

CSC 648 - MILESTONE 4- TESTING

TEAM 05- SECTION-04

DATE- 05/03/2023

TRAVELMATE - GPT API/MERN STACK TRAVEL PLANNING WEBSITE

TEAM-

MUHAMMED NAFEES- FRONTEND LEAD

YAHYA OBEID - BACKEND LEAD

DANIA - SCRUM-MASTER

HANEI

FARAH JAMJOUR

SIDDHARTH SANKAR - TEAM LEAD

1) QA TESTING

For implementing test cases we have chosen 5 P1 features which are - login, signup. Database connection, gpt api and save plans. All the test cases are done in the backend by checking the user model and the established routes for each function in the routes folder.

Test cases directory containing all the test files -

<https://github.com/CSC-648-SFSU/csc648-01-csc648-04-team-05/tree/master/application/backend/tests>

SIGNUP -

Allows the new users that are visiting travelmate to create an account for themselves. This account will be used by them to save their favourite plans to their profile for future reference. To signup, users will need to provide their first and last names, email id and a strong password.

Frontend-

<https://github.com/CSC-648-SFSU/csc648-01-csc648-04-team-05/blob/master/application/frontend/src/pages/SignupPage.js>

Backend-

<https://github.com/CSC-648-SFSU/csc648-01-csc648-04-team-05/blob/master/application/backend/routes/users.js>

Testcase- the test case covers both successful and failure scenarios. The first testcase expects the user to successfully login by providing the correct details while the second test case expects the signup to fail by providing a weak password/invalid email.

```
user.test.js x login.test.js package.json ...\frontend JS ExplorePage.js JS api.js api.test.js JS server
application > backend > tests > user.test.js > describe("POST /api/users") callback
12 test("should create a new user and return 201 status", async () => {
13   const newUser = {
14     firstName: "John",
15     lastName: "Doe3",
16     email: "johndoe3@example.com",
17     password: "Password123!",
18   };
19
20   const res = await request(app).post("/api/users").send(newUser);
21
22   if (res.status !== 201) {
23     console.log("Error message:", res.body.message);
24   }
25
26   expect(res.status).toBe(201);
27   expect(res.body).toHaveProperty("_id");
28   expect(res.body).toMatchObject({
29     firstName: "John",
30     lastName: "Doe3",
31     email: "johndoe3@example.com",
32   });
33 });
34
35 test("should fail to create a new user with an invalid email and return 400 status", async () => {
36   const newUser = {
37     firstName: "Jane",
38     lastName: "Doe",
39     email: "janedoe.invalidemail", // Invalid email address
40     password: "Password123!",
41   };
42
43   const res = await request(app).post("/api/users").send(newUser);
44
45   if (res.status !== 400) {
46     console.log("Error message:", res.body.message);
47   }
48
49   expect(res.status).toBe(400);
50 }
```

PS C:\Users\siddh\Desktop\csc648-01-csc648-04-team-05\application\backend> npx jest user.test.js

>>

```
console.log
  Listening on port 5000...
```

```
    at Object.log (server.js:26:26)
```

```
console.log
  Connected to database successfully
```

```
    at log (db.js:13:13)
```

PASS tests/user.test.js (7.282 s)

POST /api/users

✓ should create a new user and return 201 status (1678 ms)

✓ should fail to create a new user with an invalid email and return 400 status (97 ms)

Test Suites: 1 passed, 1 total

Tests: 2 passed, 2 total

Snapshots: 0 total

Time: 7.524 s, estimated 8 s

Ran all test suites matching /user.test.js/i.

LOGIN- this function allows the new/returning users to access the features of travelmate by entering the details of their registered account. If the details are valid and stored in the database, user will be allowed to enter.

