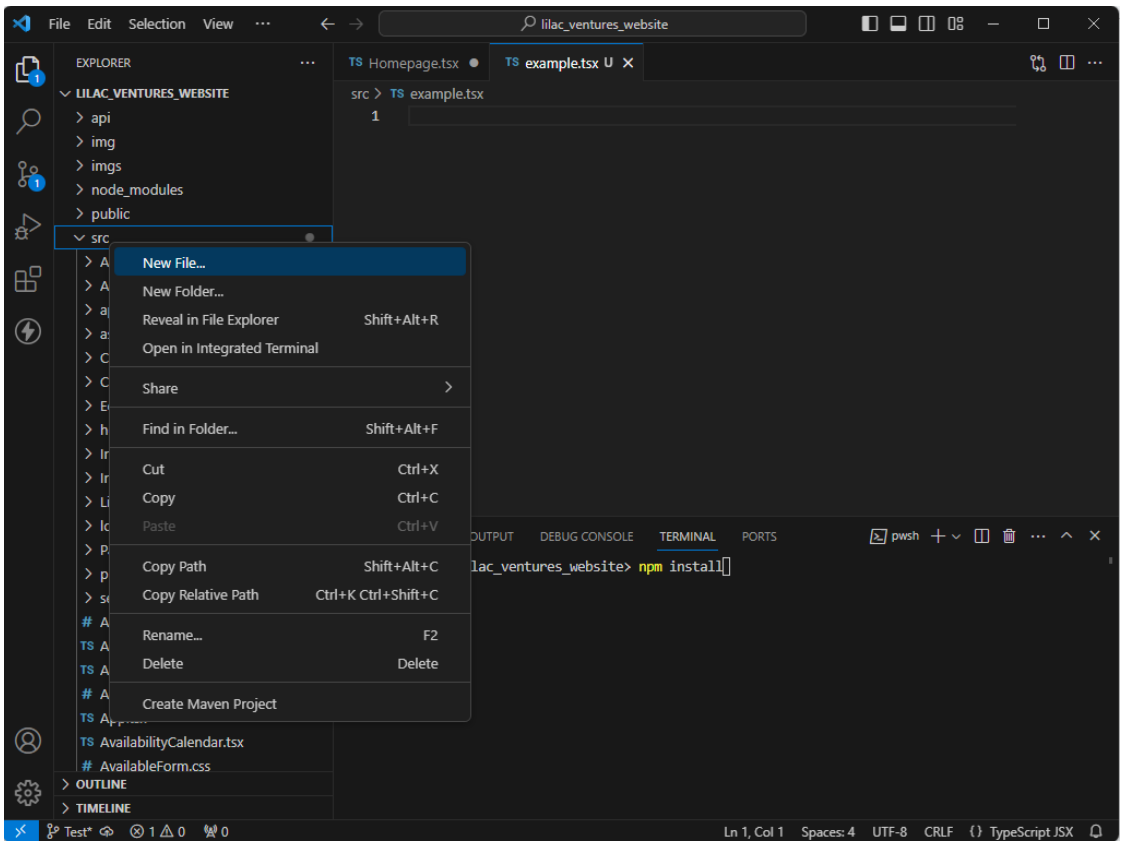


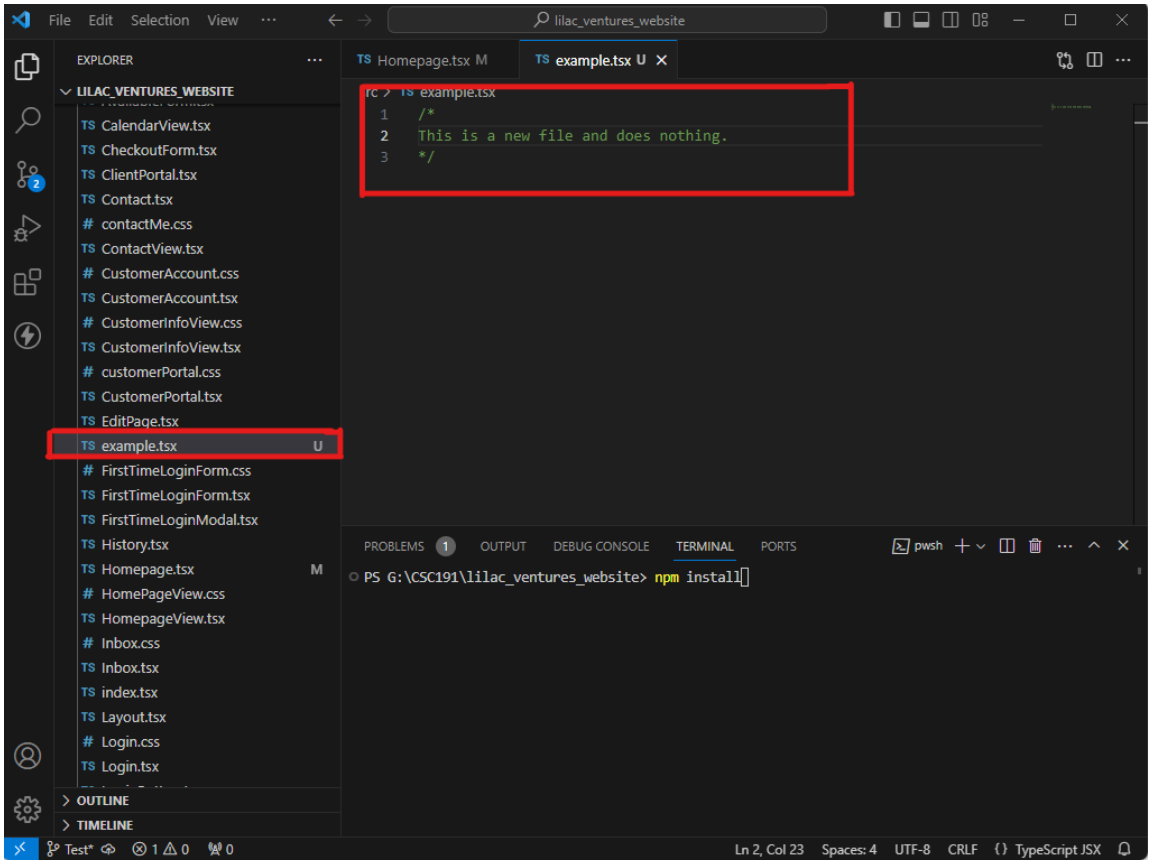
d. Enter the name of the new branch. Once done, hit the enter key



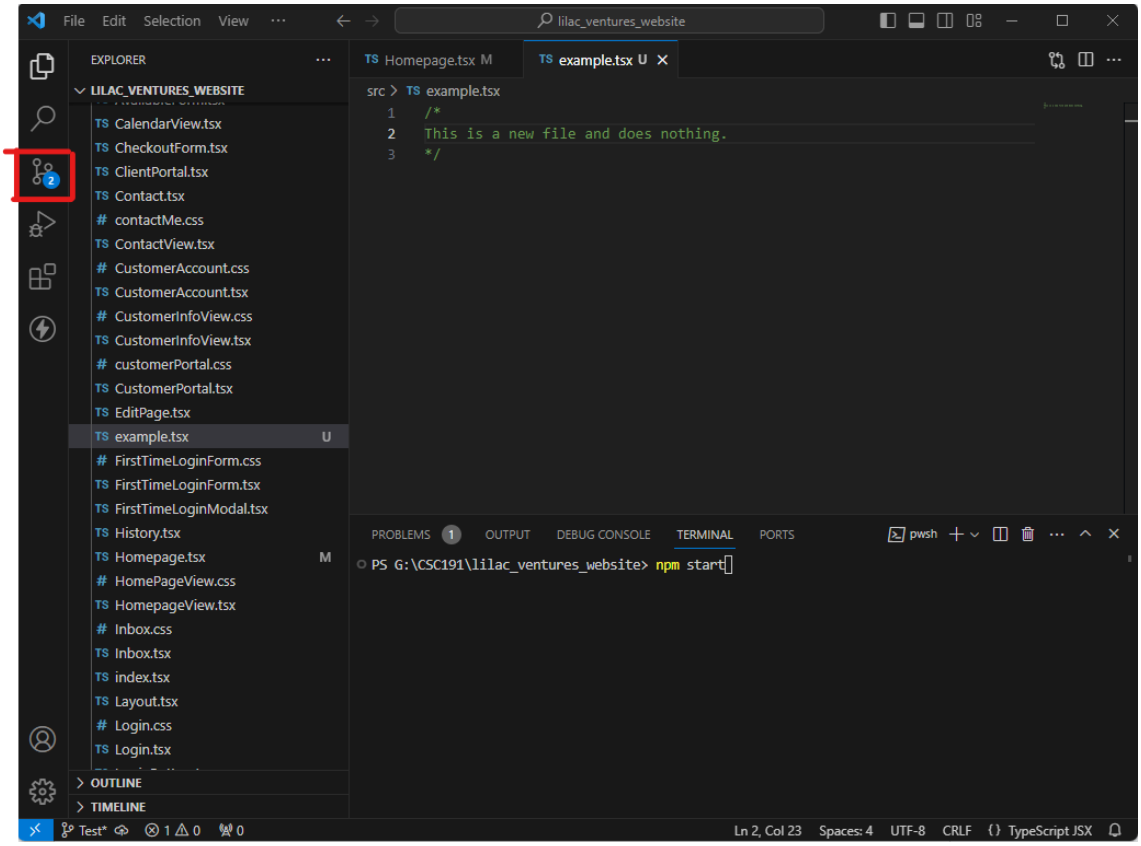
e. Open and edit any necessary file or right click on the appropriate folder and select new file to add new files from the Explorer Tab


File Edit Selection View
lilac_ventures_website
EXPLORER
LILAC_VENTURES_WEBSITE
ContactView.tsx
CustomerAccount.css
CustomerAccount.tsx
CustomerInfoView.css
CustomerInfoView.tsx
customerPortal.css
CustomerPortal.tsx
EditPage.tsx
FirstTimeLoginForm.css
FirstTimeLoginForm.tsx
FirstTimeLoginModal.tsx
History.tsx
Homepage.tsx
HomePageView.css
HomepageView.tsx
Inbox.css
Inbox.tsx
index.tsx
Layout.tsx
Login.css
Login.tsx
LoginButton.tsx
LogoutButton.tsx
Mission.tsx
navbar.css
NavBar.tsx
NavbarData.tsx
OwnerDashboard.tsx
OUTLINE
TIMELINE
Homepage.tsx
src > Homepage.tsx > ...
1 import React from "react";
2 import money from "./assets/img/homepage-photo-gigapixel-low_res-scale-2
3 import { Link, useNavigate } from "react-router-dom";
4 import HomepageView from "./HomepageView";
5 import useAuth from "./hooks/useAuth";
6
7 /*
8 This is an edit I am making that doesn't do anything.
9 Realistically this wouldn't be a comment and would serve a purpose.
10 */
11
12 function Homepage() {
13 let navigate = useNavigate();
14 const routeChange = () => {
15 let path = `/contact`;
16 navigate(path);
17 };
18 const {auth, setAuth}: any = useAuth();
19
20 return (
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS
pwsh
PS G:\CSC191\lilac_ventures_website> npm install
Test
1 0
0
Ln 9, Col 68 Spaces: 2 UTF-8 CRLF {} TypeScript JSX

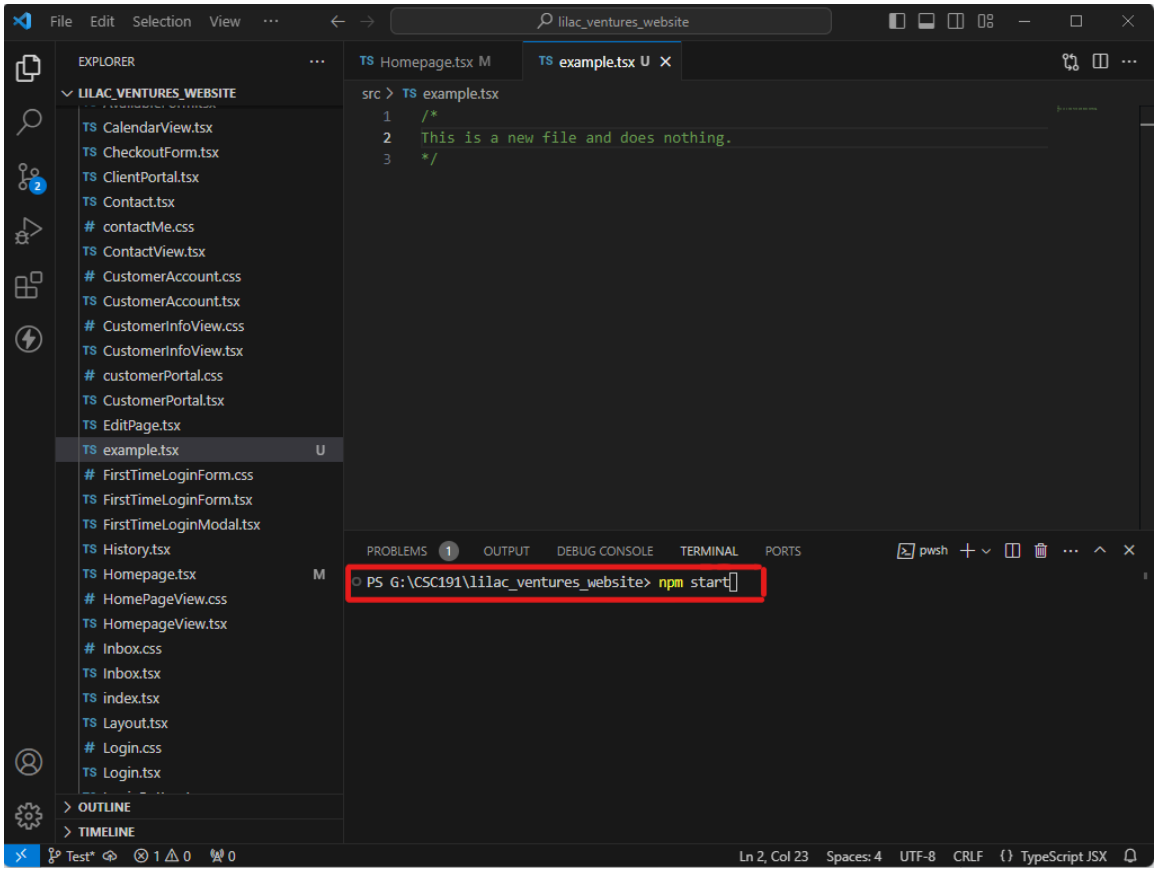

File Edit Selection View
lilac_ventures_website
EXPLORER
LILAC_VENTURES_WEBSITE
api
img
imgs
node_modules
public
src
AvailabilityCalendar.tsx
AvailableForm.css
OUTLINE
TIMELINE
Homepage.tsx
example.tsx U
src > example.tsx
1
New File...
New Folder...
Reveal in File Explorer Shift+Alt+R
Open in Integrated Terminal
Share
Find in Folder... Shift+Alt+F
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Copy Path Shift+Alt+C
Copy Relative Path Ctrl+K Ctrl+Shift+C
Rename... F2
Delete Delete
Create Maven Project
OUTPUT DEBUG CONSOLE TERMINAL PORTS
pwsh
lac_ventures_website> npm install
Test*
1 0
0
Ln 1, Col 1 Spaces: 4 UTF-8 CRLF {} TypeScript JSX



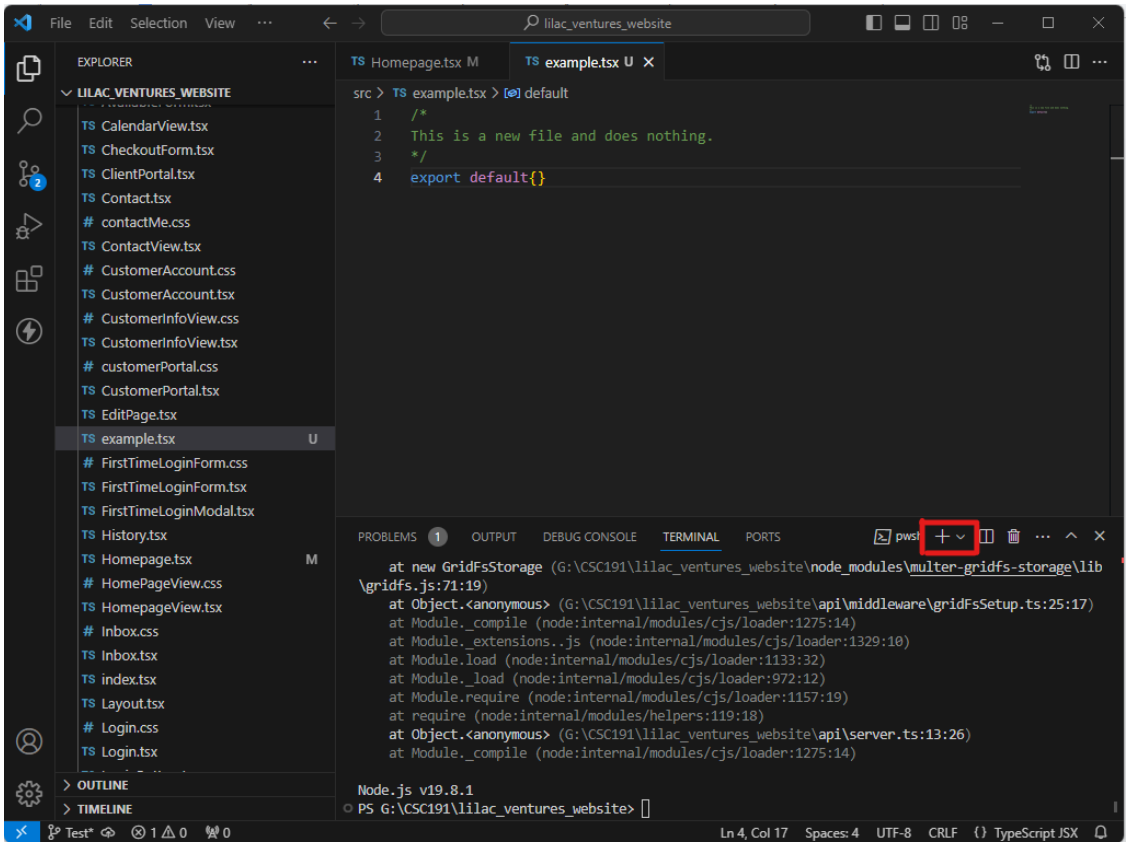
f. Press Ctrl+s to save any changes. Save changes will be added to the source control tab.



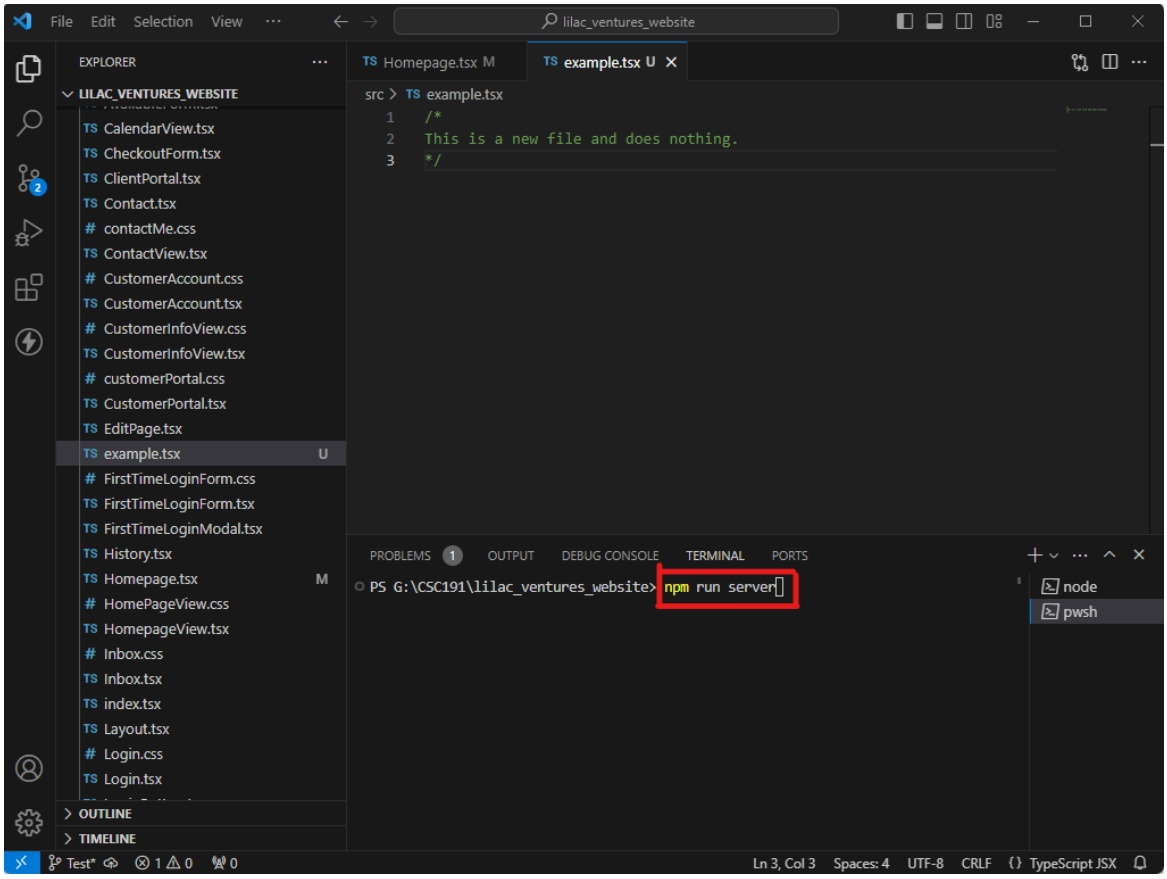
g. To test the changes, in a terminal, type npm start and press enter. This will load the frontend components. Note: The site will not have full functionality until the backend is loaded, details are in the next step.



