Software Test Plan

1 Analysis

Frontend

For the frontend, a combination of black box and white box testing was applied to each page of the application via a suite of automated unit tests to verify the correctness of core UI elements. These tests focus extra attention onto all inputs and forms present in the system and test both valid and invalid entries in each input. Cypress was used for all our UI tests. Cypress has a test-runner that runs in the browser and mimics the actions of a real user.

The aim of the black box tests in the frontend was to ensure the presence of core UI elements and the correctness of important text. These are tests that do not require knowledge of the inner workings of the project and are essentially sanity checks for simple error handling and ensuring that elements aren't lost between merges.

The white box tests use the intercept() function provided by Cypress to intercept the real API calls made by our system and inject them with mock data provided in the /cypress/fixtures folder. The white box tests apply branch coverage for every API call in the system to cover every potential HTTP status code that could be returned. These tests also accomodate places that use conditional rendering to display elements based on a certain state of the application by replicating those states. Branch coverage also ensures that we are testing the outcomes of submitting both valid and invalid inputs and performing the necessary error handling. Sometimes the lines between black and white box tests feel a little bit blurry but in our system, anytime there is mock data, it is referred to as a white box test.

Some advantages of using mock data to test the frontend is that we don't have to worry about running the server and we don't have to worry about clogging the database with test data. Using mock data is also much quicker than interfacing with our real backend.

The following tables outline the tests designed to meet the criteria specified above which involves checking essential UI elements via black box testing and performing branch coverage via white box testing. A description of their value and the functional requirement that they can be linked back to is also provided. The shaded cells represent the black box tests and the non-shaded cells represent the white box tests. Note that no expected vs. actual criteria have been provided here since the test name describes the expected behavior and all tests should succeed.

To derive the white box test cases for branch coverage, the code was analyzed and a description of each potential path for each function was provided in this table to ensure complete coverage.

analogyTestPage.spec.js

ID	FR	Test Name	Description
1	10.2	should have "Data Mining System" in the header	Ensures the branding is consistent on the analogy testing page.
2	10.2	should have a clickable profile icon in the header	Ensures the system can be logged out of from the analogy testing page
3	10.2	should have "Analogy Test" title	Ensures the title reflects the functionality
4	10.2	should have "Enter a third word below to perform an analogy test on the given words.	Ensures the subtitle gives the user the correct instructions
5	10.2	should contain the text entered on the previous page	Ensures the flow of information is correct in between pages
6	10.2	should contain the text clicked on the previous page	Ensures the flow of information is correct in between pages
7	10.2	should have a text input with "Word is to..." placeholder	Ensures the user has a place to enter the analogy test with the correct hint
8	10.2	should have a number input with default value 500	Ensures the default count is properly filled
9	10.2	accepts word input	Ensures the user has a place to enter the analogy test word and it accepts input correctly
10	10.2	accepts number input	Ensures the user has a place to enter the analogy test count and it accepts input correctly
11	10.2	should show error message if submit is clicked without entering word	Ensures error message is correct if no word is submitted
12	10.2	should contain the text clicked on the previous page	Ensures correct data flow between closestWords and analogyTest pages
13	10.2	should show error message if non-number is entered in number field	Ensures error message is correct if invalid number is submitted
14	10.2	should show error message if negative number is entered in	Ensures error message is correct if invalid number is submitted

		number field	
15	10.2	should route home if word A, word B and trained model ID is not defined	Ensures application does not break when users mess with the url
16	10.2	should show alert if /verify/analogy-test throws 401	Ensures unauthorized users are backed out of the app
17	10.2	should show loading indicator while fetching results	Helps verify that users know the application isn't broken
18	10.2	should not show table if no results	Ensures no results are shown if API returns an empty array
19		should show table on successful API call	Ensures the system can correctly retrieve and display results for analogy testing
20	10.2	should show newly entered word on successful API call	Ensures correct flow of data between API calls