Frontend-

<https://github.com/CSC-648-SFSU/csc648-01-csc648-04-team-05/blob/master/application/frontend/src/pages/LoginPage.js>

Backend-

<https://github.com/CSC-648-SFSU/csc648-01-csc648-04-team-05/blob/master/application/backend/routes/auth.js>

test case-

It covers both successful and unsuccessful cases by having 2 test cases in login.test.js. The first test enters the details of a registered user and it passes. The second test also passes by failing to login as it expects because the test will willingly enter wrong details to test the authentication.

```
test("should login a user and return 200 status", async () => {
  const loginUser = {
    email: "johndoe3@example.com",
    password: "Password123!",
  };

  const res = await request(app).post("/api/auth").send(loginUser);

  expect(res.status).toBe(200);
  expect(res.body).toHaveProperty("data");
  expect(res.body.message).toBe("logged in successfully");
});

test("should fail to login a user with incorrect password and return 401 status", async () => {
  const loginUser = {
    email: "johndoe3@example.com",
    password: "WrongPassword123!", // Incorrect password
  };

  const res = await request(app).post("/api/auth").send(loginUser);

  expect(res.status).toBe(401);
  expect(res.body).toHaveProperty("message");
  expect(res.body.message).toBe("Invalid Email or Password");
});
```

```
PASS tests/login.test.js (7.92 s)
  POST /api/auth
    ✓ should login a user and return 200 status (1607 ms)
    ✓ should fail to login a user with incorrect password and return 401 status (900 ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        8.248 s
Ran all test suites matching /login.test.js/i.
```

API-

This is the heart of our application as it is the feature that travelmate is built around. It is an application programming interface provided by Openai which uses the prompt given by the user to generate a plan according to the given specifications. The prompt is built by using multiple if statements to check if the user has added any additional specification and is dynamic as it can handle any number of them.

Backend api file-

<https://github.com/CSC-648-SFSU/csc648-01-csc648-04-team-05/blob/master/application/backend/services/api.js>

Test case-

It covers two scenarios when the prompt is given and when the prompt field is empty. When the prompt is given, the test case checks if the api is implemented correctly and a response is returned from the server to the client.

When a prompt is not given, the response will not be generated and we can make sure that responses are not generated on empty prompts.

```
PASS tests/api.test.js (5.066 s)

GET /api/completion

  ✓ should return a completion from the OpenAI API (92 ms)
  ✓ should return a 400 error if prompt is missing (83 ms)
  ✓ should return a 500 error if an error occurs during completion creation (22 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        6.421 s
Ran all test suites matching /api.test.js/i.
```

```
describe("GET /api/completion", () => {
  it("should return a completion from the OpenAI API", async () => {
    const mockResponse = { result: "success" };

    // Mock the createCompletion function
    createCompletion.mockImplementation(() => Promise.resolve(mockResponse));

    const prompt = "test prompt";

    const response = await request(app).get(`/api/completion?prompt=${prompt}`);

    expect(response.status).toBe(200);
    expect(response.body).toEqual(mockResponse);
  });

  it("should return a 400 error if prompt is missing", async () => {
    const response = await request(app).get("/api/completion");

    expect(response.status).toBe(400);
    expect(response.body).toEqual({ error: "Prompt is required" });
  });

  it("should return a 500 error if an error occurs during completion creation", async () => {
    const errorMessage = "Error creating completion";
    createCompletion.mockImplementation(() =>
      Promise.reject(new Error(errorMessage))
    );

    const prompt = "test prompt";

    const response = await request(app).get(`/api/completion?prompt=${prompt}`);

    expect(response.status).toBe(500);
    expect(response.body).toEqual({ error: "Internal server error" });
  });
});
```

Database-

The mongodb database is used to store all the user data along with the saved plans associated with each user id. It is essential to make the app function as everything other than prompt generation refers to the backend database.