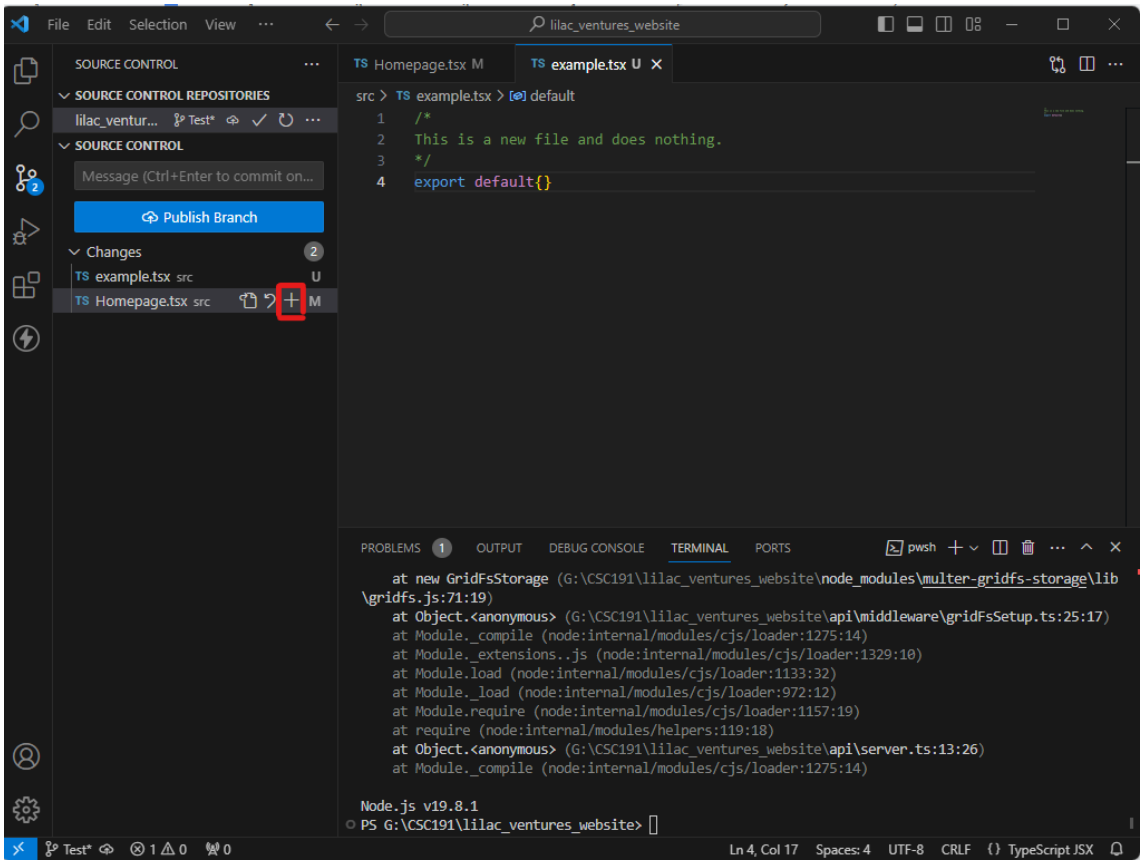
h. To load the backend, click the plus icon in the top right of the terminal window to open a new terminal



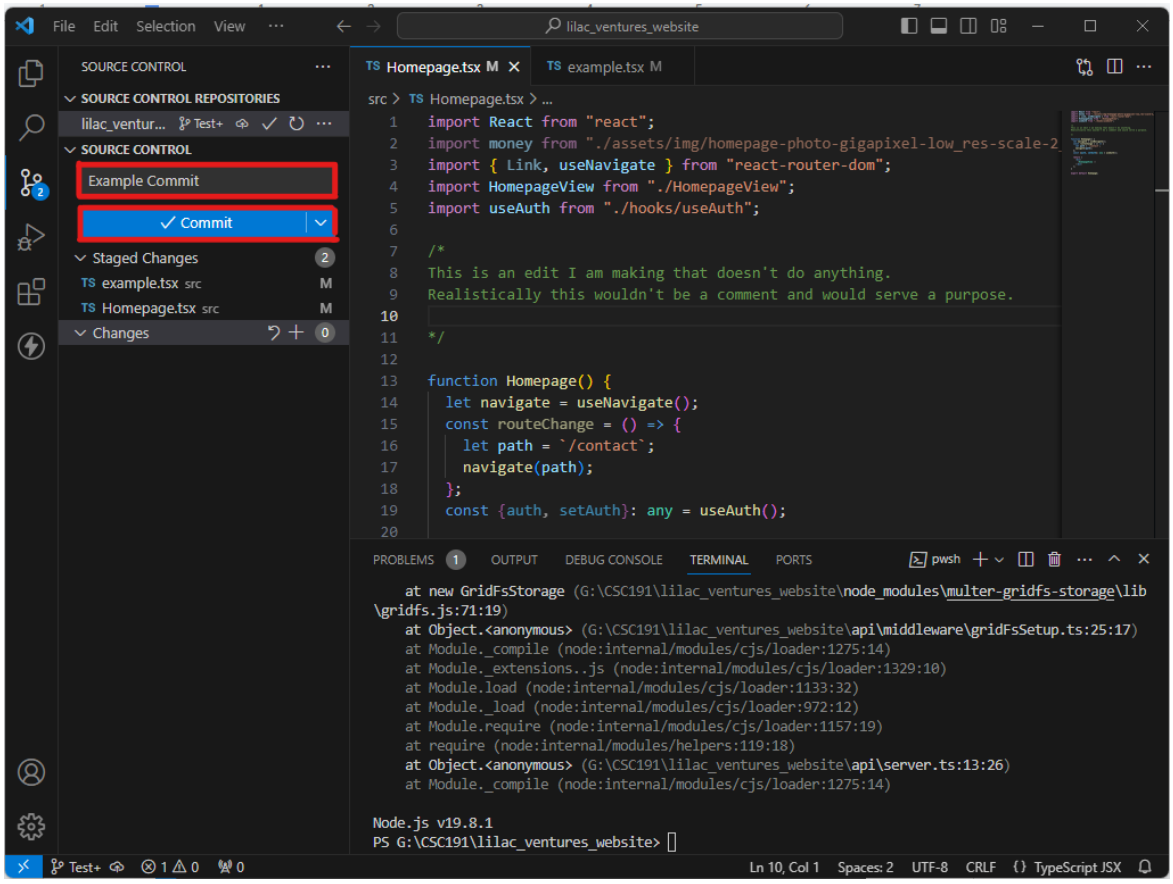
i. In the new terminal, type npm run server and press enter. This will start a locally hosted server for API calls to the MongoDB database.



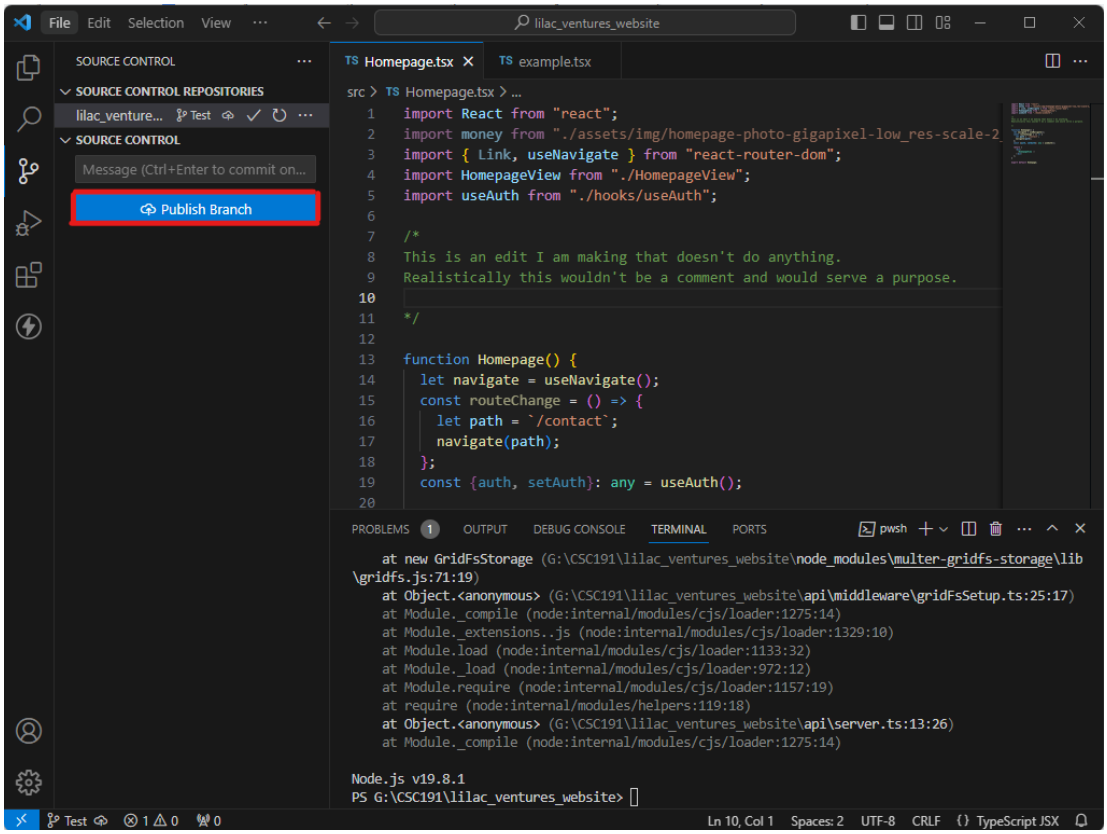
j. Once a coding task has been completed and all changed or new files have been saved navigate to the Source Control tab. Click the + symbol next to each file name in the Changes section.



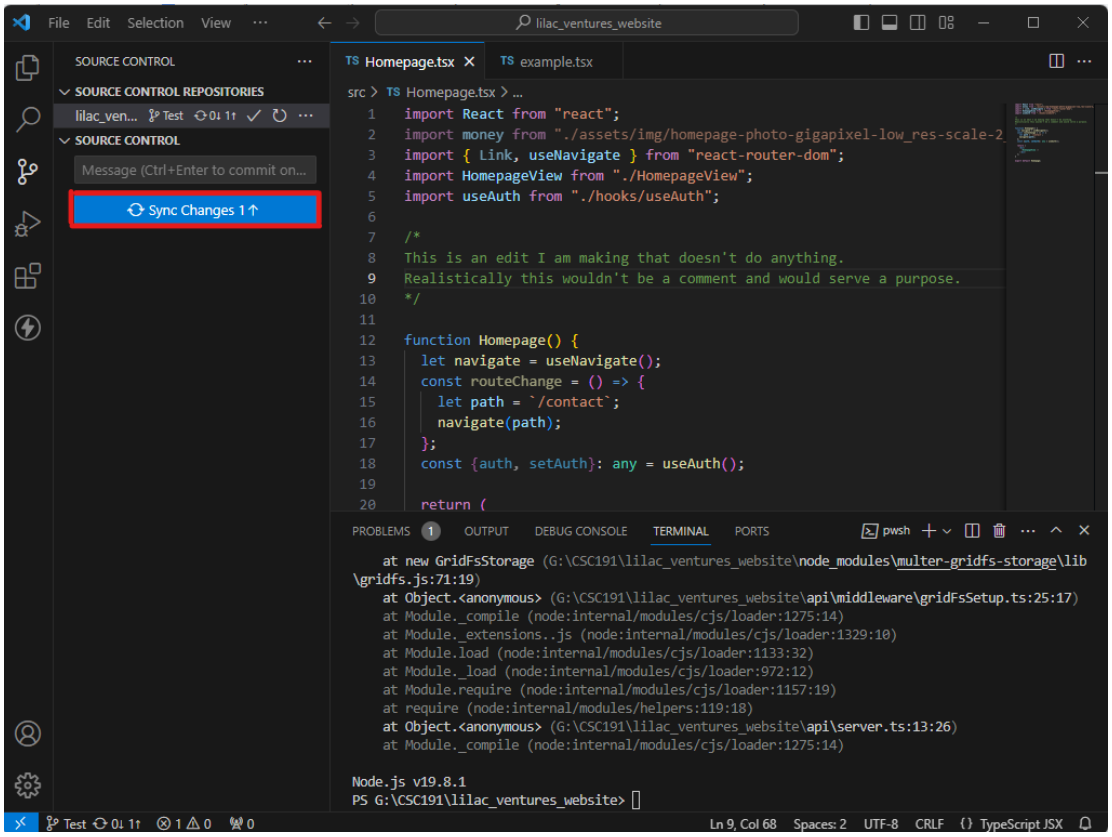
k. Once the changes are staged, type a common message then click the Commit button.



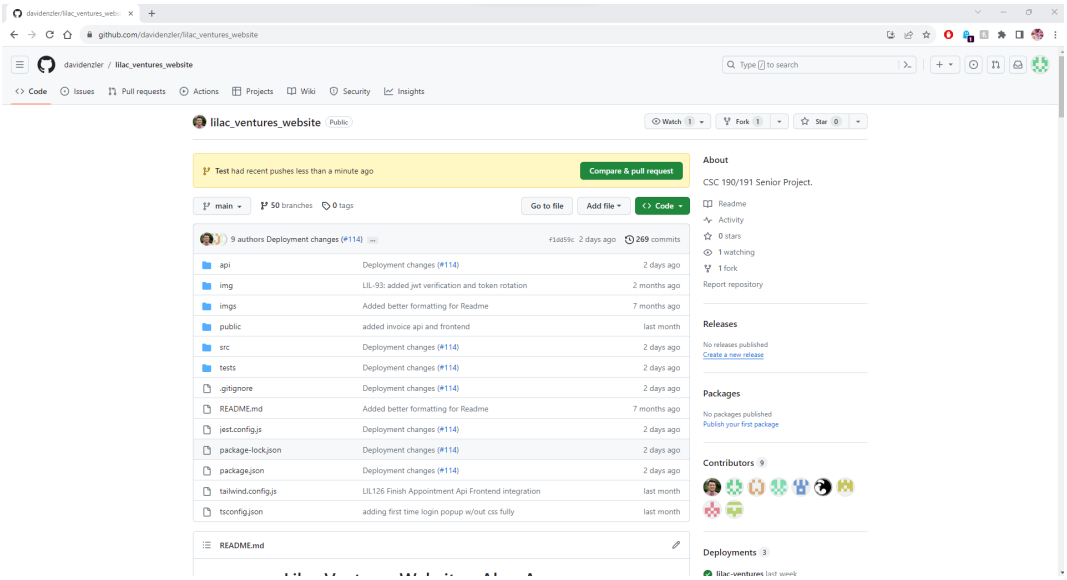
I. Click the Publish Branch button.



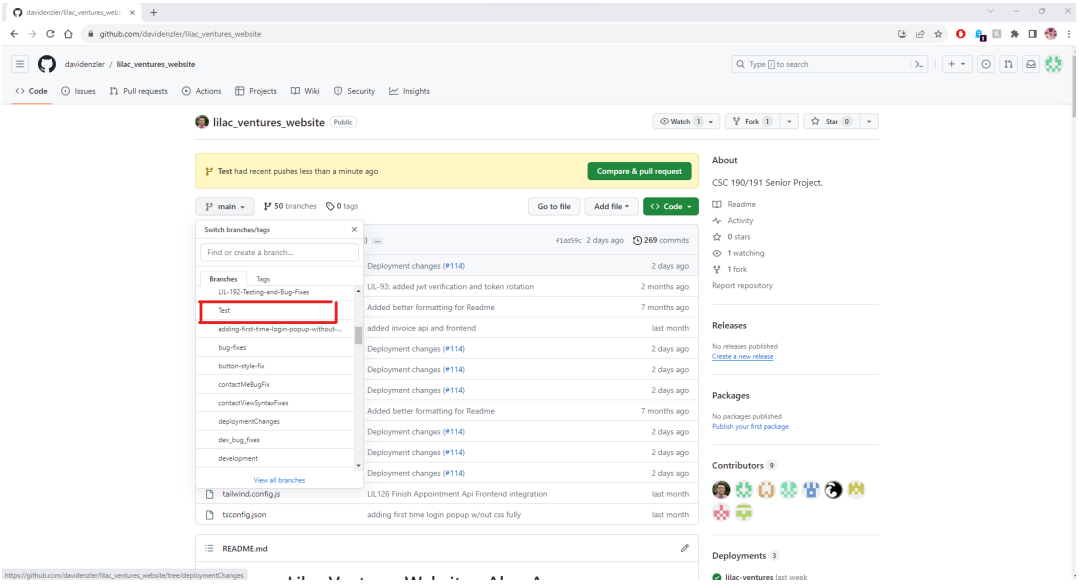
m. Repeat Steps e-f and j-k for each coding task. Additional Commits will only be synced with the repository once the Sync Branch button is pressed after commits are completed.



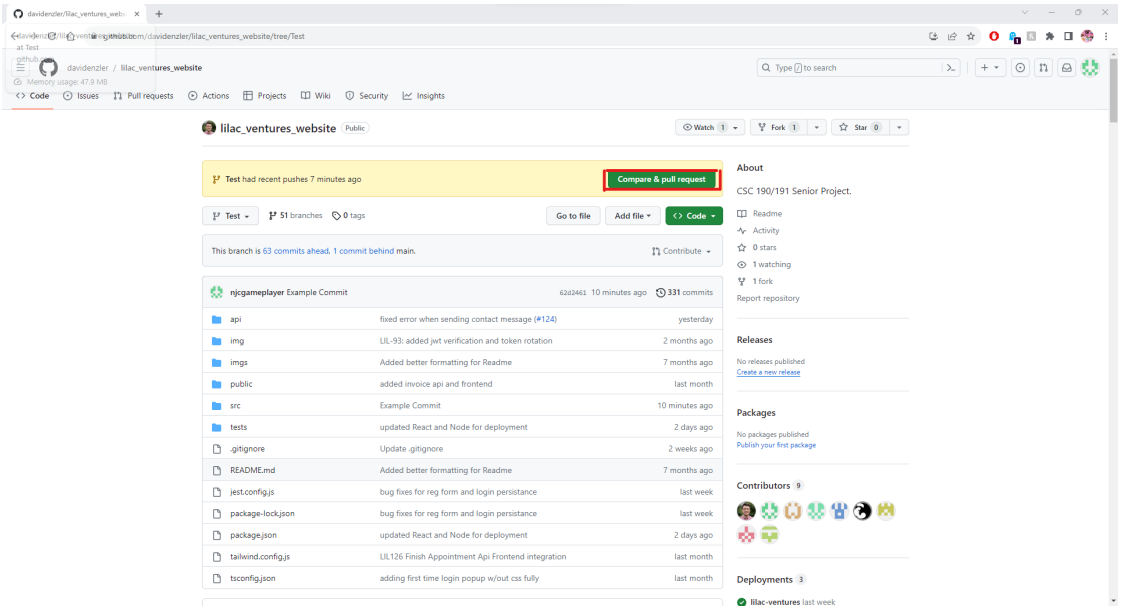
n. Once all changes are complete and thoroughly tested for bugs/errors, go to the GitHub repository in a web browser.