changePasswordPage.spec.js

ID	FR	Test Name	Description
21	3	it should have "Data Mining System" in the header	Ensures the branding is consistent on the change password page
23	3	it should not have profile icon in the header	Profile icon should not show until the user has logged in
24	3	it has a double input form container	Ensures user has a place to enter in their new and old password
25	3	should show generic error message on API error	Should catch any unexpected HTTP errors
26	3	should show error message if new password isn't strong enough	Ensures users are informed of password requirements
27	3	should route home on successful password change	Ensures correct flow of pages on successful password change
28	3	should show error message if old password doesn't match temporary password	User should be informed if there is a password mismatch

closestWordsPage.spec.js

ID	FR	Test Name	Description
29	10.1	it should have "Data Mining System"	Ensures correct branding on closest

		in the header	words page
30	10.1	should have a clickable profile icon in the header	Ensures the system can be logged out of from the closest words page
31	10.1	should have "Closest Words" title	Page title should be correct
32	10.1	should have "Enter a word below to receive the n most similar words to it" subtitle	Page subtitle should be correct
33	10.1	should have a text input with "Word" placeholder	Ensures the user has a place to enter the test word with the correct hint
34	10.1	should have a number input with default value 100	Ensures user has a place to enter number of closest words with correct default value
35	10.1	accepts word input	Ensures words can be submitted correctly
36	10.1	accepts number input	Ensures number can be submitted correctly
37	10.1	should show error message if submit is clicked without entering word	Ensures only non-empty strings can be submitted for proximity check
38	10.1	should show error message if non-number is entered in number field	Ensures a valid number will be submitted
39	10.1	should show error message if negative number is entered in number field	Ensures a positive number will be submitted
40	10.1	should route to home page if trained model Id is not given	Ensures application does not break when users mess with the url
41	10.1	should show alert if API throws 401	Ensures unauthorized users are backed out of the app
42	10.1	should not show table if no results	Ensures correct behavior when results are empty
43	10.1	should show table on successful API call	Ensures correct behavior when results are returned
44	10.1	should show loading indicator while fetching results	Ensures users know the page is loading, not broken
45	10.1	should route to analogy test page if	Ensures correct navigation to analogy

		a word is clicked	test

homePage.spec.js

Some black box tests are repeated as we switch between the Dataset and Models tabs

ID	FR	Test Name	Description
46	5	should have keyword bar with placeholder text	Ensures keyword bar exists and has correct hint
47	5	should have submit button for keywords with plus icon	Ensures keyword bar has a submit button
48	5	should not send a request to filter-task if keywords are empty	Ensures that only non-empty strings are sent for filtering
49	5	should send keywords to request in lowercase	Ensures that the backend can correctly interpret the given words
50	5	should send keywords when enter is typed versus clicking the button	Ensures the user can get results by pressing enter key
51	5	should show alert if 401 error is thrown on keyword submit	Ensures unauthorized users are backed out of the app
52	6	should have "Data Mining System" in the header	Ensures branding is consistent on home page
53	6	should have a clickable profile icon in the header	Ensures the system can be logged out of from the analogy testing page
54	6	should have "New Dataset" title	Ensures page has the correct title
55	6	should have dataset tab	Dataset tab should exist on home page
56	6	should have models tab	Models tab should exist on home page
57	6	should have dataset tab selected	Default active tab should be Dataset
58	6	should show loading indicator while fetching datasets	User should know the system is loading and not broken
59	6	should show message when no datasets are returned	Users should be aware that there were no results
60	6	should show correct datasets on successful API call	Returned datasets should have expected text in the UI
61	6	should show 'Empty dataset' when num_papers = 0	Users should know when they’ve come across an empty dataset