Database definition -

<https://github.com/CSC-648-SFSU/csc648-01-csc648-04-team-05/blob/master/application/backend/models/User.js>

Test case -

The test case check whether the mongodb database is connected to the backend server everytime it starts. This is essential as the database should be connected before the user tried to login/signup to prevent the app from crashing.

```
application > backend > tests > db.test.js > ...
1  const mongoose = require("mongoose");
2  const connection = require("../db");
3
4  describe("database connection", () => {
5    it("should connect to the database", async () => {
6      await connection();
7      expect(mongoose.connection.readyState).toBe(1);
8    });
9  });
10
```

```
PASS tests/db.test.js
  database connection
    ✓ should connect to the database (810 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        3.757 s, estimated 4 s
Ran all test suites matching /db.test.js/i.
```

2. Integration testing-

Integration testing excel sheet-

A	B	C	D	E	F	G
1	Test Case ID	TM_001	Test Case Description	Test the login page on first entering the website		
2	Created By	Siddharth	Reviewed By	Muhammed		
3	Version	1				
4						
5	QA tester's log	Check the login loading up on every start of the website				
6						
7	Tester's Name	Siddharth	Date Tested	1-May-23	Test Case (Pass/Fail)	Pass
8						
9	NO.	Prerequisites	No.	Test Data		
10	1	Access to chrome browser	1	localhost:3000 (website link)		
11						
12						
13						
14	Test Scenario	Verify that login page has no errors while loading				
15						
16	Step#	Step Details	Expected Results	Actual Results	Pass/Fail/Not Executed	
17	1	Start the server using npm start	Console loads up nodemon to start frontend/backend	As expected	Pass	
18	2	Enter website url(local host)	Loads up the login page as user isn't logged in	As expected	Pass	
19						
20						
21						
22						
23	Test Case ID	TM_002	Test Case Description	Try to login without signup		
24	Created By	Muhammed	Reviewed By	Siddharth		
25	Version	1				
26						
27	QA tester's log	User shouldn't be able to login without signing up/creating an account				
28						
29	Tester's Name	Muhammed	Date Tested	1-May-25	Test Case (Pass/Fail)	Pass
30						
31	NO.	Prerequisites	No.	Test Data		
32	1	Access to chrome browser	1	Test_user		
33			2	123456		
34						
35						
36	Test Scenario	Verify that the user cannot enter the website without creating an account				
37						
38	Step#	Step Details	Expected Results	Actual Results	Pass/Fail/Not Executed	
39	1	Navigate to localhost:3000/	Site opens	As expected	Pass	
40	2	Try to login with invalid details	"Invalid email or password"	As expected	Pass	
41						
42						
43						
	Test Case ID	TM_003	Test Case Description	signup with invalid email/pass		
	Created By	Muhammed	Reviewed By	Siddharth		
	Version	1.1				
	QA tester's log	Signup shouldn't allow the user to create account without entering valid email and strong password				
	Tester's Name	Muhammed	Date Tested	1-May-25	Test Case (Pass/Fail)	Pass
	NO.	Prerequisites	No.	Test Data		
	1	Access to chrome browser	1	email= siddharth		
				password= 123		
	Test Scenario	Verify that signup has strong authentication				
	Step#	Step Details	Expected Results	Actual Results	Pass/Fail/Not Executed	
	1	Navigate to signuo page	signup input fields appear	as expected	pass	
	2	enter invalid details	red error message appear telling to input valid email	as expected	pass	