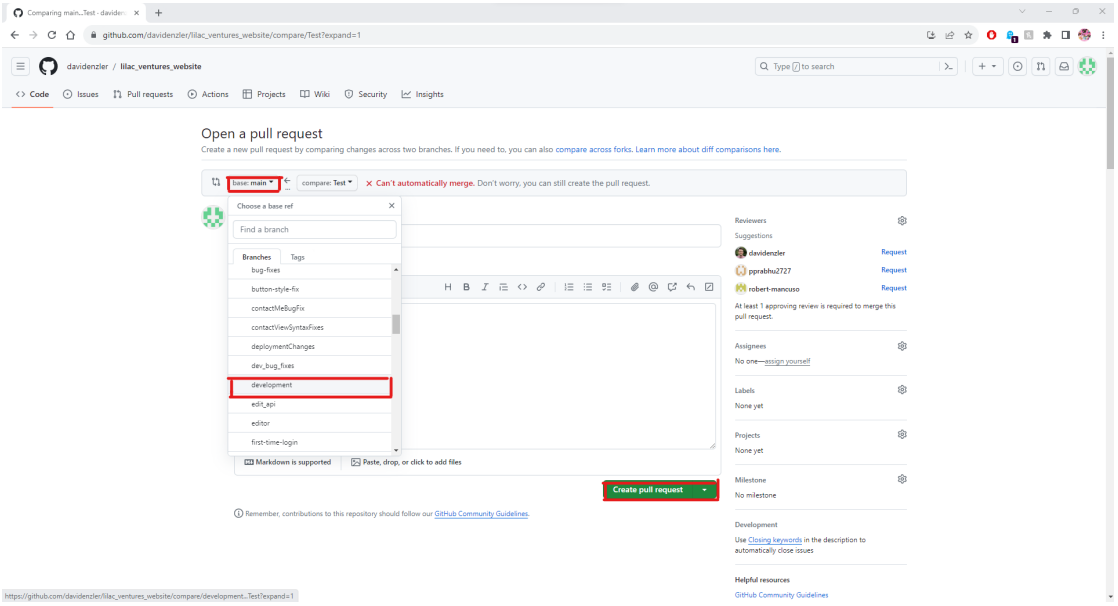
o. Change the branch from main to the new branch.



p. Click on Compare and Pull Request



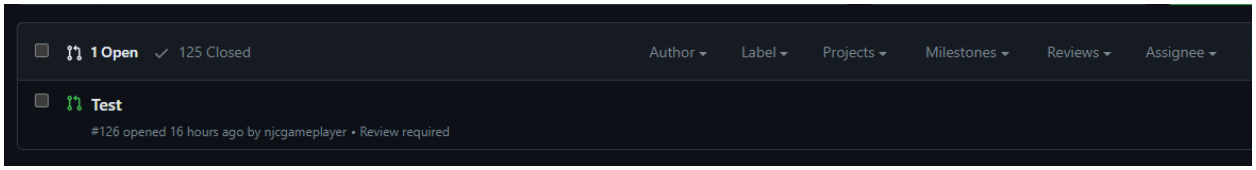
q. Change the destination branch from main to development. Then click Create Pull Request



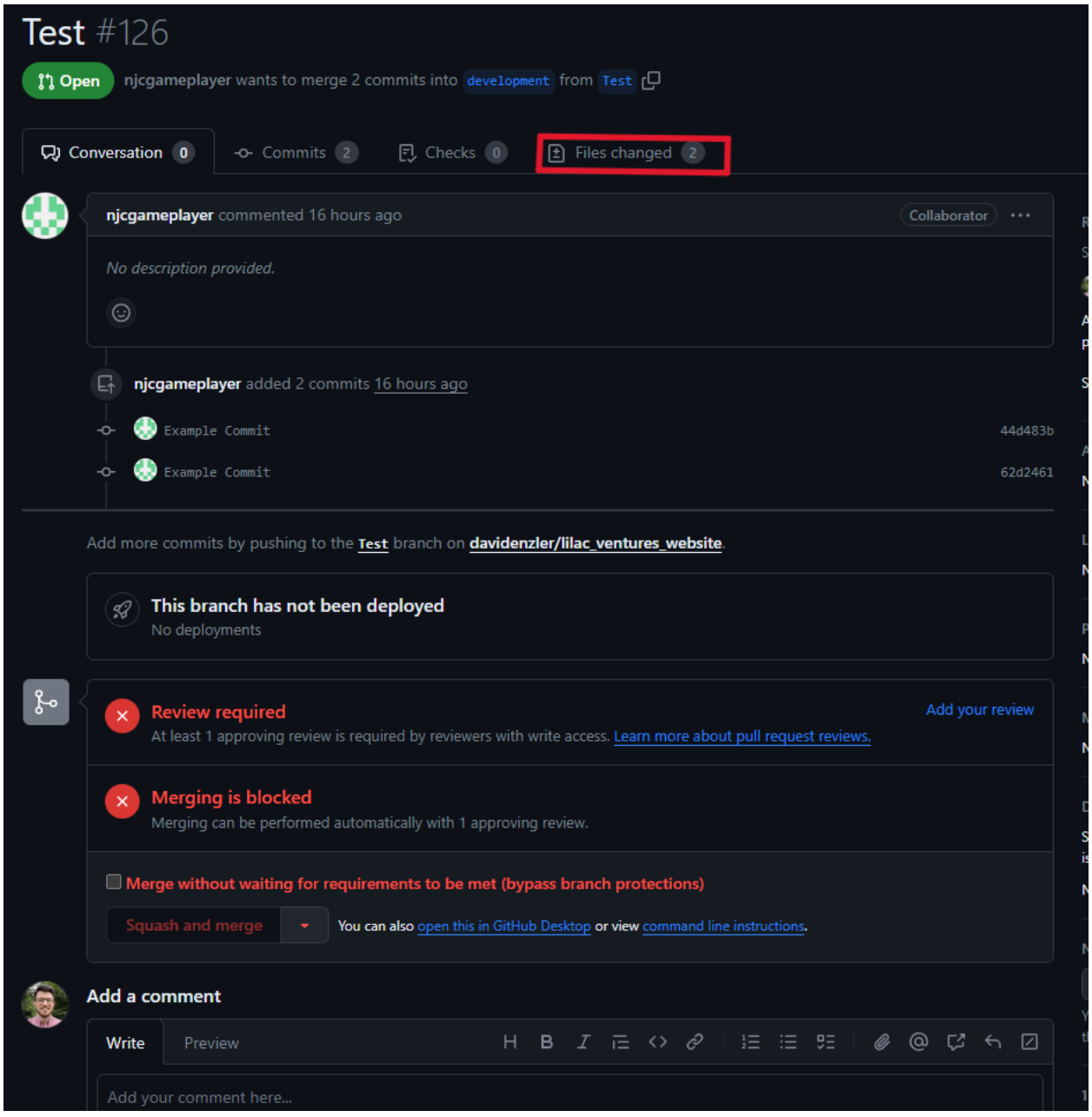
r. A collaborator with writing privileges will be able to resolve any version conflicts created by merging the branches and then completing the merge. Afterwards, the pull request will be closed.

Merging branches from pull request

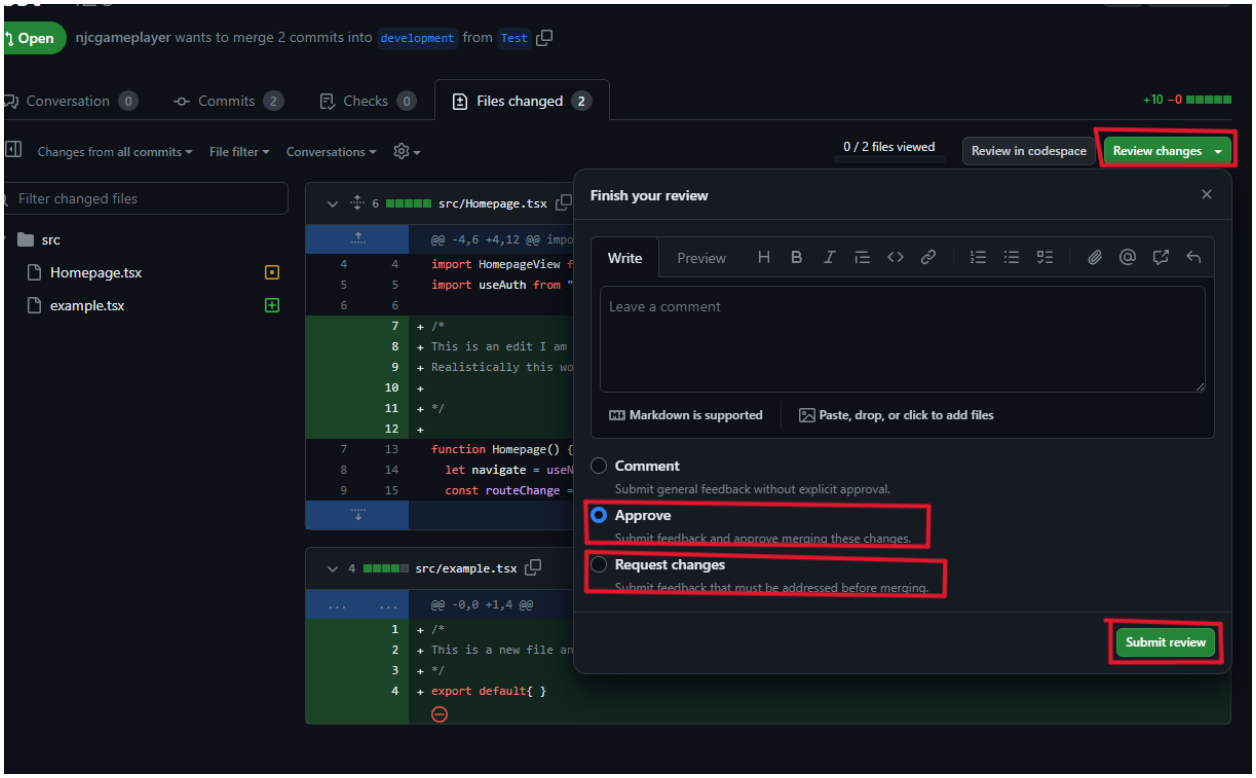
a. As a collaborator with write privileges to the development branch, open the pull request



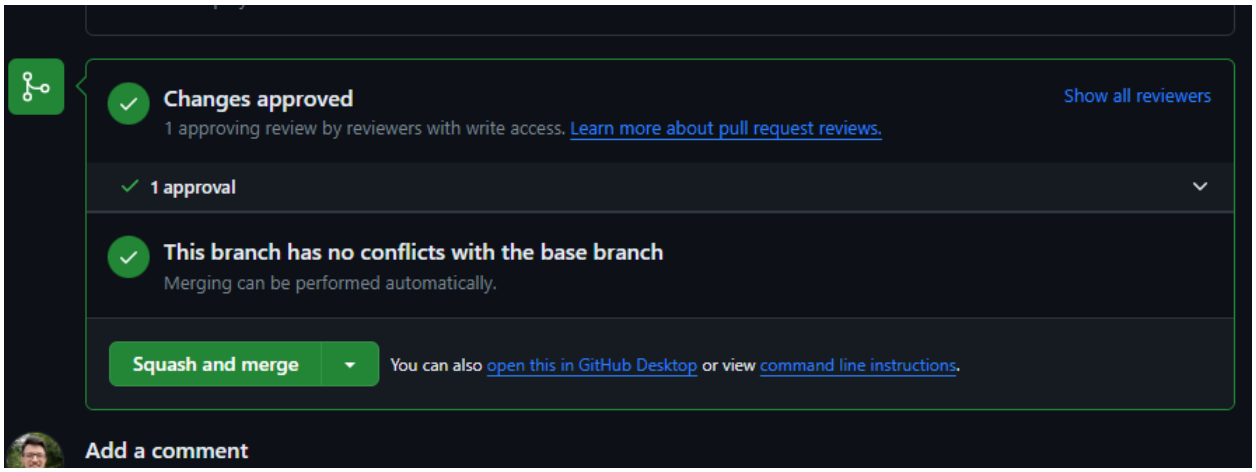
b. The pull request will note there are discrepancies in the code between the two branches and merging is blocked. Go to the file changes tab to review the discrepancies.

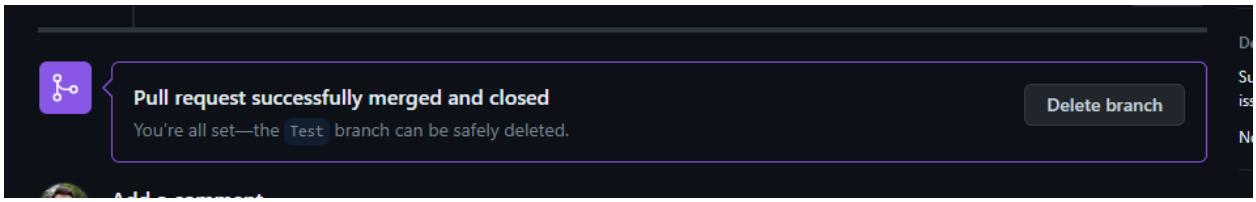


c. Review the changes and verify that the changes will not affect other functionality. Press the Review Changes button. If further changes need to be made, you can check the request changes box. Otherwise, the approve box can be checked. Press submit Review.



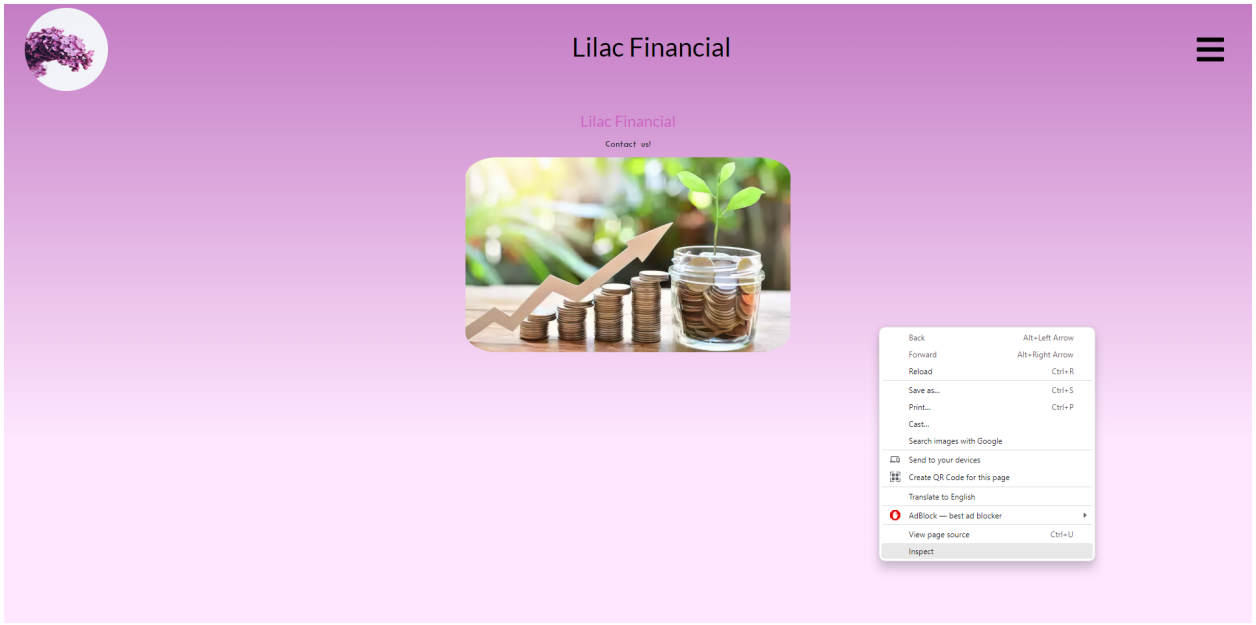
d. Once the code is ready to be merged, press the squash and merge button. This will complete the merge.



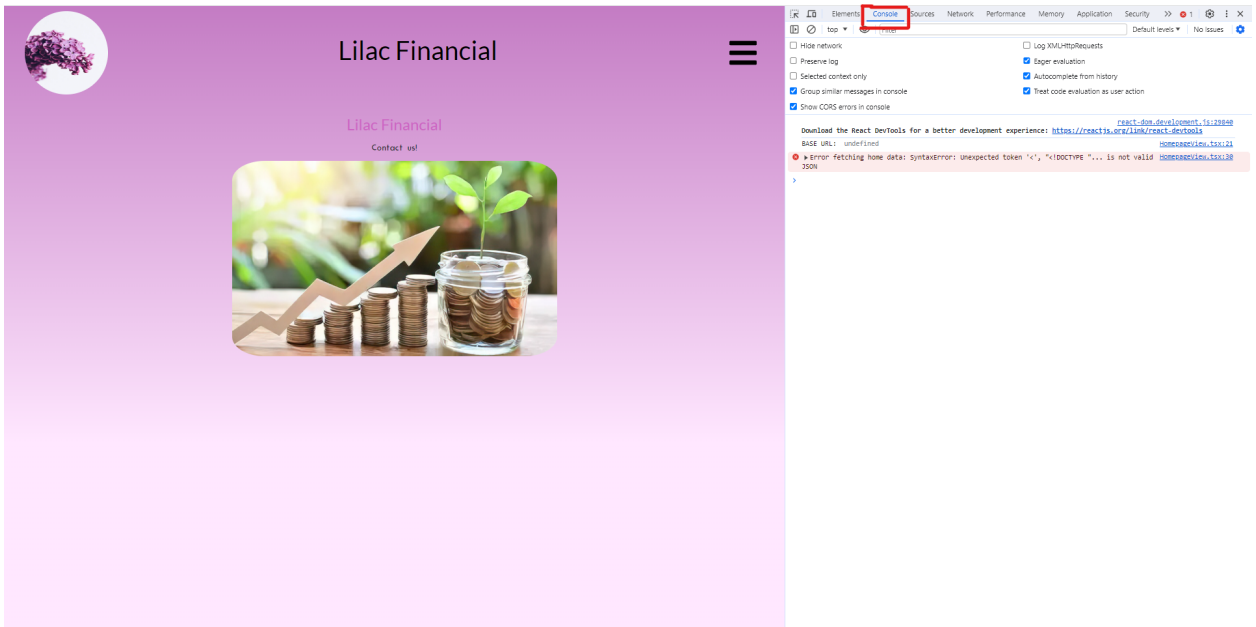


How to check for errors in the code

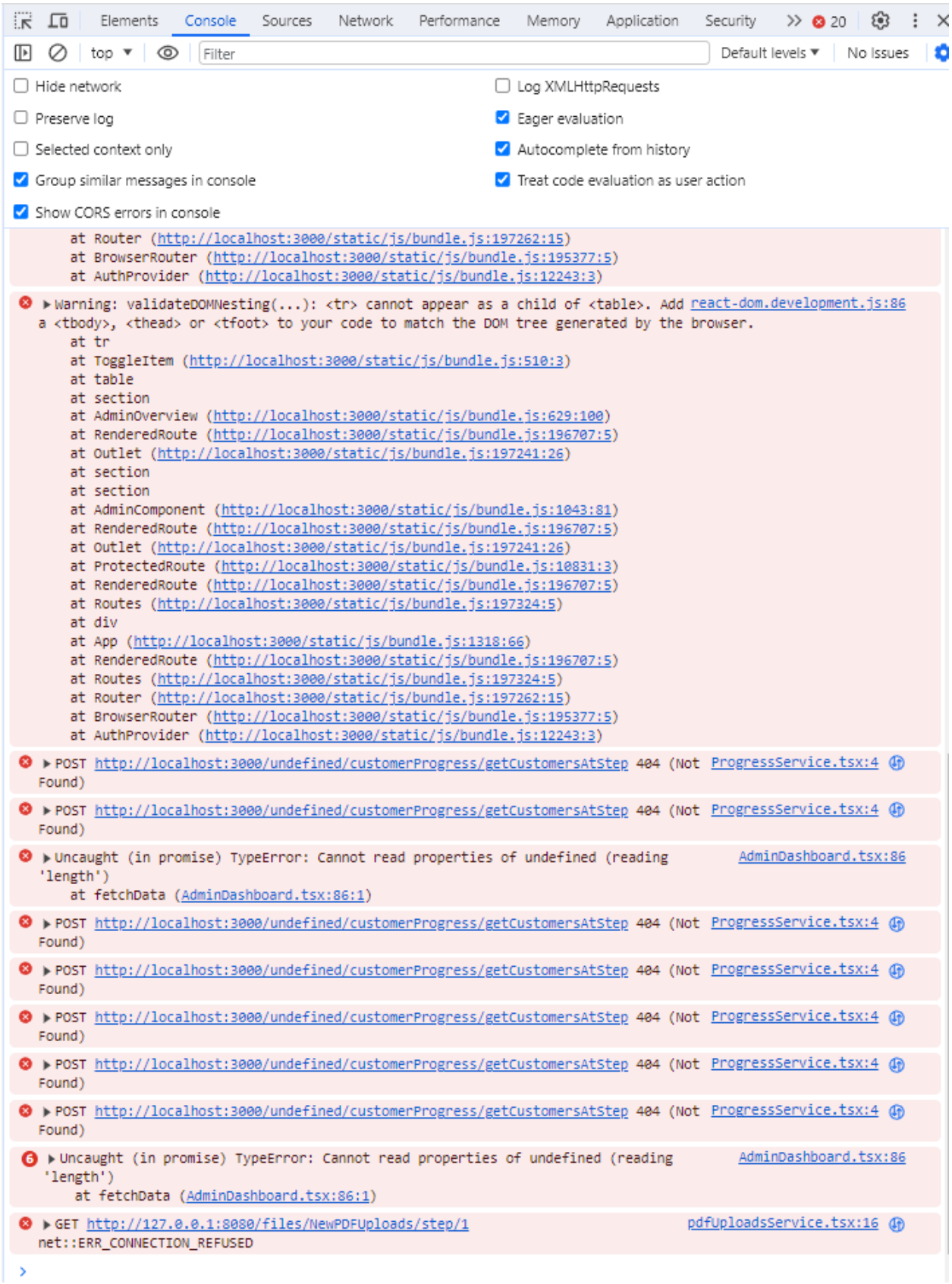
a. Errors typically are caught and logged into the browser's console. To access the console, with the site open, either press F12 or right click on the screen and select inspect



b. Click the console tab



c. Any errors or notable events will be sent to the console


Elements
Console
Sources
Network
Performance
Memory
Application
Security
20
top
Filter
Default levels
No Issues
Hide network
Preserve log
Selected context only
Group similar messages in console
Show CORS errors in console
Log XMLHttpRequests
Eager evaluation
Autocomplete from history
Treat code evaluation as user action
at Router (http://localhost:3000/static/js/bundle.js:197262:15)
at BrowserRouter (http://localhost:3000/static/js/bundle.js:195377:5)
at AuthProvider (http://localhost:3000/static/js/bundle.js:12243:3)
Warning: validateDOMNesting(...): <tr> cannot appear as a child of <table>. Add a <tbody>, <thead> or <tfoot> to your code to match the DOM tree generated by the browser.
react-dom.development.js:86
at tr
at ToggleItem (http://localhost:3000/static/js/bundle.js:510:3)
at table
at section
at AdminOverview (http://localhost:3000/static/js/bundle.js:629:100)
at RenderedRoute (http://localhost:3000/static/js/bundle.js:196707:5)
at Outlet (http://localhost:3000/static/js/bundle.js:197241:26)
at section
at section
at AdminComponent (http://localhost:3000/static/js/bundle.js:1043:81)
at RenderedRoute (http://localhost:3000/static/js/bundle.js:196707:5)
at Outlet (http://localhost:3000/static/js/bundle.js:197241:26)
at ProtectedRoute (http://localhost:3000/static/js/bundle.js:10831:3)
at RenderedRoute (http://localhost:3000/static/js/bundle.js:196707:5)
at Routes (http://localhost:3000/static/js/bundle.js:197324:5)
at div
at App (http://localhost:3000/static/js/bundle.js:1318:66)
at RenderedRoute (http://localhost:3000/static/js/bundle.js:196707:5)
at Routes (http://localhost:3000/static/js/bundle.js:197324:5)
at Router (http://localhost:3000/static/js/bundle.js:197262:15)
at BrowserRouter (http://localhost:3000/static/js/bundle.js:195377:5)
at AuthProvider (http://localhost:3000/static/js/bundle.js:12243:3)
POST http://localhost:3000/undefined/customerProgress/getCustomersAtStep 404 (Not Found)
ProgressService.tsx:4
POST http://localhost:3000/undefined/customerProgress/getCustomersAtStep 404 (Not Found)
ProgressService.tsx:4
Uncaught (in promise) TypeError: Cannot read properties of undefined (reading 'length')
at fetchData (AdminDashboard.tsx:86:1)
AdminDashboard.tsx:86
POST http://localhost:3000/undefined/customerProgress/getCustomersAtStep 404 (Not Found)
ProgressService.tsx:4
POST http://localhost:3000/undefined/customerProgress/getCustomersAtStep 404 (Not Found)
ProgressService.tsx:4
POST http://localhost:3000/undefined/customerProgress/getCustomersAtStep 404 (Not Found)
ProgressService.tsx:4
POST http://localhost:3000/undefined/customerProgress/getCustomersAtStep 404 (Not Found)
ProgressService.tsx:4
POST http://localhost:3000/undefined/customerProgress/getCustomersAtStep 404 (Not Found)
ProgressService.tsx:4
6
Uncaught (in promise) TypeError: Cannot read properties of undefined (reading 'length')
at fetchData (AdminDashboard.tsx:86:1)
AdminDashboard.tsx:86
GET http://127.0.0.1:8080/files/NewPDFUploads/step/1 net::ERR_CONNECTION_REFUSED
pdfUploadsService.tsx:16