62	6	should show alert if GET/train-task throws 401	Ensures unauthorized users are backed out of the app
63	6	should show loading indicator on incomplete dataset	User should know the system is loading and not broken
64	6	should continue fetching incomplete datasets	UI should automatically update without needed a page refresh
65	9	should have "Data Mining System" in the header	Ensures consistent branding even when model tab is clicked
66	9	should have dataset tab	Dataset should still be accessible from models tab
67	9	should have models tab	Models should still be accessible from models tab
68	9	should have models tab selected	Models tab should be active after it is selected
69	9	should show correct models on successful API call	Returned models should appear with expected titles
70	9	should show alert if GET/train-task throws 401	Ensures unauthorized users are backed out of the app
71	9	should show loading indicator while fetching models	User should know the system is loading and not broken
72	9	should show message when no models are returned	Users are informed when no results are returned

hyperparameterAdjustmentPage.spec.js

ID	FR	Test Name	Description
73	7	should have "Data Mining System" in the header	Ensures consistent branding across hyperparameter adjustment page
74	7	should have a clickable profile icon in the header	Ensures the system can be logged out of from the hyperparameterAdjustmentPage
75	7	should have all correct hyperparameter inputs	All hyperparameter inputs should be present
76	8	should have train model button	Ensures users have a way to train their datasets
77	7	should navigate to home page if no dataset id is given	Ensures every trained dataset will have an id and users can try and get to the

			training page without clicking a dataset
78	7	should show alert if 401 is thrown for /train-task/suggest-hparams	Ensures unauthorized users are backed out of the app
79	7	should input suggestions as placeholders correctly	All hyperparameter inputs should loaded with the correct values from the suggestions endpoint
80	7	should send correct hyperparameters on form submit	Ensures no inconsistencies between what is entered and what is sent to train endpoint
81	7	should send correct dataset id on form submit	Ensures no inconsistencies between the dataset that is clicked and the one that is trained on
82	7	should show 'Please complete all fields' message if not all fields are filled in	Doesn’t allow an incomplete form to be submitted
83	7	should show generic error message on API fail	Ensures any other HTTP status codes are accounted for
84	8	should route to home page on form submit	Successful trains should route to home page

loginPage.spec.js

ID	FR	Test Name	Description
85	2	it should have "Data Mining System" title	Ensures consistent branding on login screen
86	2	it has a double input form container	Ensures user has a styled place to enter in their username and password
87	2	has a username input with placeholder "Username"	Ensures users have a place to enter their username with the correct hint
88	2	has a password input with placeholder "Password"	Ensures users have a place to enter their password with the correct hint
89	2	accepts username input	Ensures the user has a place to enter their username and it accepts input correctly
90	2	accepts password input	Ensures the user has a place to enter their password and it accepts input correctly

91	2	it has a "Log in" button	Ensures users have an obvious way to log into the system
92	2	it has a "No account? Get started here" clickable text	Ensures users have a way to request an account
93	2	should show error message if username or password is incorrect	Users should get a helpful error message when their credentials don’t match
94	2	should show generic error message if API fails	Ensures unauthorized users are backed out of the app
95	2	should route to change password page on incomplete login	Incomplete accounts should be prompted to change their password
96	2	should route to home page on complete login	Complete accounts should be routed to home page on login

logoutPage.spec.js

ID	FR	Test Name	Description
97	4	it should have "Data Mining System" in the header	Ensures consistent branding on logout page
98	4	it should have a clickable profile icon in the header	Profile page should still be accessible even on logout page
99	4	it should have a logout button	Logout button should be visible on logout page
100	4	should redirect to home page when logout is clicked	All branches lead to redirecting to logout page once logout button is clicked

requestAccountPage.spec.js

ID	FR	Test Name	Description
101	1	it has a double input form container	Ensures user has a styled place to enter in their email and username
102	1	it should have "Request Account" title	Ensures the title reflects the functionality
103	1	it should have correct subtitle	Ensures the subtitle reflects the functionality
104	1	has an username input with placeholder "Username"	Ensures users have a place to enter their username with the correct hint
105	1	has an email input with placeholder	Ensures users have a place to enter their