Test Case ID	TM_004	Test Case Description	Signup with valid details		
Created By	Muhammed	Reviewed By	siddharth		
Version	1.1				
QA tester's log	User must be navigated to homepage on successful signup				
Tester's Name	Muhammed	Date Tested	1-May-23	Test Case (Pass/Fail)	pass
NO.	Prerequisites	No.	Test Data		
1	Access to browser		1 email = sid@gmail.com		
2	Details of email and pass		2 pass= Siddharth@23		
Test Scenario	User must go to homepage				
Step#	Step Details	Expected Results	Actual Results	Pass/Fail/Not Executed	
1	Enter email and password	Input fields should fill up	as expected	pass	
2	click on login	User redirected to homepage	as expected	pass	
Test Case ID	TM_005	Test Case Description	check navbar functionality		
Created By	Siddharth	Reviewed By	Muhammed		
Version	1.2				
QA tester's log	Check if the navigate button navigate to all the different pages that are mentioned on the buttons				
Tester's Name	Siddharth	Date Tested	2-May-23	Test Case (Pass/Fail)	Pass
NO.	Prerequisites	No.	Test Data		
1	Access to browser				
2	access to registered account				
Test Scenario	Redirect to all pages				
Step#	Step Details	Expected Results	Actual Results	Pass/Fail/Not Executed	
1	click on travelmate logo	Goes to homepage	as expected	pass	
2	Click on explore	Goes to explore page	as expected	PASS	
3	click on profile	goes to profile page	as expected	pass	
Test Case ID	TM_006	Test Case Description	Go to profile page to see user details		
Created By	Siddharth	Reviewed By	Muhammed		
Version	1.2				
QA tester's log	User name must be displayed on profile				
Tester's Name	Siddharth	Date Tested	2-May-23	Test Case (Pass/Fail)	pass
NO.	Prerequisites	No.	Test Data		
1	Access to browser				
2	regstered account				
Test Scenario	displays firstname and last name				
Step#	Step Details	Expected Results	Actual Results	Pass/Fail/Not Executed	
1	login	goes to homepage	as expected	pass	
2	Click on profile	displays username on profile page	as expected	pass	

0	Test Case ID	TM_007	Test Case Description	Try to see saved plans in profile		
1	Created By	Siddharth	Reviewed By	Muhammed		
2	Version	1.5				
3						
4	QA tester's log	Try to see the saved plans on the users profile page				
5						
6	Tester's Name	siddharth	Date Tested	2-May-23	Test Case (Pass/Fail)	Fail
7						
8	NO.	Prerequisites	No.	Test Data		
9	1	access to browser		email= sid@gmail.com		
0	2	registered account		pass= Siddharth@23		
1						
2						
3	Test Scenario	Displays a list of the saved plans				
4						
5	Step#	Step Details	Exprected Results	Actual Results	Pass/Fail/Not Executed	
6	1	click on the profile page	goes to profile page	as expected	pass	
7	2	scroll down	must see list of saved plans under the saved plans hea	not ass expected	fail	
8						
9						
n						
2	Test Case ID	TM_008	Test Case Description	click on explore page to see random destinations		
3	Created By	Siddharth	Reviewed By	Muhammed		
4	Version	1.5				
5						
6	QA tester's log	Explore page must display all the randomly generated destination with their photos				
7						
8	Tester's Name	Siddharth	Date Tested	3-May-23	Test Case (Pass/Fail)	Pass
9						
0	NO.	Prerequisites	No.	Test Data		
1	1	access to browser		email= sid@gmail.com		
2	2	registered account		pass= Siddharth@23		
3						
4						
5	Test Scenario	Explore page displays 5 random destinations				
6						
7	Step#	Step Details	Exprected Results	Actual Results	Pass/Fail/Not Executed	
8	1	click on explore	5 random destinations are displayed	as expected	pass	
9						
0						
1						
2						
4	Test Case ID	TM_009	Test Case Description	click on random destination to generate plan		
5	Created By	Siddharth	Reviewed By	Muhammed		
5	Version	1.1				
7						
8	QA tester's log	The buttons must generate a random plan using the api in the plan page				
9						
0	Tester's Name	Siddharth	Date Tested	3-May-23	Test Case (Pass/Fail)	pass
1						
2	NO.	Prerequisites	No.	Test Data		
3	1	access to browser		email= sid@gmail.com		
4	2	registered account		pass= Siddharth@23		
5						
6						
7	Test Scenario	Generates random plan				
8						
9	Step#	Step Details	Exprected Results	Actual Results	Pass/Fail/Not Executed	
0	1	click on explore	random destinations displayed	as expected	pass	
1	2	click on a random destination	redirects to plan page with the generated plan	as expected	pass	
2						
3						
4						
5						