4.0 Deployment

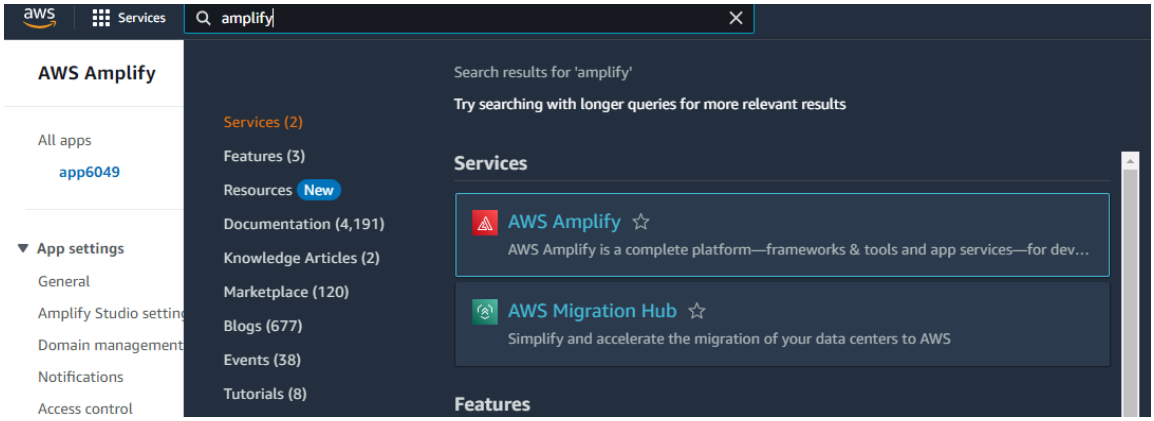
Describe how your application is currently deployed

Overview

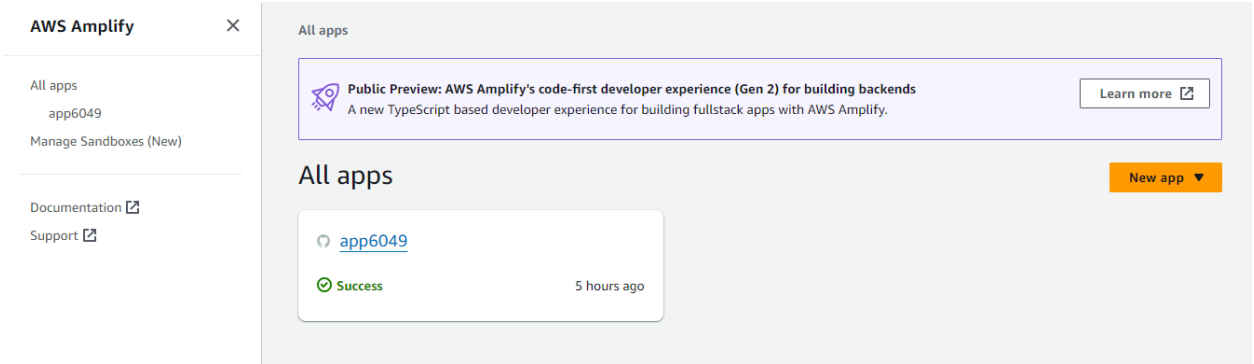
The application is currently deployed using AWS. The front end and back end are both hosted separately. The front-end React code is hosted using AWS Amplify. The backend NodeJS is hosted using AWS Elastic Beanstalk.

Amplify

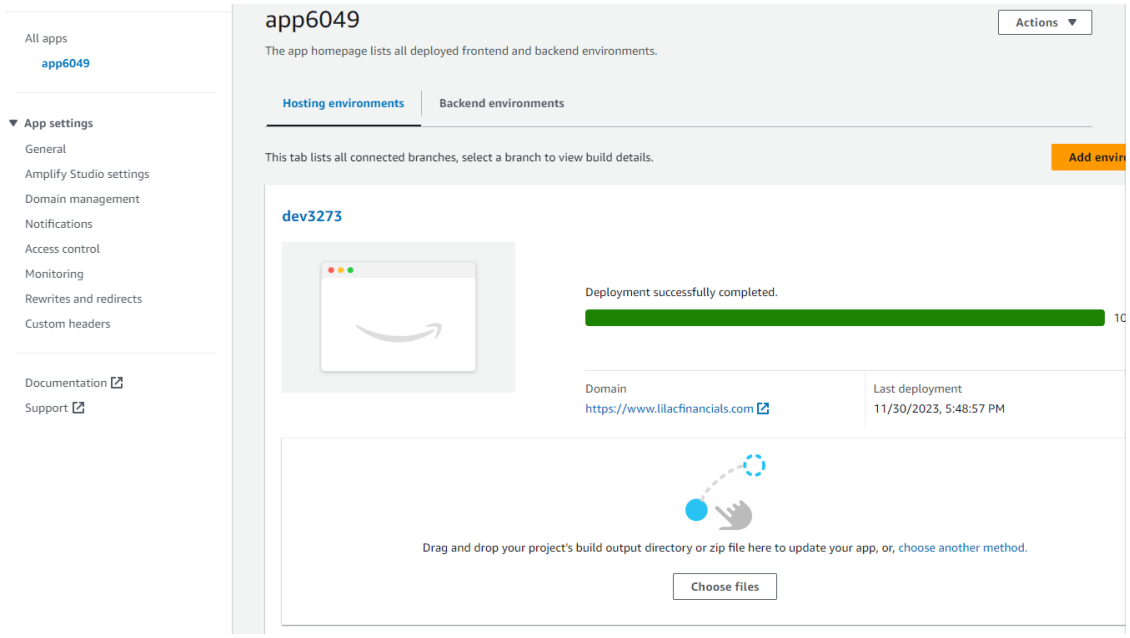
1. Login into the AWS console using the AWS credentials.
2. Search the navigation bar for “Amplify”. Click on “AWS Amplify”.



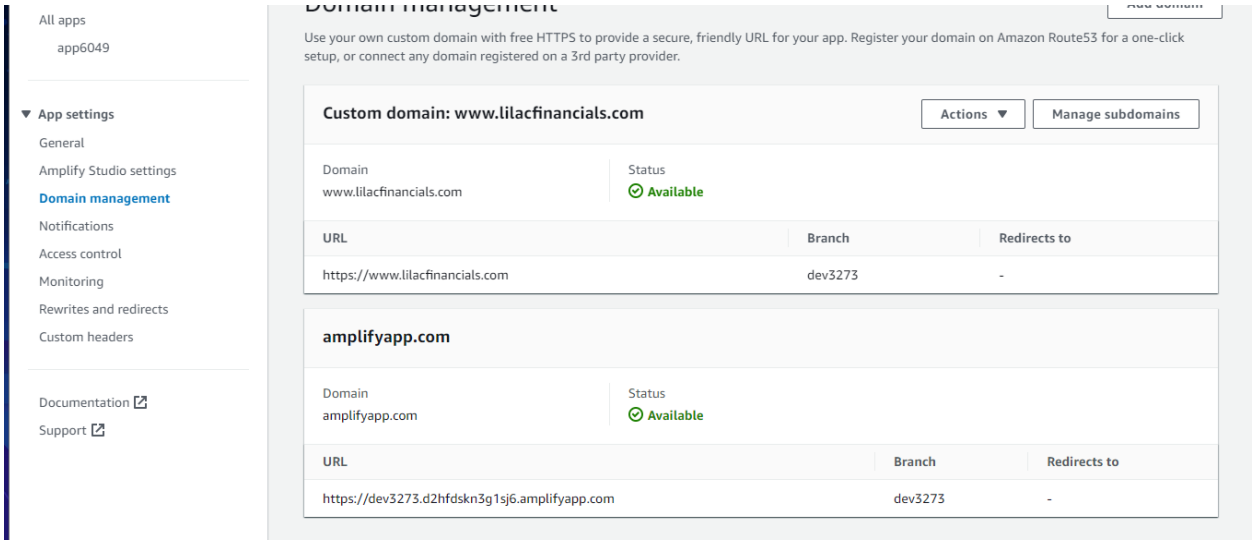
3. All currently running instances will appear in the console. Click on the running application.



4. The console for the application lists the health of the app, the domain of the app, and the current deployment status.



5. In the bottom of the screen, the build files for React can be dragged and dropped or uploaded from a local machine. Amplify will ingest the build files and automatically deploy them after a couple minutes.
6. The domain name and certs are managed from the “Domain Management” option in the left panel



7. The certs for this domain will automatically update, however, if there are routing issues, then the certs should be checked.

Building React Files for Deployment

1. In a Vscode terminal (or terminal at the root directory of the project), run the command `npm run build`

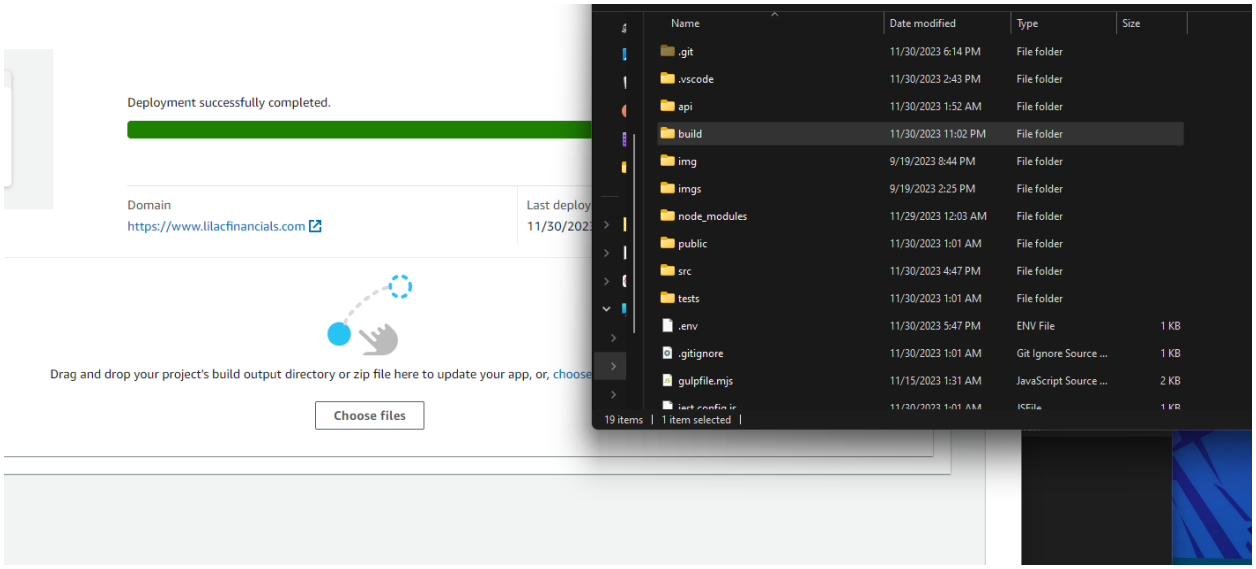
```
PS C:\Users\David\Documents\SacState\Spring 2023\190_senior\source\repos\lilac_ventures_website> npm
run build

> my-app@0.1.0 build
> react-scripts build

Creating an optimized production build...
```

2. The script will take a couple minutes to complete

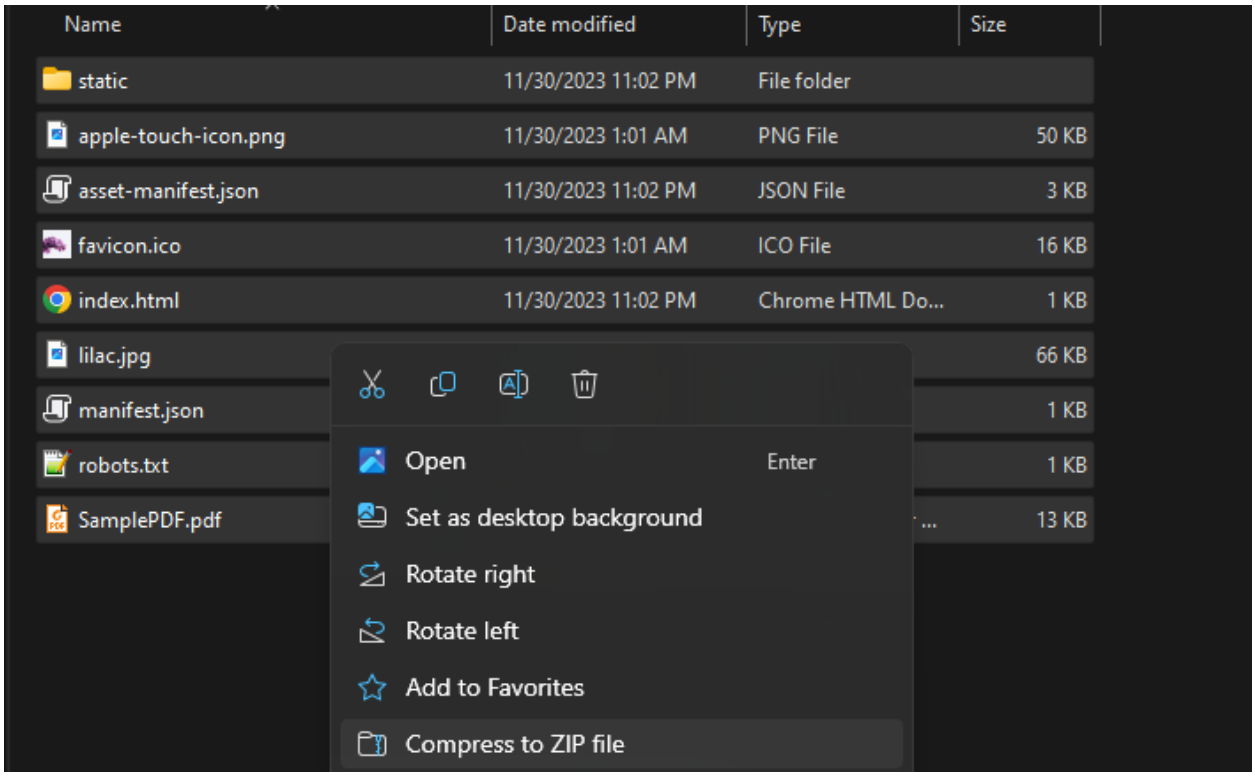
```
The build folder is ready to be deployed.
You may serve it with a static server:

  serve -s build

Find out more about deployment here:

  https://cra.link/deployment

PS C:\Users\David\Documents\SacState\Spring 2023\190_senior\source\repos\lilac_ventures_website>
```

3. When it is finished, the project root will have a new build directory. Upload this build directory to Amplify to deploy a new version of the site.

4. There are two options for uploading the code: either upload a .zip file or drag the whole build directory onto the console screen.
5. Drag the whole build directory to the console window



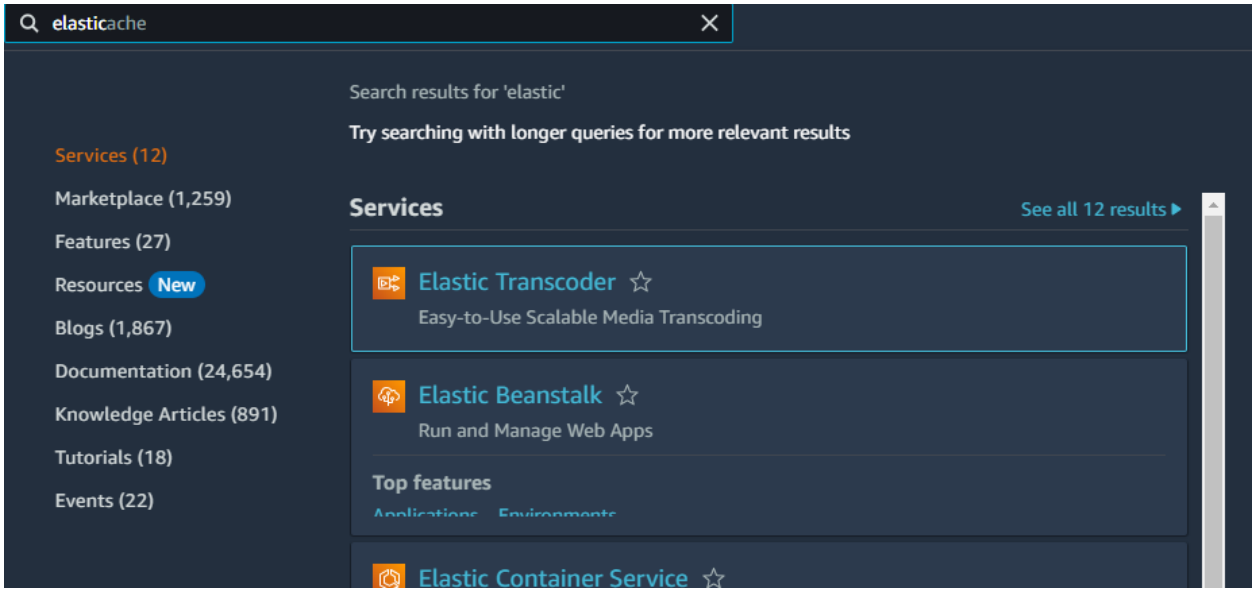
6. Alternatively, select the contents of the build directory and add to a zip archive. Upload this .zip file by clicking the ‘Choose File’ button and selecting the zip file.



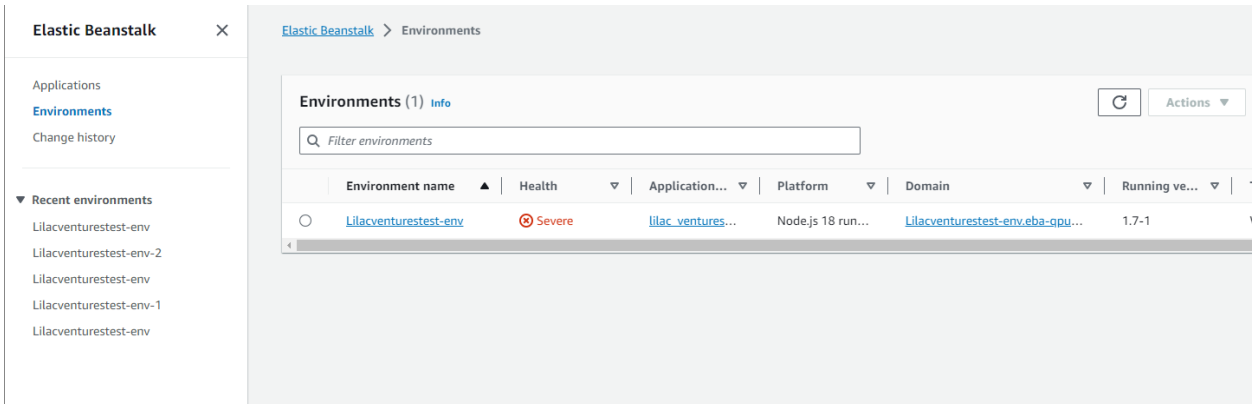
7. AWS has ample documentation on running applications on Amplify. For more information check out the link
    a. https://docs.aws.amazon.com/amplify/

Elastic Beanstalk

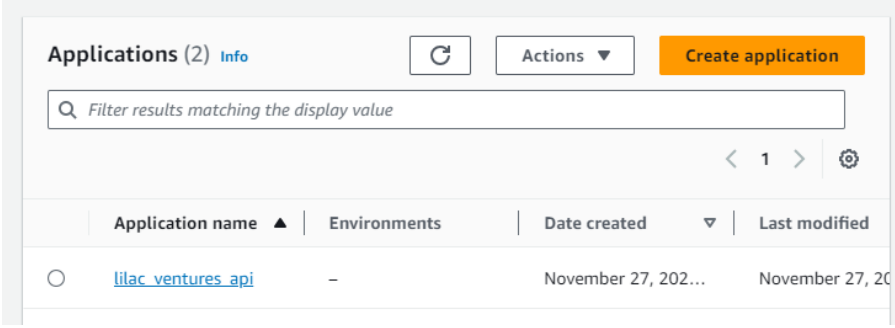
1. Log in to the AWS console with the AWS credentials.
2. Navigate to the ELB (Elastic Beanstalk) console by searching for `Elastic` in the search menu and selecting `Elastic Beanstalk`



3. The link will drop you into an overview of either the Application or the Environment. (The severe warning can be ignored in this example. The endpoints used for Health Checks was disabled at the time of this screenshot).