		"Email"	email with the correct hint
106	1	accepts username input	Ensures the user has a place to enter their username and it accepts input correctly
107	1	accepts email input	Ensures the user has a place to enter their email and it accepts input correctly
109	1	has a "Request Account" button	Users should have an obvious way to request an account
110	1	should show error message if either input is blank	Ensures an incomplete will not be submitted
111	1	should show error message if username input is blank	Ensures an incomplete will not be submitted
112	1	should show error message if email input is blank	Ensures an incomplete will not be submitted
113	1	should show error message if email is invalid	Ensures an invalid email will not be submitted
114	1	should show error message if email/password already exists	Ensures users have a thorough error message if username is taken
115	1	should show generic error message if API fails	Ensures users see an error message if something else goes wrong
116	1	should navigate to login page on success	Gives user an indication that their request was submitted

visualizationPage.spec.js

ID	FR	Test Name	Description
117	11, 12	should have "Data Mining System" in the header	Ensures consistent branding on visualization page
118	11, 12	should have a clickable profile icon in the header	Ensures users have a way to log get to the profile page from visualization page
119	11, 12	should have a chart visible	Ensures the visualization page has a chart visible
120	11, 12	should route to home page if trained model Id is not given	Ensures application does not break when users mess with the url
121	11, 12	should show alert if API throws 401	Ensures unauthorized users are backed out of the app

122	11, 12	should show chart on successful API call	Chart should be visible when /visualize succeeds

Backend

There were two main considerations in the design of the backend tests. First, each endpoint of the backend was written separately from the part of the frontend that used that endpoint. Second, the error cases (e.g. 404 errors) of the endpoints were generally much less complex than the success cases of those endpoints.

The backend tests were written as the code was developed, and were designed to maximize branch coverage while being concise and easy to write.

Most endpoints were quite logically simple, and black-box testing was used to test them. More complex parts of the code, such as the session management system and the filter and train task runners, were tested in layers with important functions tested individually and the subsystem tested as a whole. Throughout, an effort was made to limit redundancy while layering tests like this, which reduced the number of tests needed but also somewhat reduced the comprehensiveness of the test suite as a whole.

Integrated Testing

For our integrated testing, we wanted a way to verify our conformance to our functional requirements for the integrated system. This process was a bit more involved since we had to ensure both frontend and backend repositories are running, and provide a way to seed the database in between runs. Since, however, the integrated tests interface with our real backend and a real database, they provide us with a piece of mind that all the functional requirements can be completed end-to-end by a user of our system.

The test database has a subset of our user accounts with archived metadata for datasets and models. The test database does not link to the research paper data due to the sheer volume of it. Thus, the tests do not require that neither filter nor train task-runner be running since there are no papers to perform operations on. These tests focus solely on the ability to complete the requirements end-to-end with the correct API calls being made.

The integrated tests closely follow the flow of the application that one would take to use our system. Like the frontend tests, our integrated tests use Cypress to enter our system from the browser and mimic the actions of a user.

The following tables outline the tests designed to demonstrate the conformance to our functional requirements. The description provided outlines how each test validates each of our requirements.

addDataset.spec.js

ID	FR	Test Name	Description
1	5, 6	should add incomplete dataset to the list of datasets	FR5: Test runner uses keyword bar to filter a dataset with the keyword ‘test’ FR6: After the runner submits the keyword, it waits for the page to reload and verifies that a dataset with the keyword ‘test’ was added to the dataset tab, and that the loading indicator is showing (since the filter task-runner has/will not pick up the task)

analogyTest.spec.js

ID	FR	Test Name	Description
2	10.1, 10.2	should perform an end to end analogy test	FR10.1: Test runner navigates to the closest words page, submits a word and verifies that the expected result appears. FR10.2: Test runner clicks on the expected result and enters a new word into the analogy test. The test runner verifies that the correct word was entered and that the expected result appears

changePassword.spec.js

ID	FR	Test Name	Description
3	3	should redirect to change password page on login	FR3: Test runner verifies that an incomplete account will be prompted to change their password
4	3	should show error message if old password does not match temp password	FR3: Test runner verifies that the user sees an error message if they entered in a password mismatch
5	3	should redirect home on successful password change	FR3: Verifies that the change password API completes successfully and the user is directed to the home page
6	2, 3	should log in successfully after password change	FR2, 3: Verifies the user can log in with the updated credentials

login.spec.js

ID	FR	Test Name	Description
7	2	should show error message if	FR2: Verifies that user cannot login with

		username or password is incorrect	incorrect credentials and sees an error message
8	2	should route to home page on complete login	FR2: Verifies that user can login with correct credentials and is redirected to the correct page
9	2	should route to change password page on complete login	FR2: Verifies that users who have a temporary password will be prompted to change it