7	Test Case ID	TM_010	Test Case Description	click on logout to go back to login page		
8	Created By	Siddharth	Reviewed By	Muhammed		
9	Version	1				
10	QA tester's log	User must be able to exit the website through logout				
11	Tester's Name	Siddharth	Date Tested		Test Case (Pass/Fail)	pass(minor bugs)
12	NO.	Prerequisites	No.	Test Data		
13		1 access to browser		1 email= sid@gmail.com		
14		2 registered account		2 pass= Siddharth@23		
15	Test Scenario	User must be logout to the login page				
16	Step#	Step Details	Exprected Results	Actual Results	Pass/Fail/Not Executed	
17		1 click on logout	user is logged out and redirected to login page	user is logged out, but have to press twice to go to login		
18						
19						
20						
21						
22	Test Case ID	TM_011	Test Case Description	Generate plan in homepage with only destination		
23	Created By	Siddharth	Reviewed By	Muhammed		
24	Version	1				
25	QA tester's log	The api must generate a plan based on user input				
26	Tester's Name	siddharth	Date Tested	3-May-25	Test Case (Pass/Fail)	pass
27	NO.	Prerequisites	No.	Test Data		
28		1 access to browser		email= sid@gmail.com		
29		2 registered account		pass= Siddharth@23		
30	Test Scenario	On clicking submit plan is generated				
31	Step#	Step Details	Exprected Results	Actual Results	Pass/Fail/Not Executed	
32		1 enter the destination on the search bar	search bar gets filled and starts generating the plan	As expected	pass	
33		2 click on submit	Random plan appears as fields were empty	as expected	pass	
34						
35						
36	Test Case ID	TM_012	Test Case Description	Generate plan with all fields filled		
37	Created By	Siddharth	Reviewed By	Muhammed		
38	Version	1				
39	QA tester's log	Plan is generated according to all inputs of the user				
40	Tester's Name	Siddharth	Date Tested	3-May-23	Test Case (Pass/Fail)	pass
41	NO.	Prerequisites	No.	Test Data		
42		1 access to browser		email= sid@gmail.com		
43		2 registered account		pass= Siddharth@23		
44	Test Scenario					
45	Step#	Step Details	Exprected Results	Actual Results	Pass/Fail/Not Executed	
46		1 enter destination	search bar filled	as expected	pass	
47		2 enter number of days	search bar filled	as expected	pass	
48		3 enter preferences	search bar filled	as expected	pass	
49		4 enter mode of transport	search bar filled	as expected	pass	
50		5 click submit	specific plan is generated for the specified no. of days	as expected	pass	

Test Case ID	TM_013	Test Case Description	Click on save plan in plan page to save it to profile	
Created By	Siddharth	Reviewed By	Yahya	
Version	1			
QA tester's log	Clicking on save plan must save it to user profile			
Tester's Name	siddharth	Date Tested	30-Apr-23	Test Case (Pass/Fail) Fail
NO.	Prerequisites	No.	Test Data	
1	access to browser		email= sid@gmail.com	
2	registered account		pass= Siddharth@23	
Test Scenario	Plan must be saved			
Step#	Step Details	Expected Results	Actual Results	Pass/Fail/Not Executed
1	generate a plan by clicking submit	plan appears with save button below	as expected	pass
2	click on save plan	plan is saved and appears on user profile	button doesn't work	fail

GITHUB ISSUES TO TRACK ISSUES-

<https://github.com/CSC-648-SFSU/csc648-01-csc648-04-team-05/issues>

The screenshot shows the GitHub interface for the repository `CSC-648-SFSU / csc648-01-csc648-04-team-05`. The 'Issues' tab is selected, showing 5 open issues. The issues are:

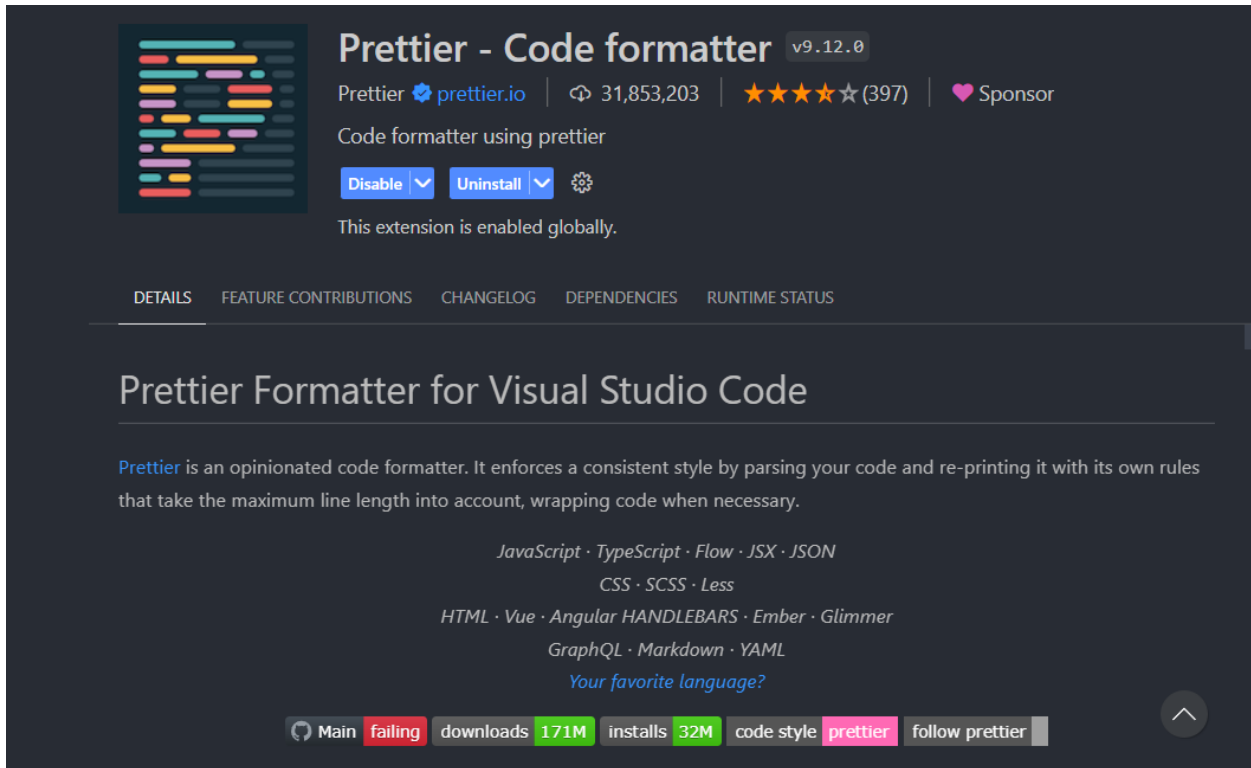
- save plan button in profile page needs to bne completed** (#11, opened 3 minutes ago by siddharths23)
- Logout sometimes doesn't navigate back to login page** (#10, opened 4 minutes ago by siddharths23)
- frontend for profile page and explore page after finishing backend** (#7, opened 3 weeks ago by siddharths23)
- frontend for homepage and navbar** (#6, opened 3 weeks ago by siddharths23)
- frontend for login and register** (#5, opened 3 weeks ago by siddharths23)

2) Coding practices

Coding style

As part of our coding standards, we decided to use Prettier to ensure consistent coding style throughout our codebase. We configured Prettier to follow the popular Google JavaScript Style Guide which covers a range of topics including variable naming, indentation, line length, and more. Prettier also helps us format our code automatically so we don't have to worry about minor formatting issues. It has significantly improved the readability and maintainability of our code. We have integrated Prettier with our code editor and version control system so every developer in our team can easily follow the same coding style. With Prettier, we can focus on writing clean and efficient code without worrying about formatting details.