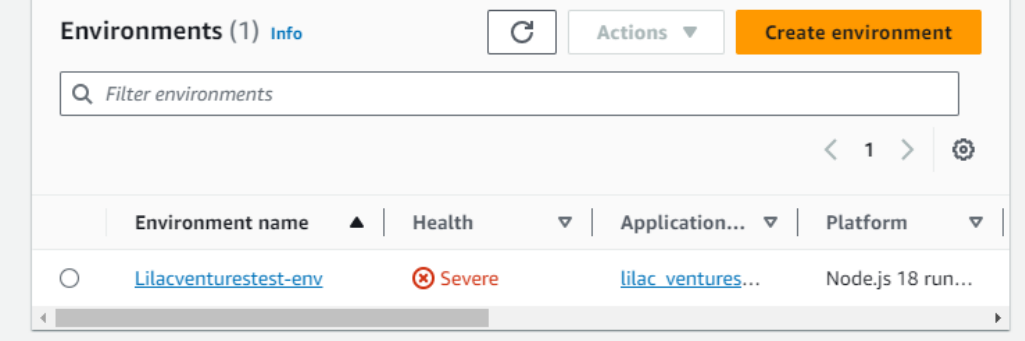


4. A Note on Application vs Environment
    a. Application: The NodeJS source code. Anything referring to application will refer to the code deployed by ELB. Applications can be individually configured and assigned resources.



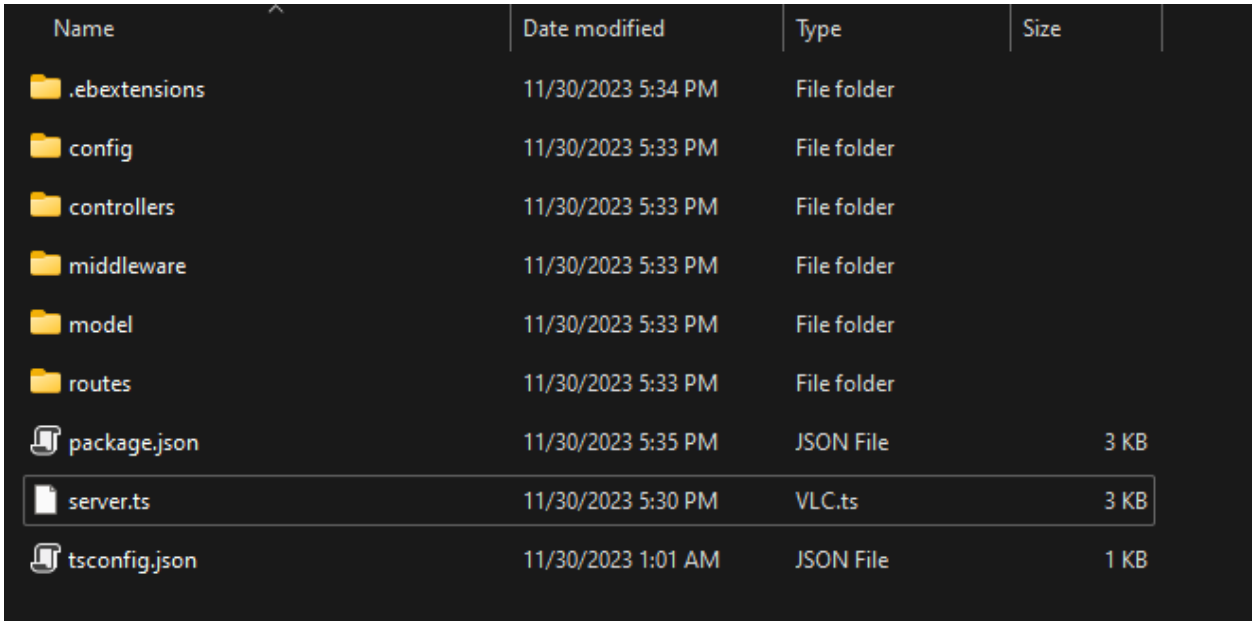
    b. Environment: This is the set of resources deployed by ELB to host the Application. Depending on the configuration, Beanstalk will stand up one or more

EC2 instances and a load balancer. The configuration for this site uses a single EC2 instance and a load balancer. This provides a cost effect solution while allowing for scaling as the site grows.



Deploying NodeJS Code to Existing Environment

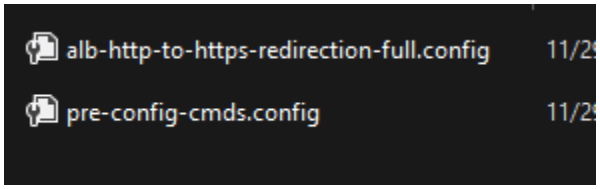
1. The following files will need to be zipped into an archive:
    a. The contents of `./api/`
    b. The package.json
    c. The tsconfig.json
    d. The .ebextensions folder and its contents
2. IMPORTANT: The structure of the zipped archive matters. Failing to follow the correct structure may result in a failed deployment
3. The following image is of the unzipped directory with all the appropriate files and directories



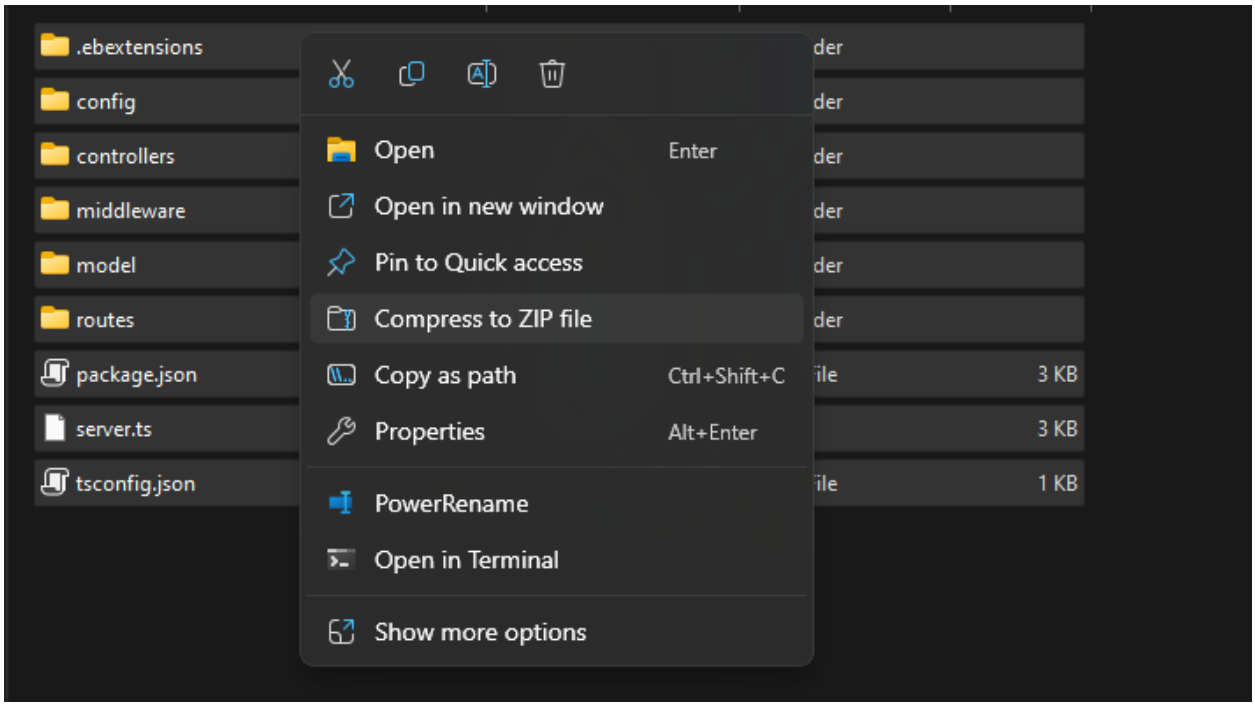
4. WARNING: `package.json` must be reconfigured for the ELB deployment process. Please ensure that the `scripts` field of package.json contains a start script with the following format

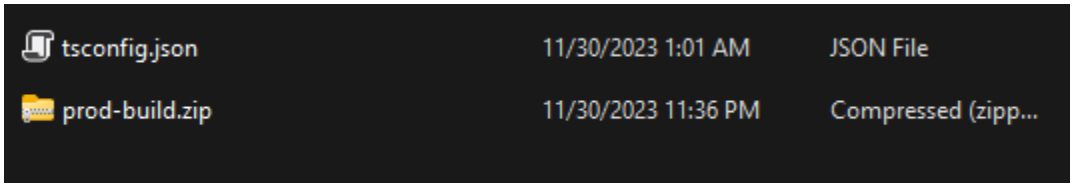
```
},
"scripts": {
  "start": "node ./api/server.ts",
},
```

5. Depending on the development environment, the package.json may need to be modified before it is zipped.
6. Important: please make sure that the .ebextensions folder has the following files:

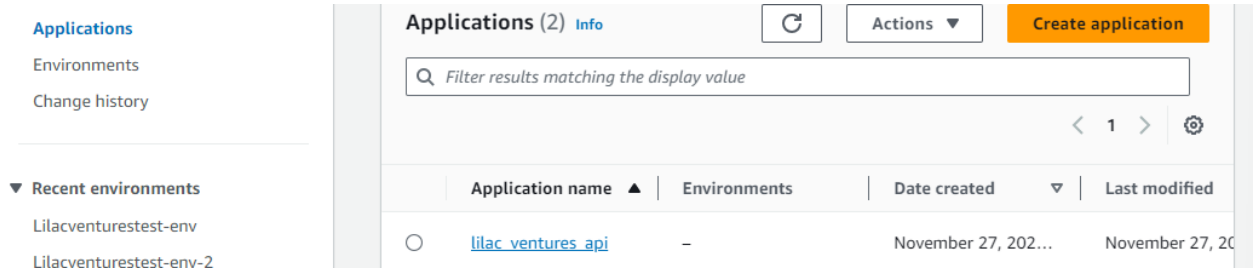


7. These are used to configure SSL and HTTP redirects in new environments.
8. Once all the files are in place, select all the files and directories and zip them together.

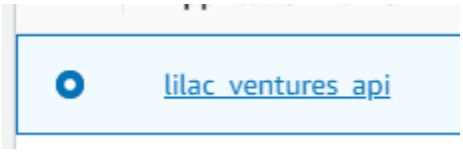


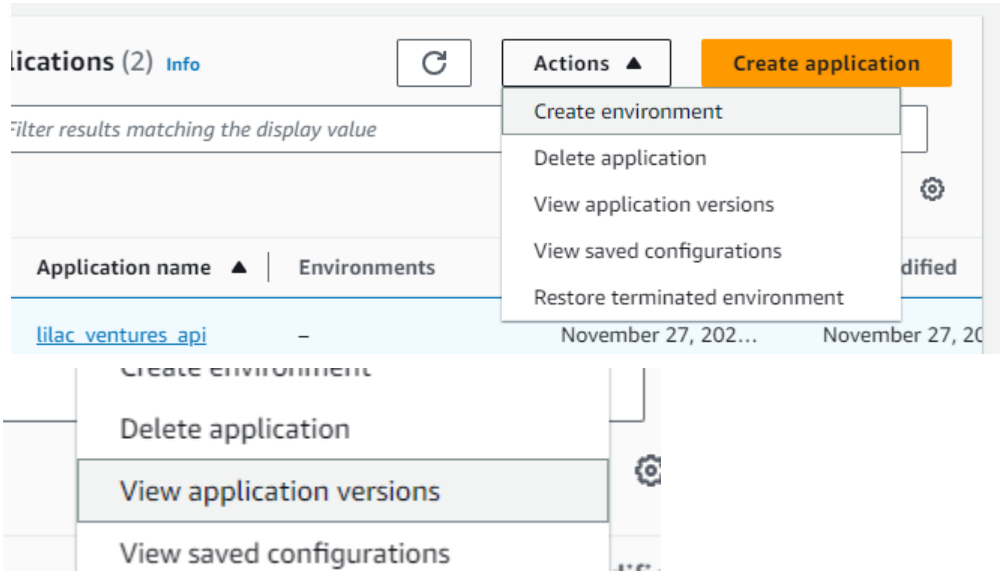


9. In the AWS Beanstalk console, navigate to the application overview

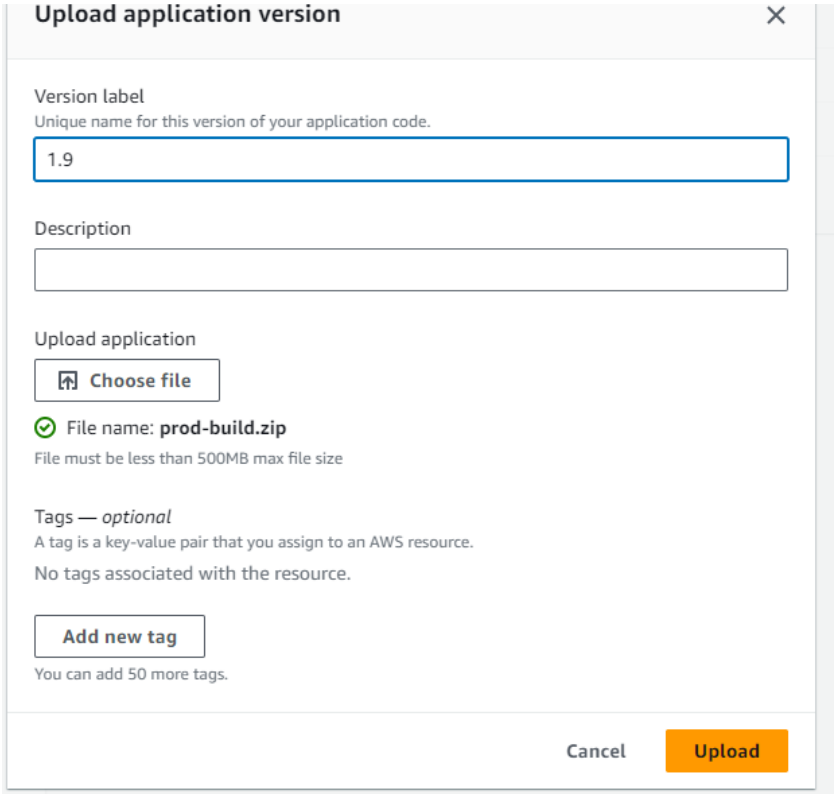


10. Select the application and select the `Actions` dropdown menu. Select `Application versions`

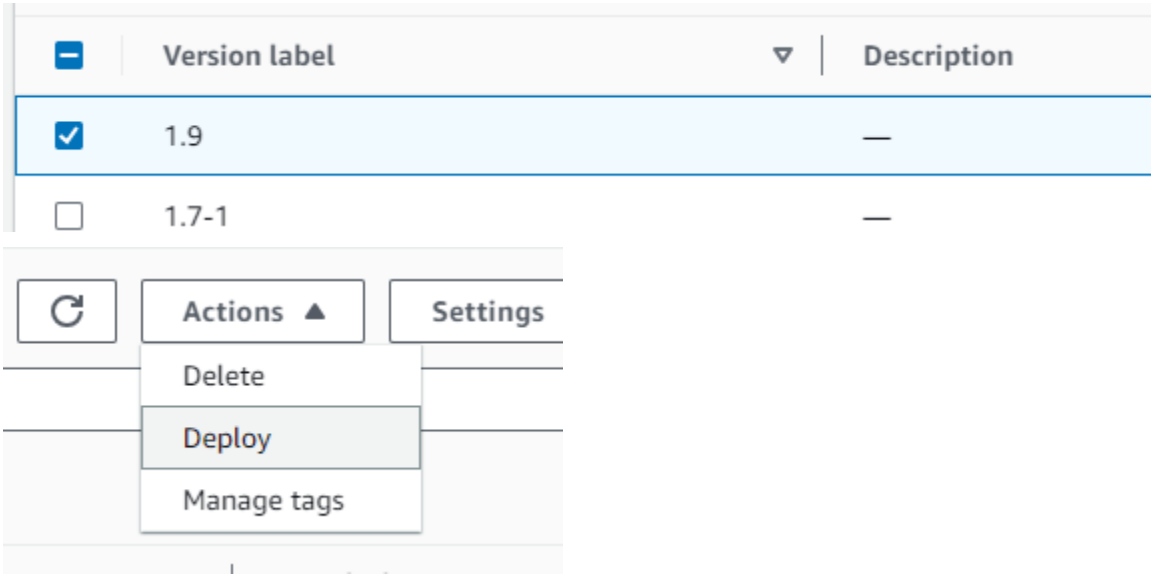




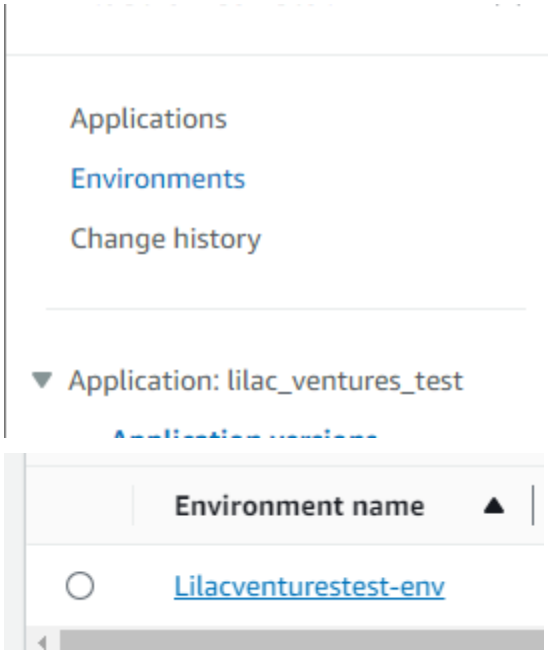
11. In the following screen, select `Upload`, give the new code base an updated version number, choose the zip file from the previous step, and hit upload.