logout.spec.js

ID	FR	Test Name	Description
10	4	should log user out without errors	FR4: Test runner finds logout button and verifies that the user is correctly navigated to the login page

requestAccount.spec.js

ID	FR	Test Name	Description
11	1	should show error message if using duplicate username	FR1: Verifies that users can not request accounts with a duplicate username
12	1	should show error message if using duplicate email	FR1: Verifies that users can not request accounts with a duplicate email
13	1	should redirect to login screen on successful submission	FR1: Verifies that an account is requested without errors and user is returned to login screen

trainDataset.spec.js

ID	FR	Test Name	Description
14	7	should suggest correct hyperparameters	FR7: Task runner enters dataset tab and clicks ‘Train’ on a dataset. Task runner is taken to hyperparameter adjustment page where it verifies suggested hyperparameters
15	7, 8	should train model with correct id	FR7: Task runner enters dataset tab and clicks ‘Train’ on a dataset. Task runner is taken to hyperparameter adjustment page where it updates them FR8: After the hyperparameters are adjusted, task runner presses the 'Train Model’ button and verifies that the

			request has the correct id.
16	9	should show newly trained model in models list	FR9: Test runner navigates to models page and verifies that the trained model now appears in the models tab with the correct name

visualize.spec.js

ID	FR	Test Name	Description
17	11, 12	should request plot data and receive the correct arrays in the response	FR12: Test runner navigates to a previously existing model to visualize it by clicking the 'Visualize' button FR11: Test runner verifies that the visualize endpoint was called and correctly returns the labels, x and y arrays in the expected format.

2 Running Instructions

Before you run testing, follow the setup instructions found in the deployment guide ("Set up backend" and "Set up frontend" in Deployment.md), neglecting the section "Initialize the database" as the tests will use their own, dedicated database if necessary.

Frontend

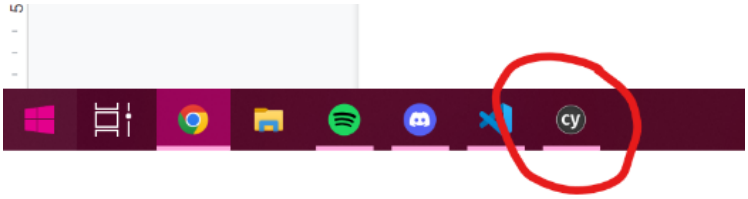
Navigate to the /frontend directory and run the following commands:

```
npm install
npm start
```

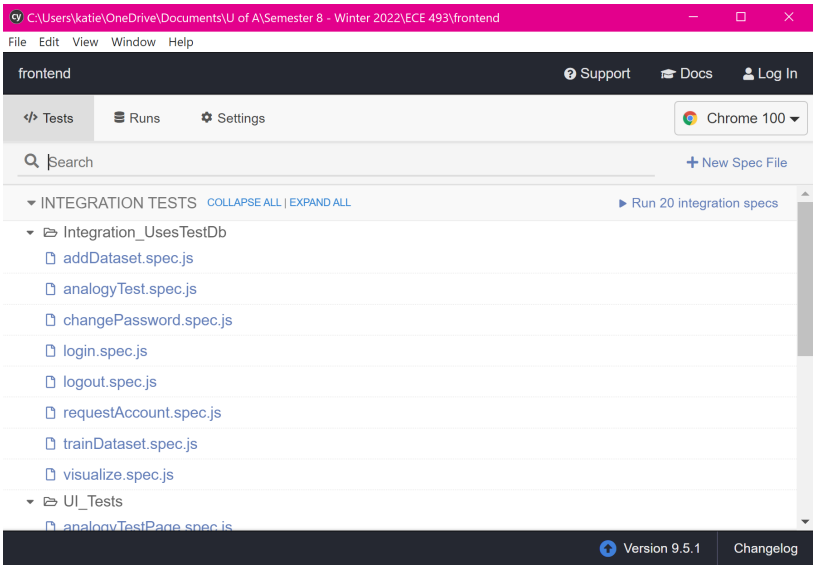
This will start the frontend web-app. Open up another terminal, also in the /frontend directory, and run the following command:

```
npm run cypress
```