Extension-



The screenshot shows the Visual Studio Code extension marketplace page for "Prettier - Code formatter" (version v9.12.0). The page includes a header with the extension name, version, and a "Sponsor" button. Below the header, there are tabs for "DETAILS", "FEATURE CONTRIBUTIONS", "CHANGELOG", "DEPENDENCIES", and "RUNTIME STATUS". The "DETAILS" tab is selected, showing the extension's description: "Prettier is an opinionated code formatter. It enforces a consistent style by parsing your code and re-printing it with its own rules that take the maximum line length into account, wrapping code when necessary." Below the description, there is a list of supported languages: JavaScript, TypeScript, Flow, JSX, JSON, CSS, SCSS, Less, HTML, Vue, Angular, HANDLEBARS, Ember, Glimmer, GraphQL, Markdown, and YAML. At the bottom, there is a status bar showing the extension's status: "Main" (failing), "downloads" (171M), "installs" (32M), "code style" (prettier), and "follow prettier".

Source files formatted by prettier-

<https://github.com/CSC-648-SFSU/csc648-01-csc648-04-team-05/blob/master/application/frontend/src/pages/LoginPage.js>

<https://github.com/CSC-648-SFSU/csc648-01-csc648-04-team-05/blob/master/application/frontend/src/pages/HomePage.js>

<https://github.com/CSC-648-SFSU/csc648-01-csc648-04-team-05/blob/master/application/frontend/src/pages/ExplorePage.js>

<https://github.com/CSC-648-SFSU/csc648-01-csc648-04-team-05/blob/master/application/frontend/src/pages/SignupPage.js>

<https://github.com/CSC-648-SFSU/csc648-01-csc648-04-team-05/blob/master/application/frontend/src/pages/ProfilePage.js>

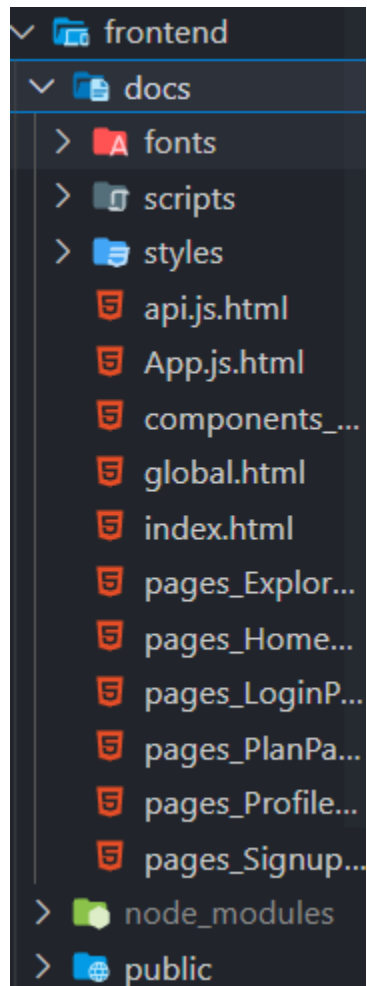
EXTRA CREDIT - DOCUMENTATION GENERATION

We used jsdoc to implement our documentation generator.

We first installed it by using `npm i jsdoc`. And then setup a docs start script in package.json to generate the documentation whenever we run `npm docs`.

```
"docs": "jsdoc -c jsdoc.conf.json",
```

The documentation is generated by adding comments that are recognized by jsdoc before every function and is automatically generated in the src folder as a docs folder. In order to see the generated documentation, we need to load the index.html file inside the docs folder.



> Press Enter to trigger a test run.

```
PS C:\Users\siddh\Desktop\csc648-01-csc648-04-team-05\application\frontend> cd docs
PS C:\Users\siddh\Desktop\csc648-01-csc648-04-team-05\application\frontend\docs> start index.html
PS C:\Users\siddh\Desktop\csc648-01-csc648-04-team-05\application\frontend\docs> 
```

Home

Home

Global

- App
- Explore
- Homepage
- LoginPage
- Navbar
- PlanPage
- ProfilePage
- SignupPage
- createCompletion
- destinations
- fetchCompletion
- getRandomDestinations
- handleLogout
- isValidJson

Documentation generated by [JSDoc 4.0.2](#) on Wed May 03 2023 16:21:24 GMT-0700 (Pacific Daylight Time)

Video demonstration of jsdoc generation-

https://drive.google.com/file/d/1Jf_Uckw2CzDI4XWb1u_RjCLNPMFPd1a-/view?usp=sharing