12. In the versions console, select the recently uploaded version and then select `Actions`, and then `Deploy`.



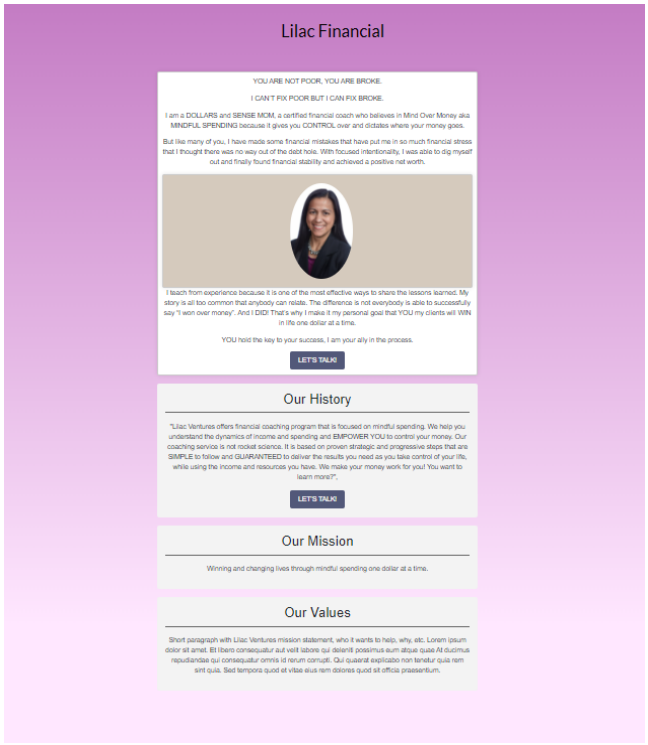
13. Select the currently used environment from the drop down and select deploy. The deployment will take several minutes to complete. To view the status and get updates, go to the Environments console from the left pane and select the current environment


Applications
Environments
Change history
Application: lilac_ventures_test
Environment name
Lilacventurestest-env

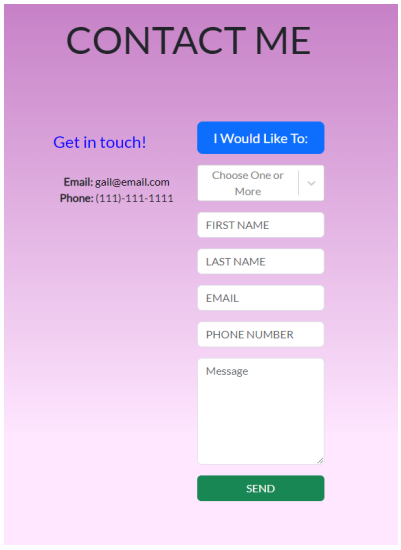
5.0 User interaction

Description: Home Screen
What types of users see the screen (admin, customer, etc): Admin and customer
Related Code Files: HomePage.tsx, HomePageView.tsx, HomePageView.css
Screenshot:

Description: About Screen
What types of users see the screen (admin, customer, etc): Admin and customer
Related Code Files: About.tsx, AboutView.tsx, About.css
Screenshot:

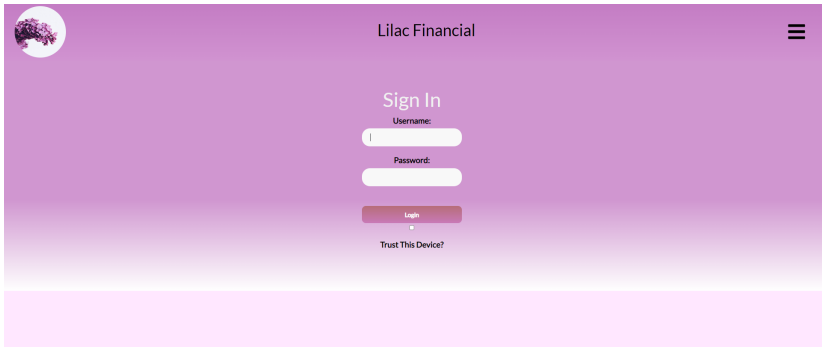


Description: Contact Screen
What types of users see the screen (admin, customer, etc): Customer
Related Code Files: Contact.tsx, ContactView.tsx
Screenshot:

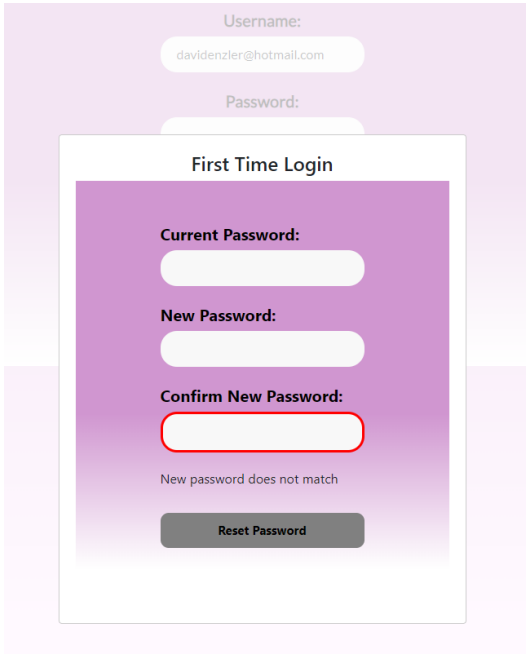


Description: Login Screen
What types of users see the screen (admin, customer, etc): Customer

Related Code Files: LoginButton.tsx, LogoutButton.tsx, Login.tsx, Login.css, AuthProvider.tsx, logoutService.tsx
Screenshot:

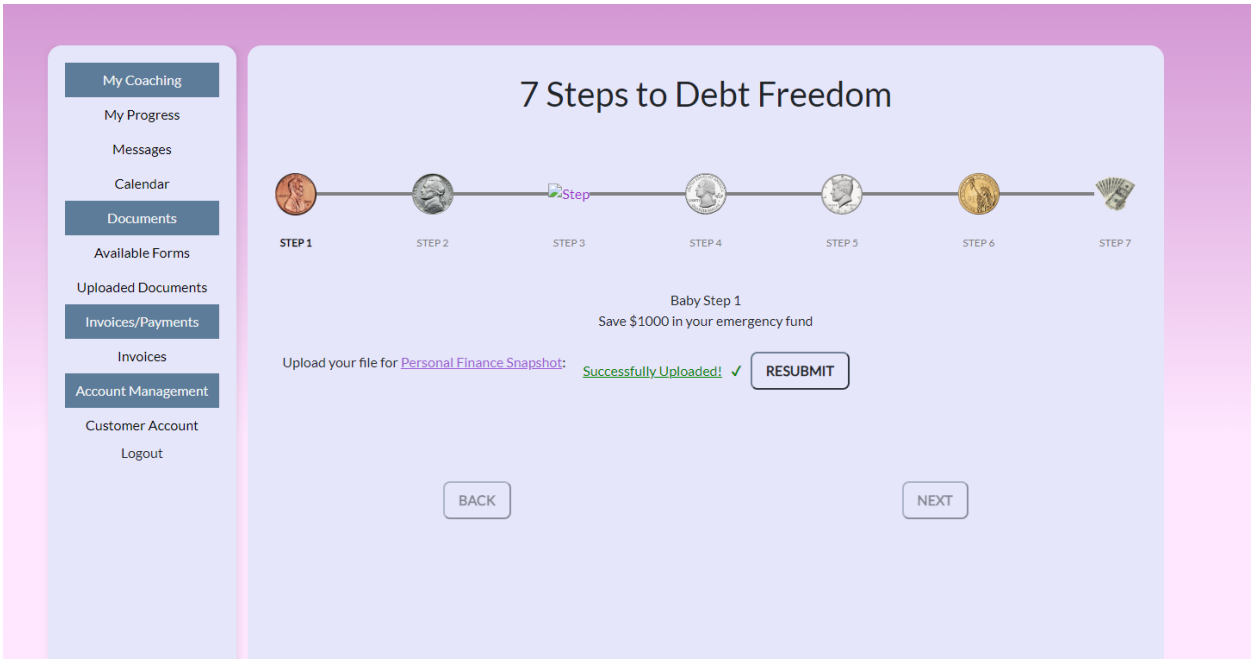


Description: First Time Login Screen
What types of users see the screen (admin, customer, etc): Customer
Related Code Files: FirstTimeLoginForm.css, FirstTimeLoginForm.tsx, FirstTimeLoginModal.tsx
Screenshot:

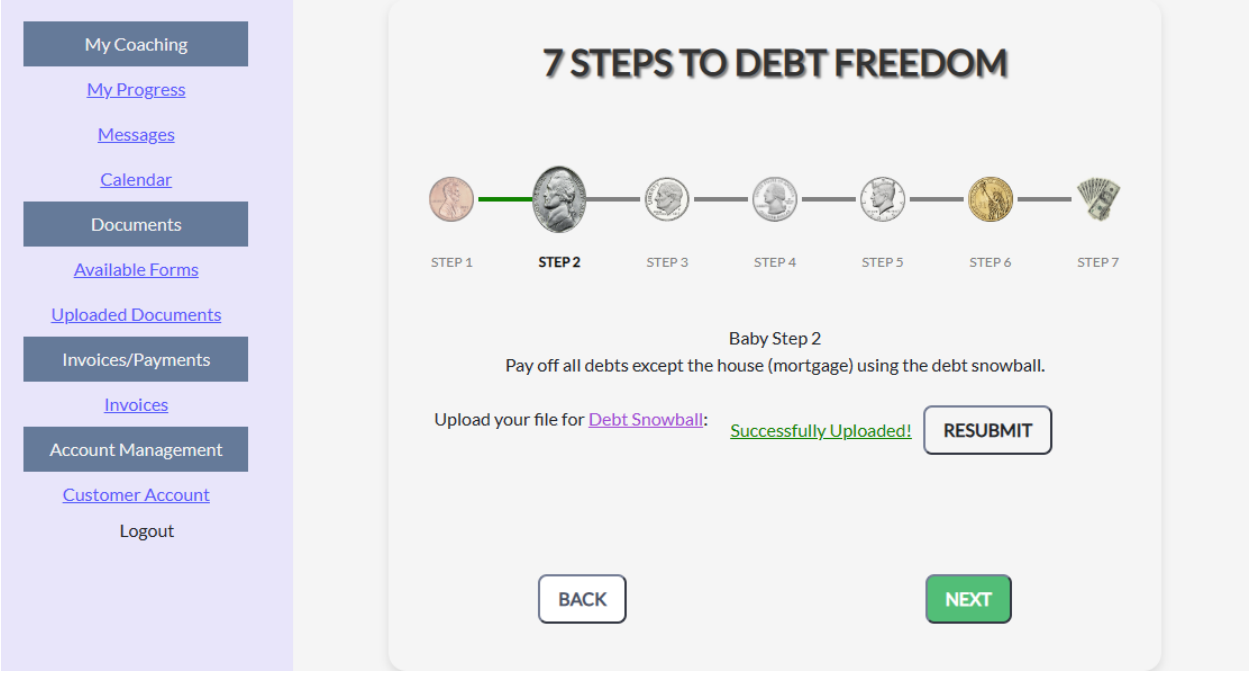


Description: Customer Portal Screen
What types of users see the screen (admin, customer, etc): Customer
Related Code Files: CustomerPortal.tsx, customerPortal.css

Screenshot:

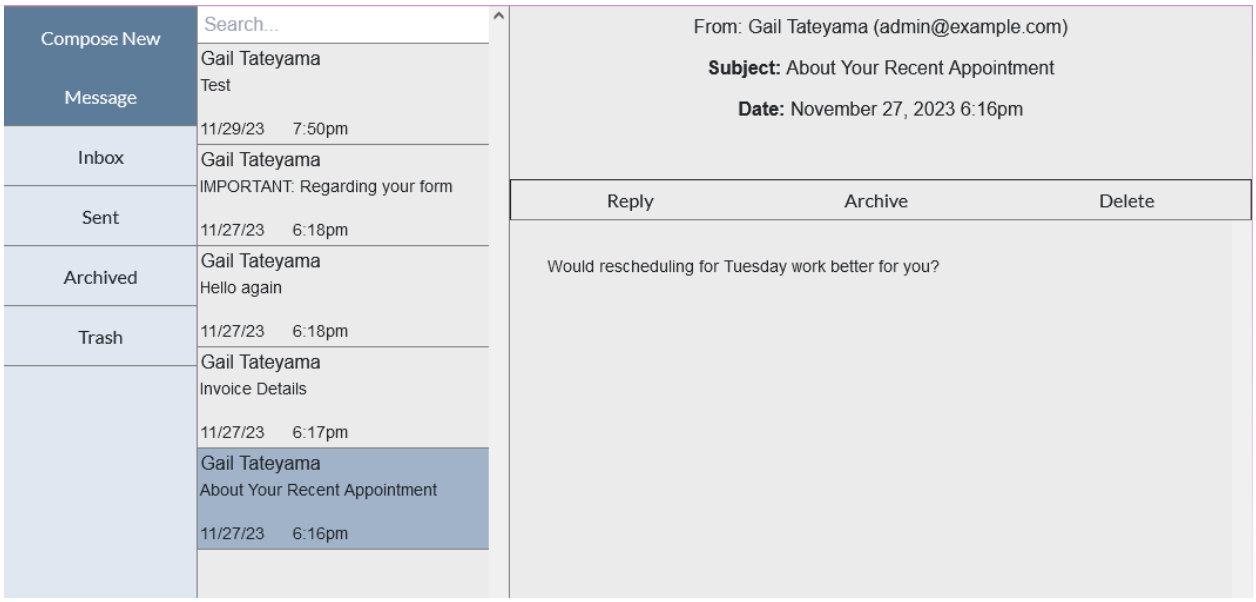


Description: My Progress Screen
What types of users see the screen (admin, customer, etc.): Customer
Related Code Files: ProgressBar.tsx and all files under the folder “progress-bar-components”
Screenshot:

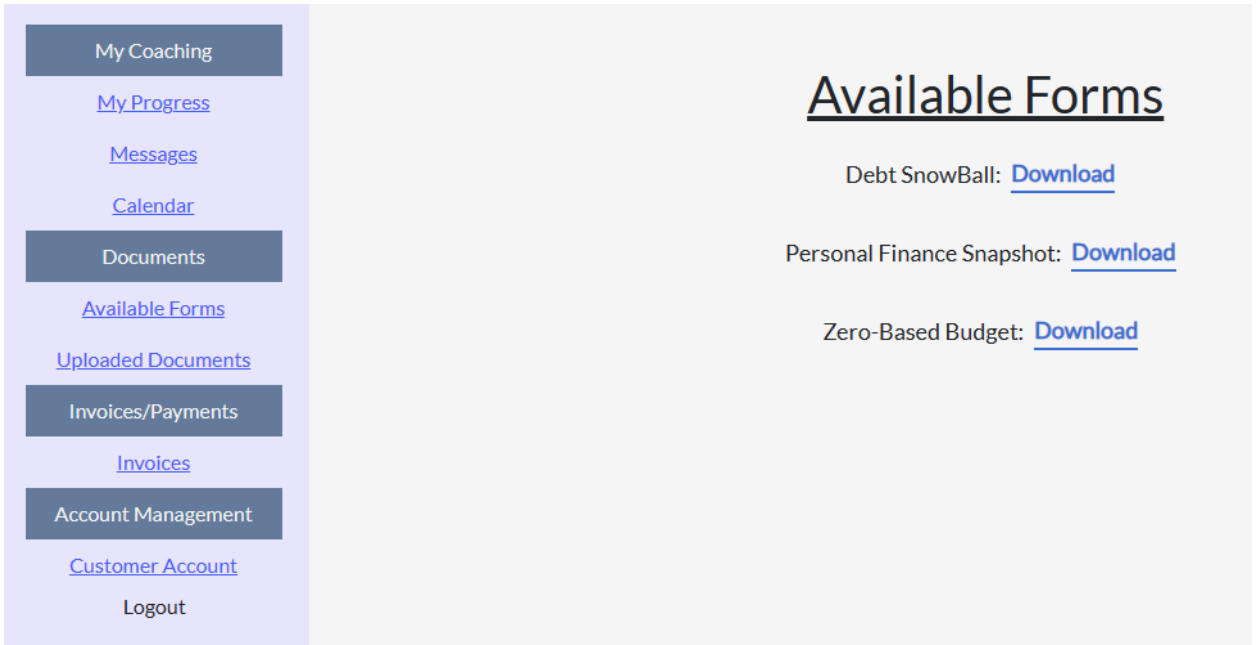


Description: Messages Screen
What types of users see the screen (admin, customer, etc): Customers and admin

Related Code Files: Inbox.tsx, Inbox.css
Screenshot:

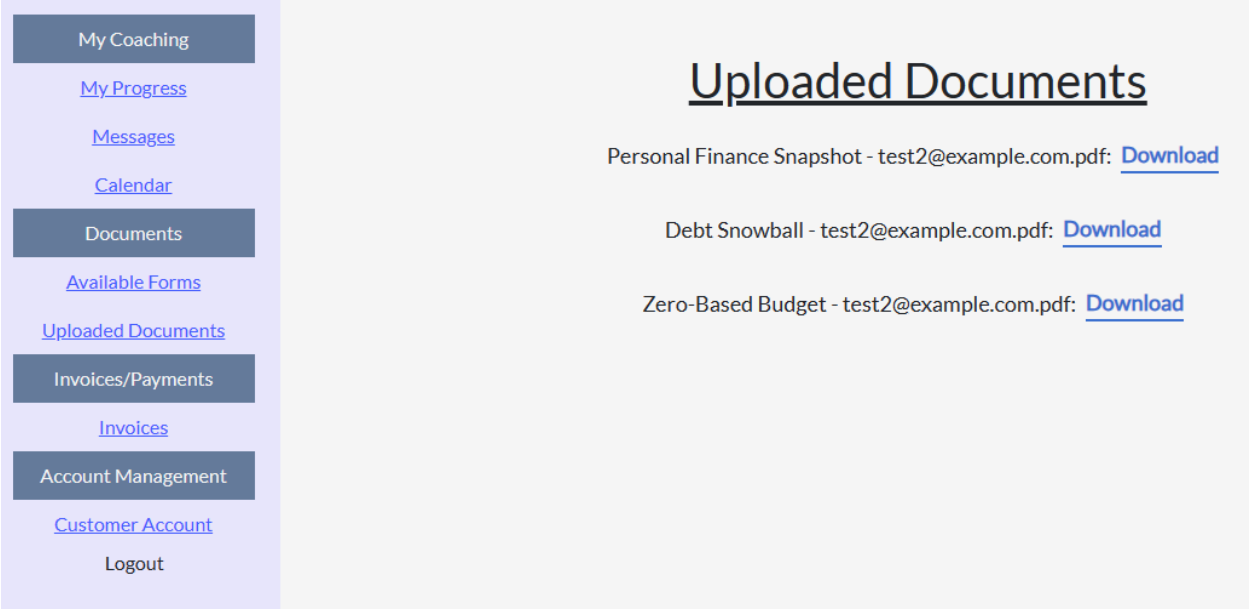


Description: Available Forms Screen
What types of users see the screen (admin, customer, etc): Customer
Related Code Files: AvailableForm.tsx
Screenshot:



Description: Uploaded Documents Screen
What types of users see the screen (admin, customer, etc): Customer
Related Code Files: UploadedDocuments.tsx

Screenshot:



Description: Customer Account Screen
What types of users see the screen (admin, customer, etc): customer
Related Code Files:
Screenshot:

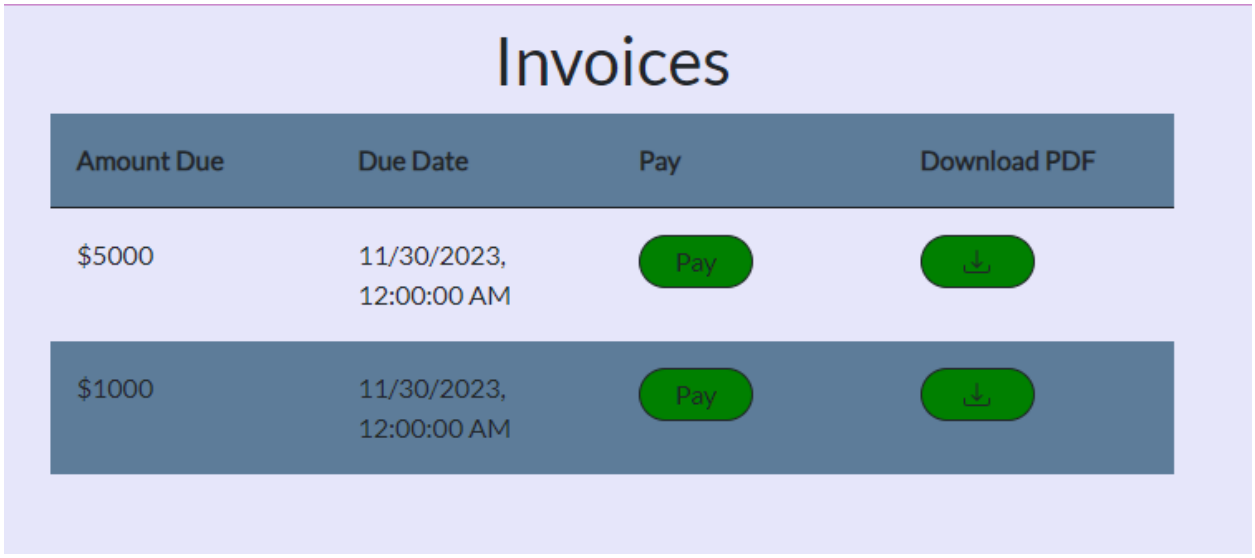
Description: Invoices Screen
What types of users see the screen (admin, customer, etc):

1. Customer

Related Code Files:

1. ./src/CustomerPortalInvoices/CustomerInvoices.css
2. ./src/CustomerPortalInvoices/CustomerInvoiceComponent.tsx
3. ./src/services/invoiceServices.tsx

Screenshot:



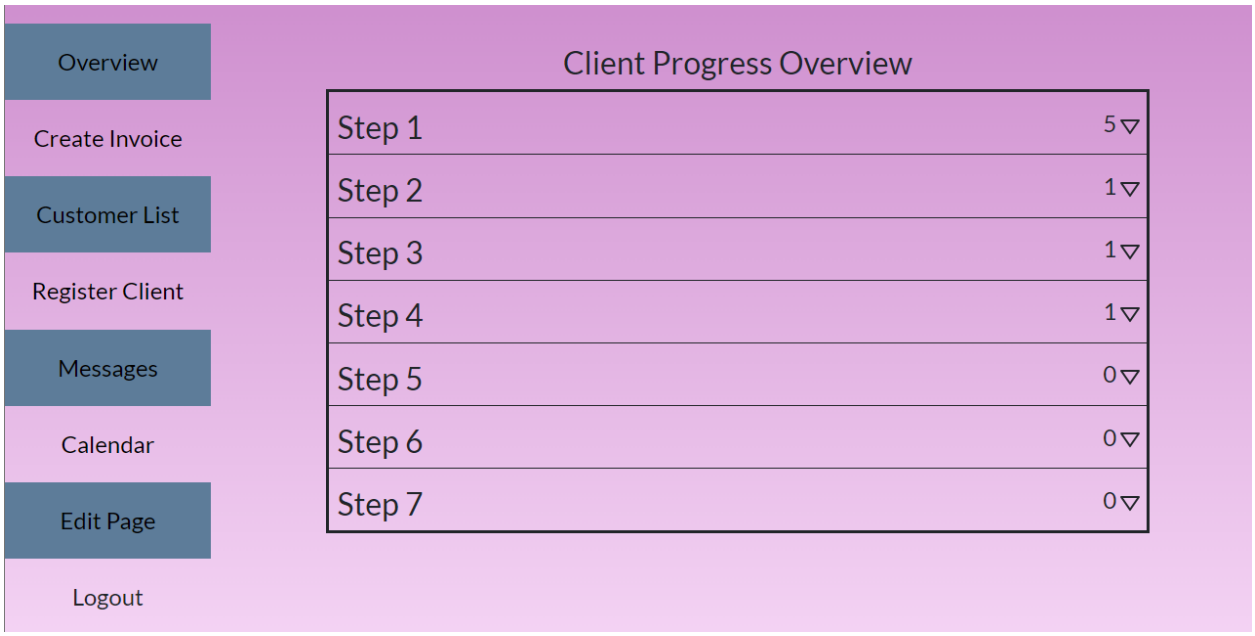
Description: Admin Portal/ Admin Overview Screen
What types of users see the screen (admin, customer, etc):