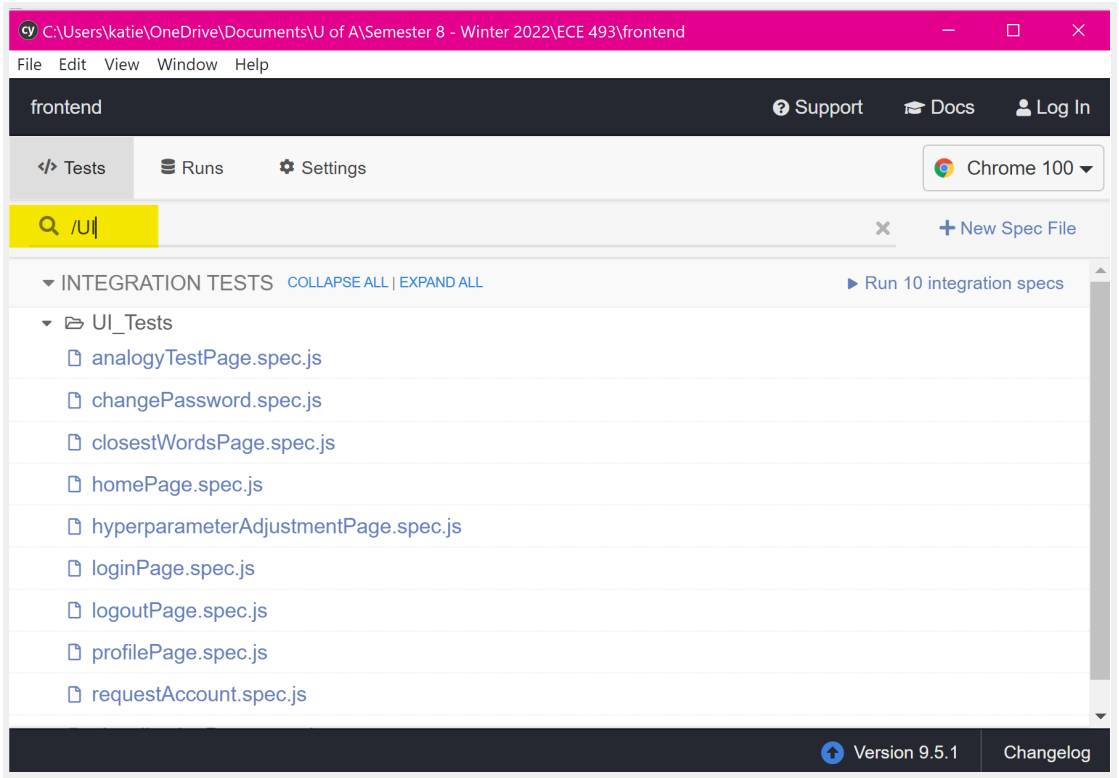
This command will open the test runner, this will take about a minute or two and when it finishes, you will see the Cypress logo on your taskbar, click on it. If you're on the paperspace machine, or if this command fails, see the Error Handling section.

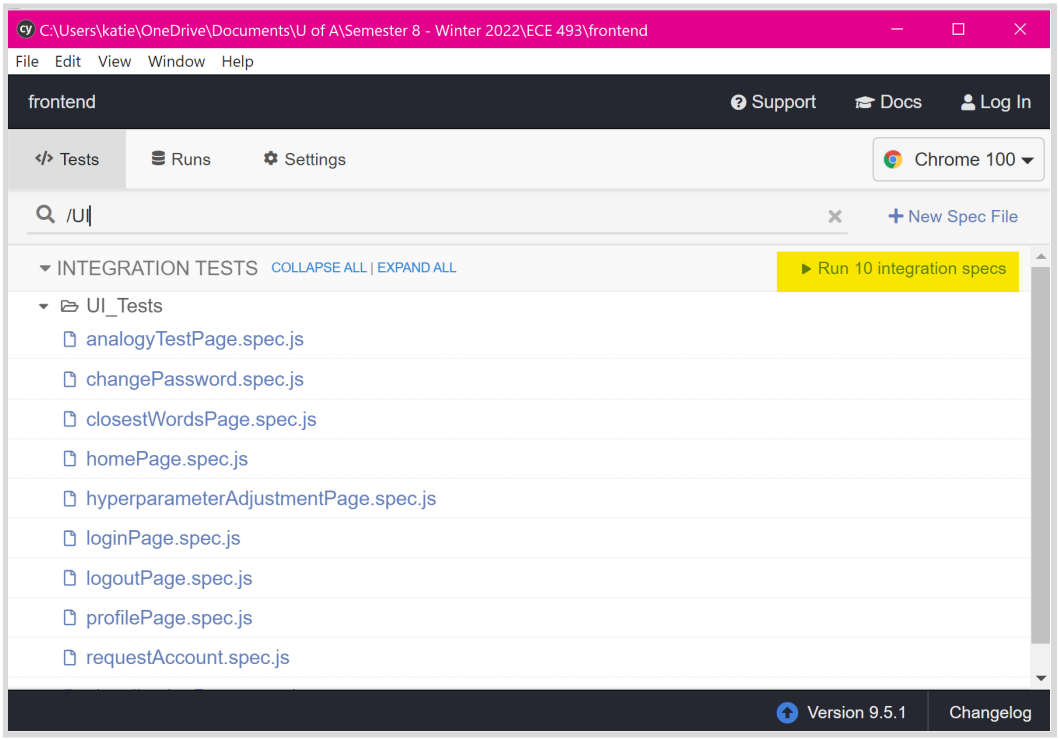


When you open the test runner, you will see a list of all the Cypress tests for our system.

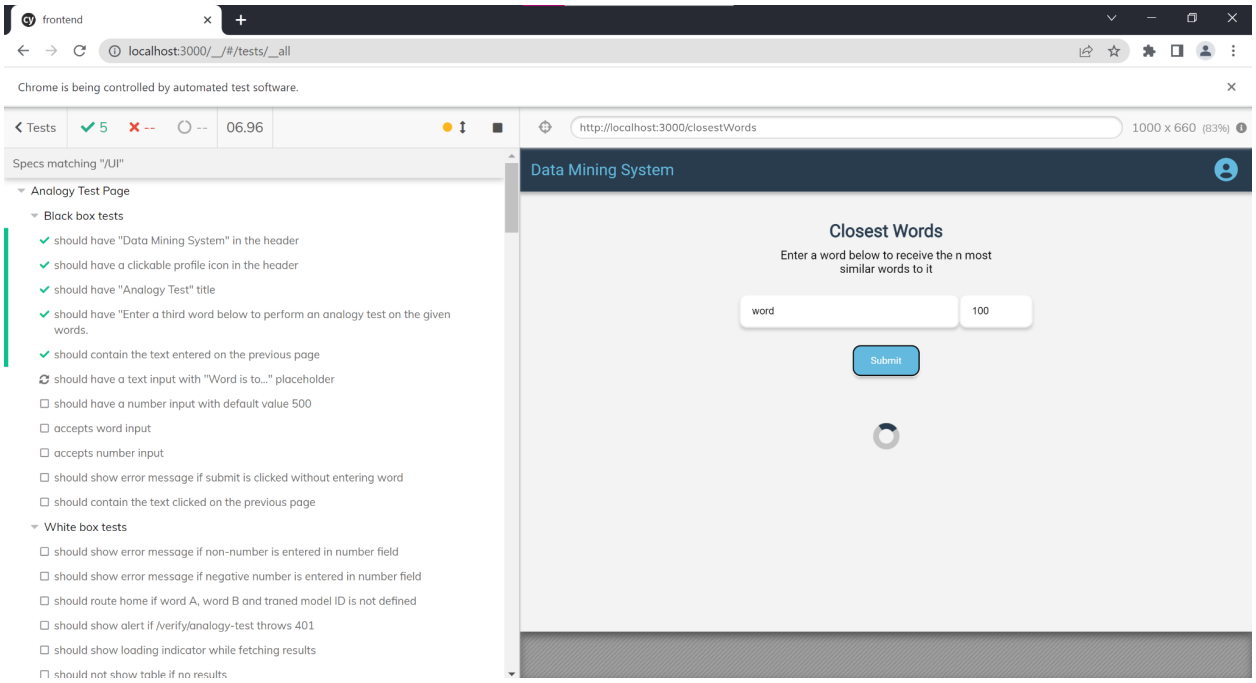


To run the frontend tests, type /UI into the search bar and then click “Run 10 integration specs”. See below.