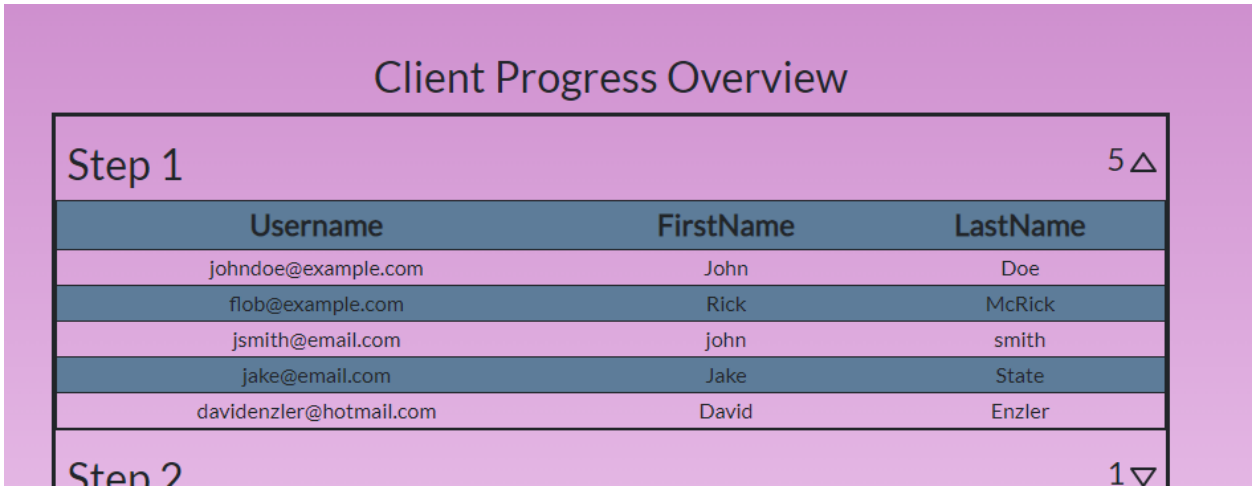
1. Admin

Related Code Files:

1. /src/AdminDashboard/AdminDashboard.tsx
2. /src/AdminDashboard/adminDash.css
3. /src/services/pdfUploadServices.tsx
4. /src/services/ProgressService.tsx

Screenshot:





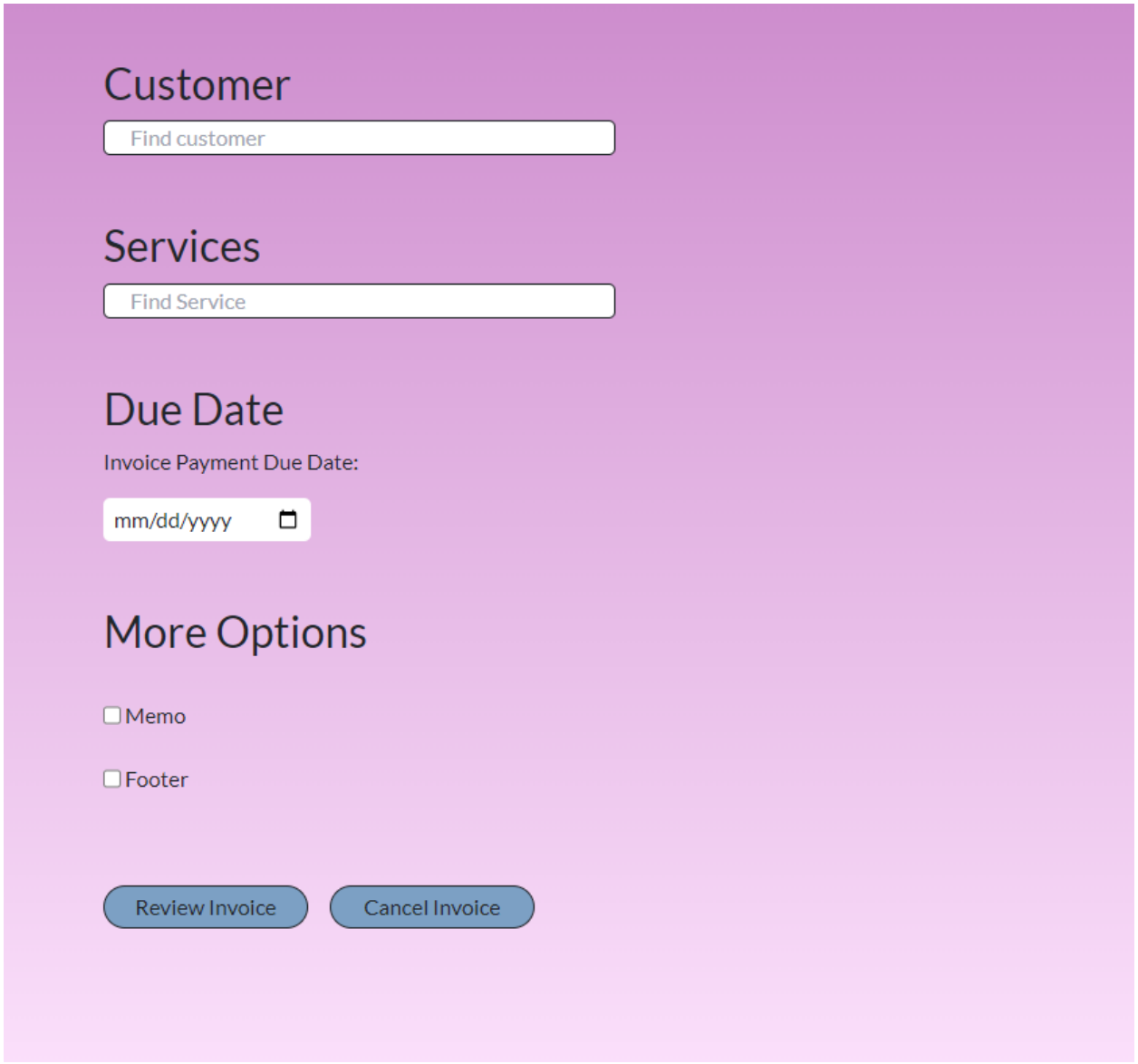
Description: Create Invoice Screen
What types of users see the screen (admin, customer, etc):

1. Admin

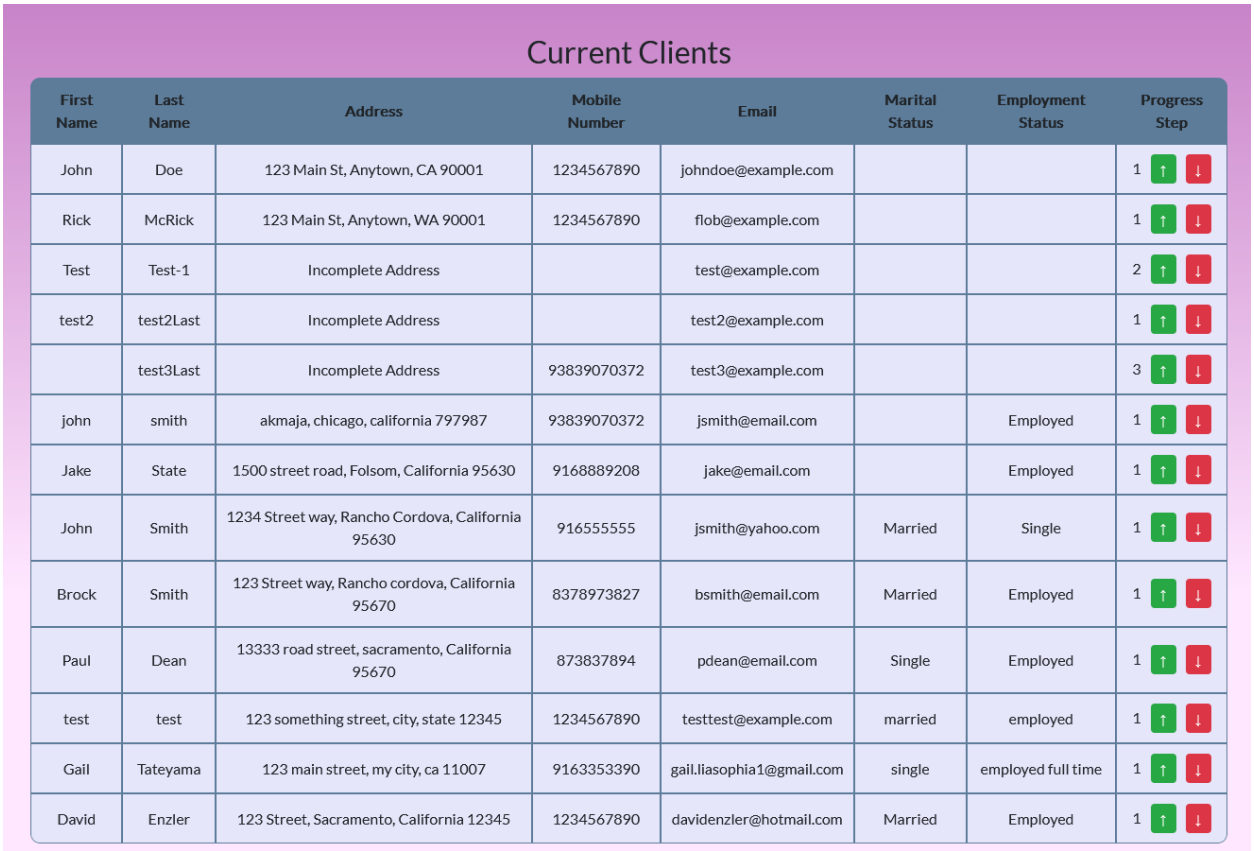
Related Code Files:

1. /src/InvoiceCreation/CreateInvoice.tsx
2. /src/InvoiceCreation/Invoice.css
3. /src/services/customerServices.tsx
4. /src/services/productSearch.tsx
5. /src/services/invoiceServices.tsx

Screenshot:



Description: Customer List Screen
What types of users see the screen (admin, customer, etc): Admin
Related Code Files: /src/CustomerInfoView.tsx
Screenshot:



Current Clients

First Name	Last Name	Address	Mobile Number	Email	Marital Status	Employment Status	Progress Step
John	Doe	123 Main St, Anytown, CA 90001	1234567890	johndoe@example.com			1 ↑ ↓
Rick	McRick	123 Main St, Anytown, WA 90001	1234567890	flob@example.com			1 ↑ ↓
Test	Test-1	Incomplete Address		test@example.com			2 ↑ ↓
test2	test2Last	Incomplete Address		test2@example.com			1 ↑ ↓
	test3Last	Incomplete Address	93839070372	test3@example.com			3 ↑ ↓
john	smith	akmaja, chicago, california 797987	93839070372	jsmith@email.com		Employed	1 ↑ ↓
Jake	State	1500 street road, Folsom, California 95630	9168889208	jake@email.com		Employed	1 ↑ ↓
John	Smith	1234 Street way, Rancho Cordova, California 95630	916555555	jsmith@yahoo.com	Married	Single	1 ↑ ↓
Brock	Smith	123 Street way, Rancho cordova, California 95670	8378973827	bsmith@email.com	Married	Employed	1 ↑ ↓
Paul	Dean	13333 road street, sacramento, California 95670	873837894	pdean@email.com	Single	Employed	1 ↑ ↓
test	test	123 something street, city, state 12345	1234567890	testtest@example.com	married	employed	1 ↑ ↓
Gail	Tateyama	123 main street, my city, ca 11007	9163353390	gail.liasophia1@gmail.com	single	employed full time	1 ↑ ↓
David	Enzler	123 Street, Sacramento, California 12345	1234567890	davidenzler@hotmail.com	Married	Employed	1 ↑ ↓

Description: Register Client Screen

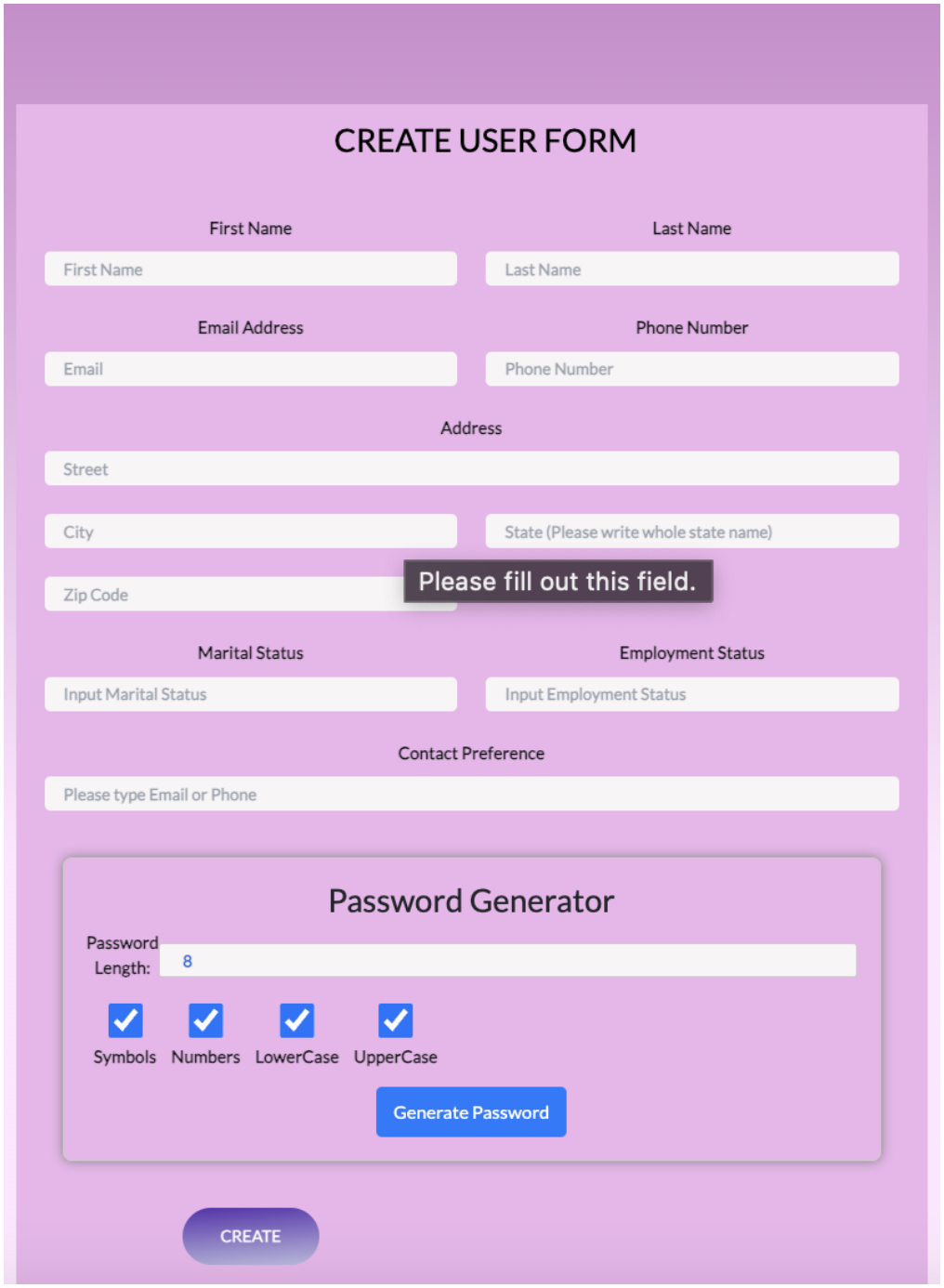
What types of users see the screen (admin, customer, etc):

1. Admin

Related Code Files:

1. /src/RegistrationForm.tsx
2. /src/RegistrationForm.css
3. /src/services/RegisterService.tsx
4. /src/services/ClientService.tsx

Screenshot:



Description: Availability Screen
What types of users see the screen (admin, customer, etc):

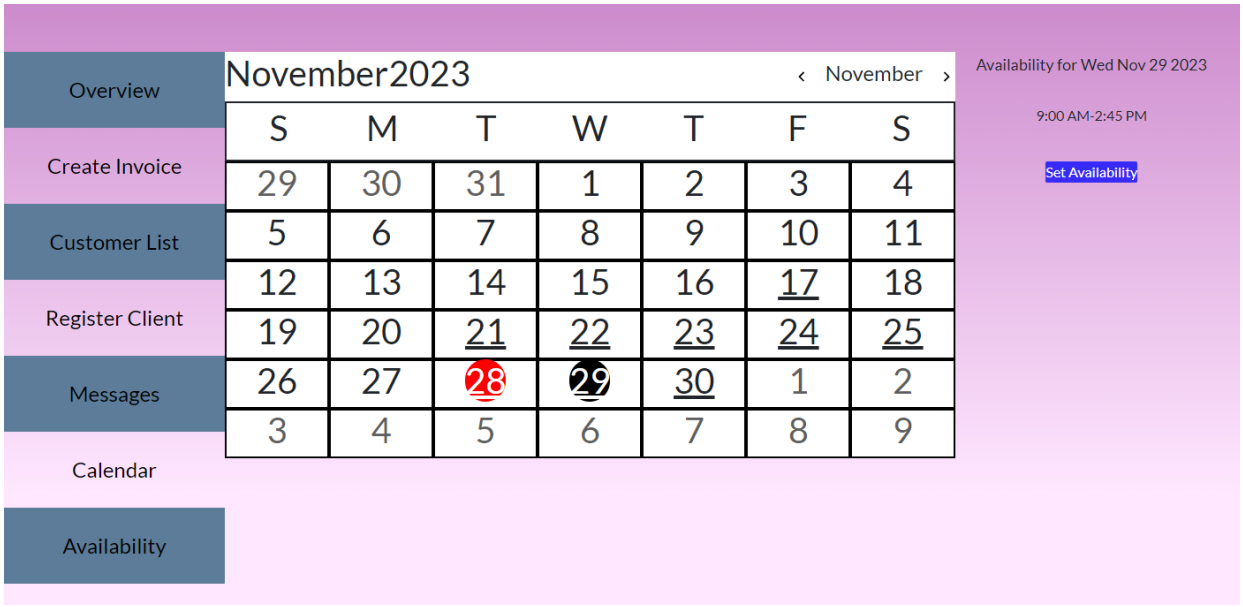
1. Admin

Related Code Files:

1. Front End:
    a. /src/AvailabilityCalendar.tsx
    b. /src/CalendarComponents/calendar.tsx

   c. /src/CalendarComponents/calendar.css
   d. /src/CalendarComponents/cn.tsx
   e. /hooks/useAuth.tsx
2. API Calls:
   a. /api/models/Availability.ts
   b. /api/routes/availability.ts
   c. /api/availabilityController.ts

Screenshot:



Description: Calendar Screen
What types of users see the screen (admin, customer, etc):

1. Admin
2. Customer

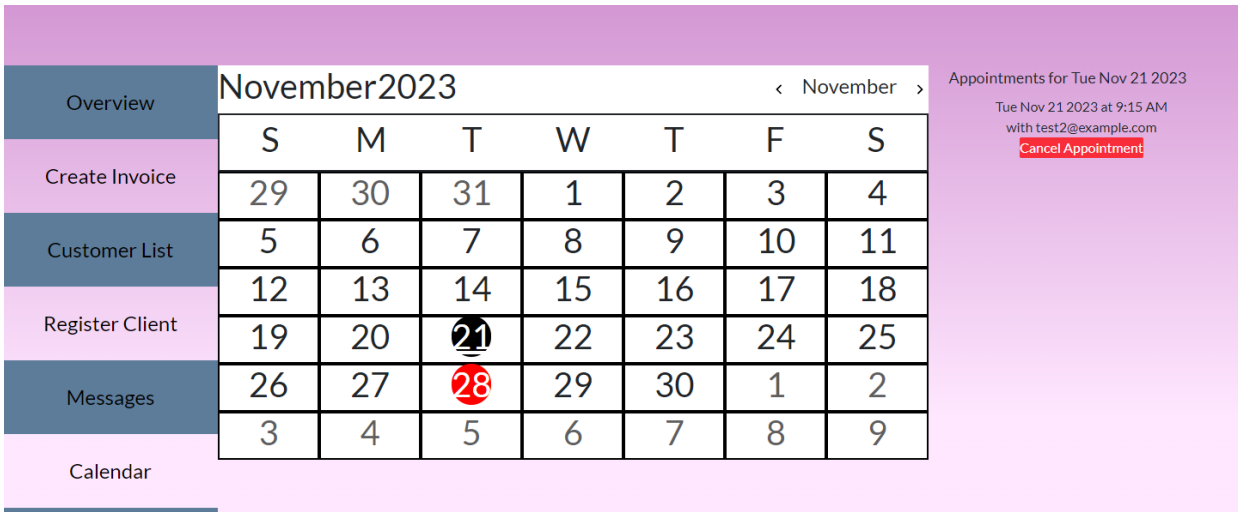
Related Code Files:

1. Front End:
   a. /src/CalendarView.tsx
   b. /src/CalendarComponents/calendar.tsx
   c. /src/CalendarComponents/calendar.css
   d. /src/CalendarComponents/cn.tsx
   e. /hooks/useAuth.tsx
2. API Calls:
   a. /api/models/Appointments.ts
   b. /api/routes/appointments.ts
   c. /api/controllers/appointmentController.ts
   d. /api/models/Availability.ts
   e. /api/routes/availability.ts
   f. /api/availabilityController.ts
   g. /api/models/Message.ts

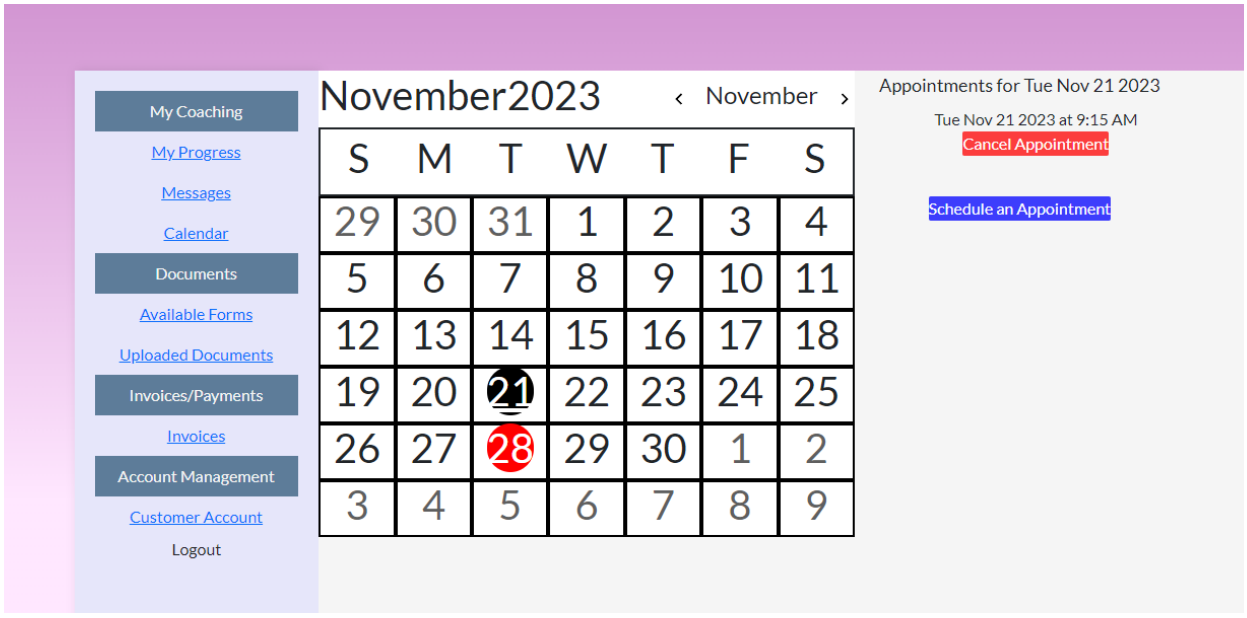
h. /api/routes/messages.ts
i. /api/messagesController.ts

Screenshot:

1. Admin



2. Customer



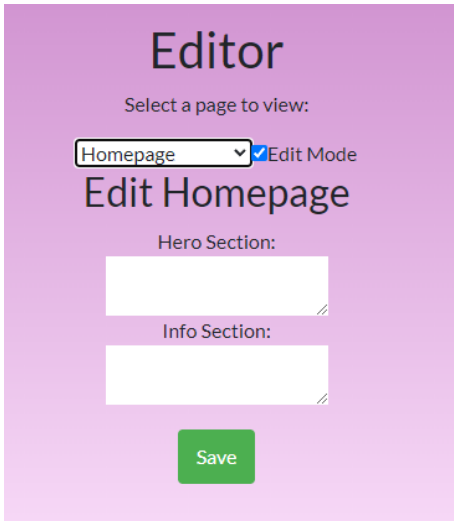
Description: Edit Page Screen
What types of users see the screen (admin, customer, etc): Admin
Related Code Files:

1. Front End:
    a. /src/Edit/AboutEdit.tsx
    b. /src/Edit/ContactEdit.tsx
    c. src/Edit/HomepageEdit.tsx
    d. /src/Edit/style_edit.css
2. API Calls:

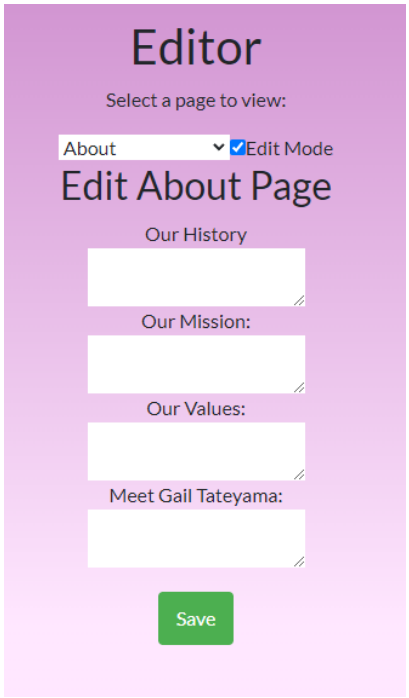
a. /api/controllers/aboutController.ts
b. /api/controllers/contactController.ts
c. /abi/controllers/homeController.ts
d. /api/model/About.ts
e. /api/model/Contact.ts
f. /api/model/Home.ts
g. /api/routes/about.ts
h. /api/routes/contact.ts
i. /api/routes/home.ts

Screenshot:

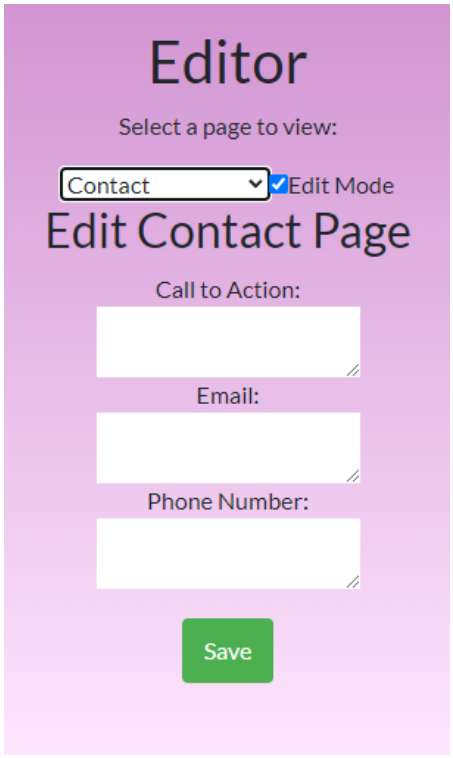
1. HomepageEdit



2. AboutEdit

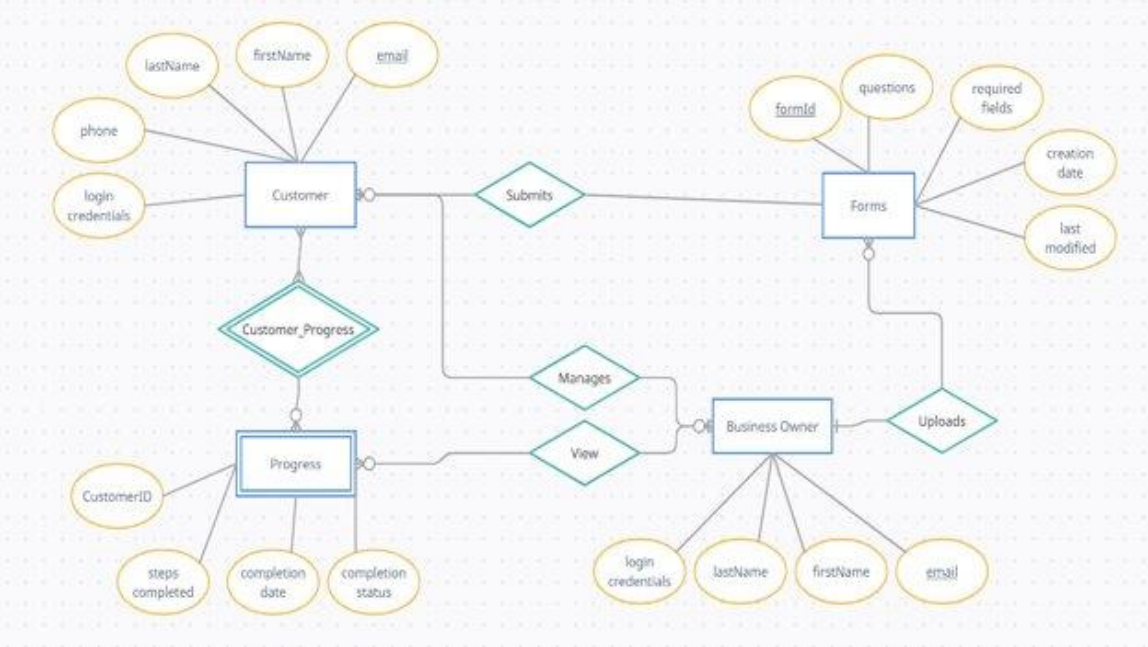


3. ContactEdit



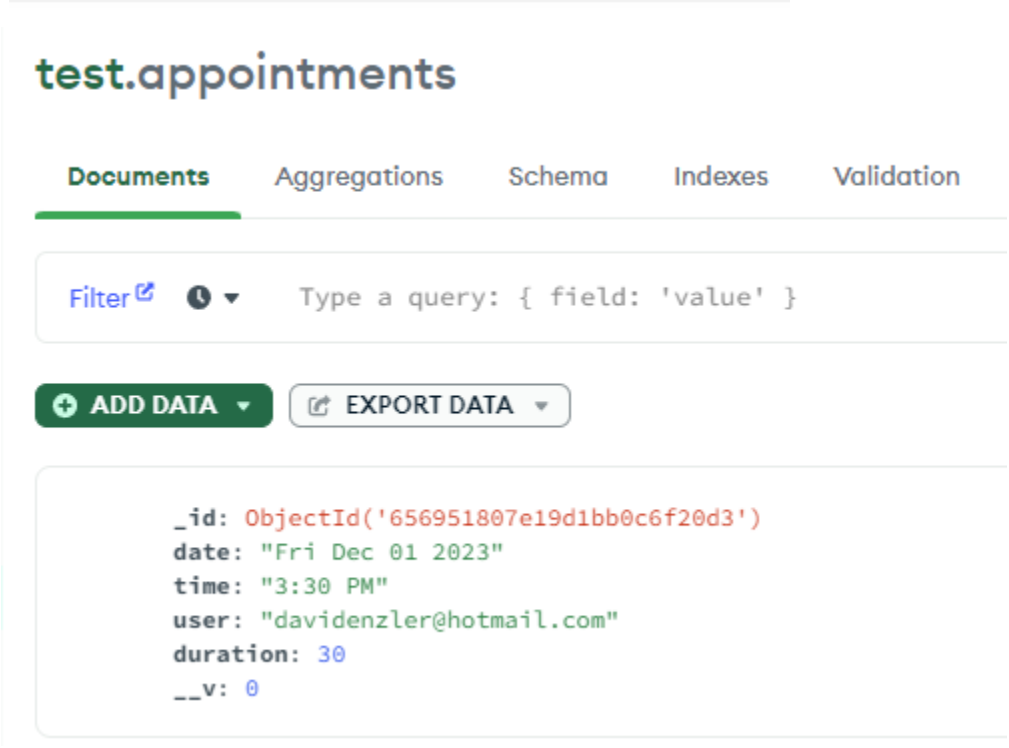
6.0 Database

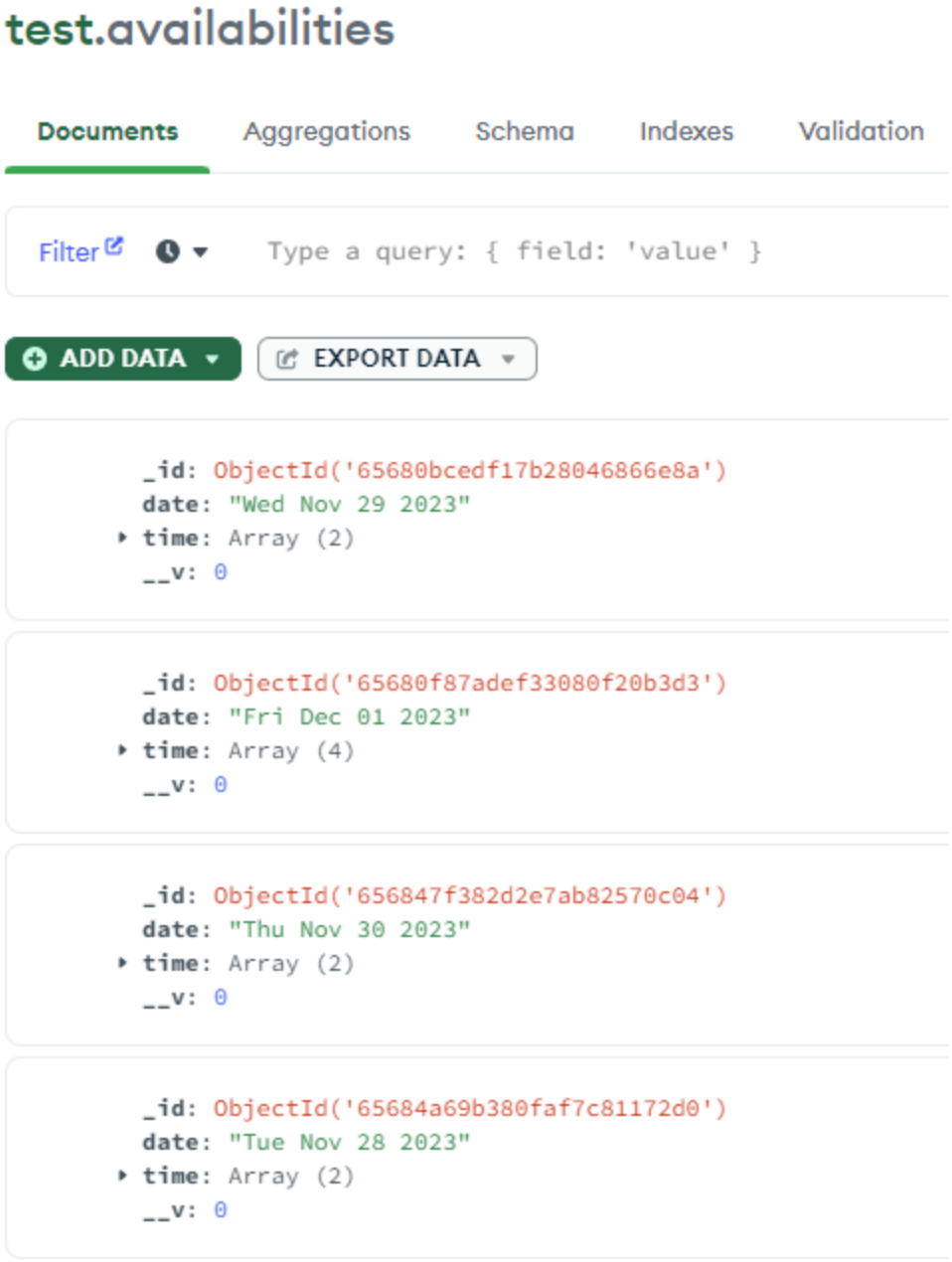
- MongoDB - NoSQL database

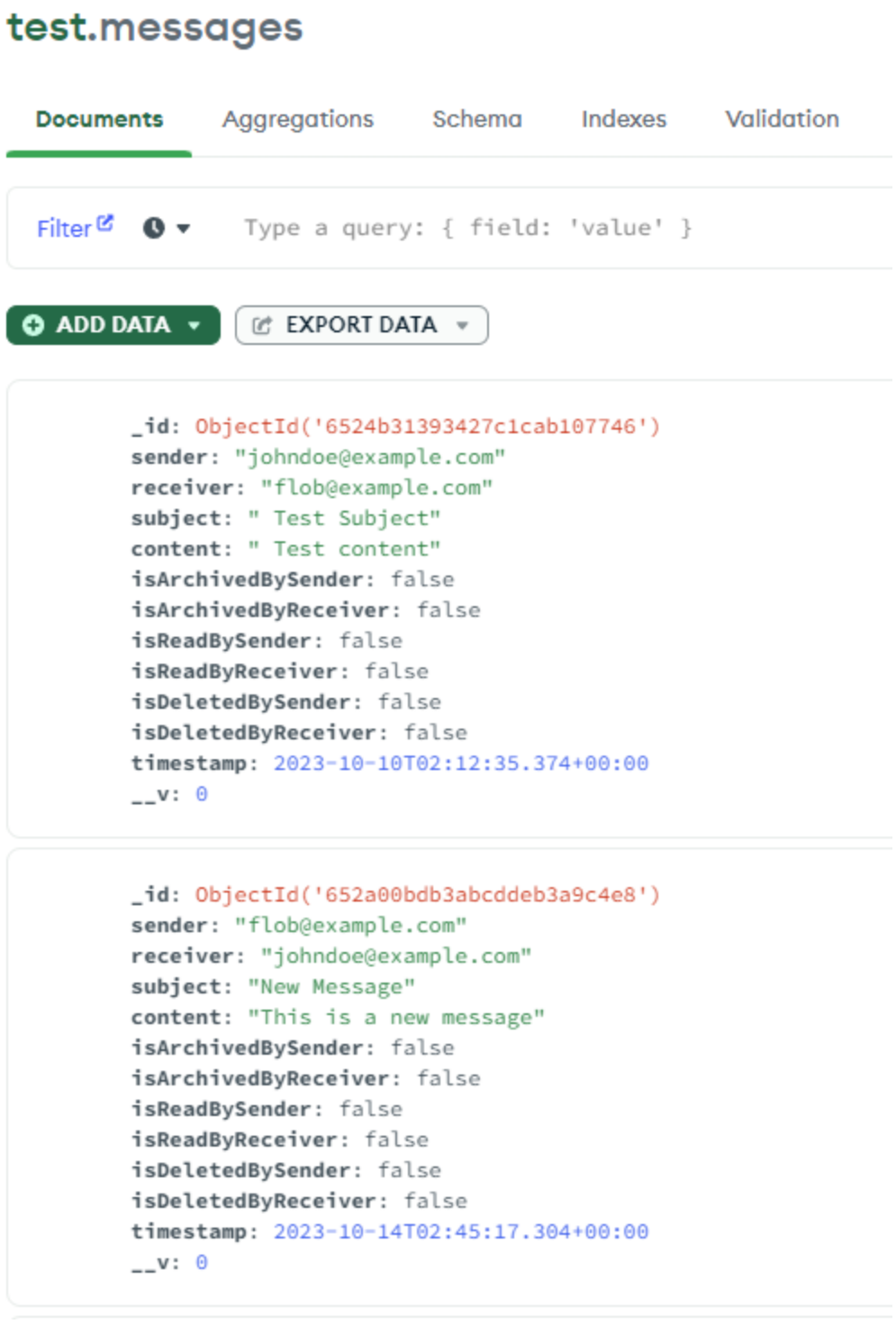


Structure for various database items:

```
_id: ObjectId('655a80637facccdb50bb79bf')
firstName: "john"
lastName: "smith"
email: "jsmith@email.com"
phone: 93839070372
street: "akmaja"
city: "chicago"
state: "california"
zip: 797987
employment: "Employed"
cPreference: "Email"
__v: 0
```


test.availabilities
Documents
Aggregations
Schema
Indexes
Validation
Filter
Type a query: { field: 'value' }
ADD DATA
EXPORT DATA
_id: ObjectId('65680bcedf17b28046866e8a')
date: "Wed Nov 29 2023"
time: Array (2)
__v: 0
_id: ObjectId('65680f87adef33080f20b3d3')
date: "Fri Dec 01 2023"
time: Array (4)
__v: 0
_id: ObjectId('656847f382d2e7ab82570c04')
date: "Thu Nov 30 2023"
time: Array (2)
__v: 0
_id: ObjectId('65684a69b380faf7c81172d0')
date: "Tue Nov 28 2023"
time: Array (2)
__v: 0


test.messages
Documents
Aggregations
Schema
Indexes
Validation
Filter
Type a query: { field: 'value' }
ADD DATA
EXPORT DATA
_id: ObjectId('6524b31393427c1cab107746')
sender: "johndoe@example.com"
receiver: "flob@example.com"
subject: " Test Subject"
content: " Test content"
isArchivedBySender: false
isArchivedByReceiver: false
isReadBySender: false
isReadByReceiver: false
isDeletedBySender: false
isDeletedByReceiver: false
timestamp: 2023-10-10T02:12:35.374+00:00
__v: 0
_id: ObjectId('652a00bdb3abcddeb3a9c4e8')
sender: "flob@example.com"
receiver: "johndoe@example.com"
subject: "New Message"
content: "This is a new message"
isArchivedBySender: false
isArchivedByReceiver: false
isReadBySender: false
isReadByReceiver: false
isDeletedBySender: false
isDeletedByReceiver: false
timestamp: 2023-10-14T02:45:17.304+00:00
__v: 0

test.pdfs

Documents Aggregations Schema Indexes Validation

Filter Type a query: { field: 'value' }

ADD DATA EXPORT DATA

```
_id: ObjectId('6521d7f41102d760313d7dd7')
fileId: ObjectId('6521d7f41102d760313d7dd5')
name: "Form A - JohnDoe.pdf"
seenByAdmin: false
__v: 0
```

- Viewer and Editor relationship