C:\Users\katie\OneDrive\Documents\U of A\Semester 8 - Winter 2022\ECE 493\frontend
File Edit View Window Help
frontend
Support
Docs
Log In
Tests
Runs
Settings
Chrome 100
/UI
New Spec File
INTEGRATION TESTS
COLLAPSE ALL | EXPAND ALL
Run 10 integration specs
UI_Tests
analogyTestPage.spec.js
changePassword.spec.js
closestWordsPage.spec.js
homePage.spec.js
hyperparameterAdjustmentPage.spec.js
loginPage.spec.js
logoutPage.spec.js
profilePage.spec.js
requestAccount.spec.js
Version 9.5.1
Changelog


C:\Users\katie\OneDrive\Documents\U of A\Semester 8 - Winter 2022\ECE 493\frontend
File Edit View Window Help
frontend
Support
Docs
Log In
Tests
Runs
Settings
Chrome 100
/UI
New Spec File
INTEGRATION TESTS
COLLAPSE ALL | EXPAND ALL
Run 10 integration specs
UI_Tests
analogyTestPage.spec.js
changePassword.spec.js
closestWordsPage.spec.js
homePage.spec.js
hyperparameterAdjustmentPage.spec.js
loginPage.spec.js
logoutPage.spec.js
profilePage.spec.js
requestAccount.spec.js
Version 9.5.1
Changelog

This will open the test runner and run all frontend tests automatically. You can see what the test runner is doing in real time and when the tests finish, you can use the left panel to inspect the tests and look at snapshots of the DOM over the duration of each test.



Error Handling

If you see an error after running `npm run cypress` you can try and run the UI tests headlessly with the following command:

```
npx cypress run --spec ".\cypress\integration\UI_Tests\*.spec.js"
```

It is expected to encounter errors running Cypress on the paperspace machine. It is suggested to clone the frontend repository locally and run Cypress in your own environment to view the test runner.

Backend

Navigate to the api directory and run

```
source venv/bin/activate
pytest tests
```

These tests use the docker daemon to test the prebuilt word2vec implementation. If your user account is not part of the *docker* group (e.g. in the paperspace machine, it is not), run

```
sudo venv/bin/pytest tests
```

Instead of the second command.

Integrated Tests

Navigate to the /api directory and run the following commands:

```
make test-db-init
```
You should only ever need to run this once, and if you are in the paperspace machine it has already been run so you don't have to worry about it.
```
make test-db-start
```
You only need to run this once per time you restart the machine. If you are on the paperspace machine you may need to run this if it has been restarted since we have used it last.
```
./integrated-tests/init.bash
source venv/bin/activate
make test-serve
```

The last command will start the backend and point it to the test database which was seeded with the init.bash script.

In another terminal, navigate to the /frontend directory and follow the steps in the Frontend section except this time, type "/Integration_" instead of "/UI" and run the 8 integration specs.

If you wish to run the test suite a second time, you will need to reseed the database by re-entering the following command:

```
./integrated-tests/init.bash
```

If you aren't on the paperspace machine, you will need to add a new secrets file to the /cypress/integration folder. Create a new file called `secrets.js` in /cypress/integration and paste in the following information:

```
export const completeAccountUsername = "user5";
export const completeAccountPassword = "password";
export const completeAccountEmail = "user5@example.com";
export const incompleteAccountUsername = "user7";
export const incompleteAccountPassword = "password";
export const incompleteAccountUsername2 = "user6";
export const incompleteAccountPassword2 = "password";
```