System and Unit Test Report

PathFinder

Pathfinders

06/01/21

System Test Scenarios

- User story 1 from sprint 1: As a user, I want to be able to access the metrics quickly.
    - 1) Start web application
    - 2) Access a job by selecting one from the drop down menu
    - 3) Click one of the categories at the bottom of the screen.
    - 4) The metrics should appear instantly.
- User story 1 from sprint 2: As a user, I want to be able to use the webapp without it crashing.
    - 1) Start web application
    - 2) Click on drop down menus
    - 3) Click on buttons - ‘skills’, ‘languages’, ‘degrees’, etc.
    - 4) Users should see the corresponding data according to the buttons all without crashing and continue to do so.
- User story 2 from sprint 2: As a developer, I want to be able to scrape data from a set of job postings: 5 story points.
    - 1) Navigate to the folder that contains the program.
    - 2) Run the following command:
        - Python3 web_scraper.py -p 2 -j 1
    - 3)This will start the scraping process. The user should be able to see the jobIDs currently being scraped, monitor any errors that may have occurred, and see 0 be printed on a successful data upload
- User story 1 from sprint 3: As a user, I want to be able to have a Webapp to access the calculations on the scraped data
    -
    - 1) Start web application
    - 2) Users should see a representation of the scraped data as a pie chart.
    - 3) Click the “Skills” button from the bottom of the card
    - 4) Users should see the data update to display the most popular skills from the job displayed in the dropdown
    - 4) Click the dropdown menu and select “Data Engineer”
    - 5) Users should see the data update to display the selected query for “Data Engineer”

- User story 1 from sprint 4: As a user, I want to know what degrees are needed for [insert job title]
    - 1) Start web application
    - 2) Select the desired job field from the drop down menu near the top of the screen.
    - 3) Click the “Degrees” button near the bottom of the page, the button should change from green to blue.
    - 4) A pie chart should appear on the screen.
    - 5) Hover over each section of the pie chart to see the percentage of job postings of the desired job that require a particular degree.
- User story 2 from sprint 4: As a user, I want to see the most in-demand skills from [insert job title]
    - 1) Start web application
    - 2) Select the desired job field from the drop down menu near the top of the screen.
    - 3) Click the “Skills” button near the bottom of the page, the button should change from green to blue.
    - 4) A pie chart should appear on the screen.
    - 5) Hover over each section of the pie chart to see the percentage of job postings of the desired job that desire a particular skill.
-

Unit Tests

- Web Scraper
    - Due to the nature of the web scraper, conventional unit testing was not viable or useful. The web scraper gathers data from websites through a browser, and as such is expected to handle a non-finite number of possible inputs, in this case job descriptions. Scraping the same job description, or the same page of job descriptions, as a unit test would not prove or disprove functionality. As such, I decided to use full system tests with parameters that could be controlled from web_scraper.py.
        - An example of a test I would run would be having the scraper scrape one job per page for 40 pages, to test the scraper’s functionality of crawling through linkedin job pages.
            - This in turn tested every sub-component, ensuring that the depended-on components functioned as desired.
        - The major acceptance test was to populate the database by successfully running the program for over 12 hours, with minimal errors.
- Data formatter
    - data_formatter.py

        - Did unit testing for every function within the DataFormatter. Below are sample equivalence classes and test cases.

Same equivalence class for Seniority levels

Equivalence classes	Description	Possible Values
ECs-garbage	Garbage values for Seniority	Anything but the list of seniority values list
ECs-associate	“Associate” for Seniority	Associate
ECs-mslvl	“Mid-Senior level” for seniority	Mid-Senior level

        - The corresponding unit test for the above is in test_extract_seniority() from test_formatter.py.
        - There’s 10 unit tests that’s similar to the example given above with similar equivalence classes.
        - Did integration testing to test the how the functions work together in the DataFormatter
- Database upload/Database connection
    - Testing the database upload functionality is a reflection of the database itself, so testing for particular ranges of values won’t prove or disprove that it will always work as intended because testing a single input when the database is under a light load versus testing a single input when the database is under heavy load is very different. Due to this whole-system tests were performed on databaseUpload.py in different scenarios and under varying loads.
        - The success of these tests were verified by querying the database and checking that the number of job posts scraped reflect the number of job posts in the database.
            - Note: the number of jobs scraped doesn’t necessarily equate to the number of jobs uploaded to the database due to repeat job IDs, so checking that isn’t a reliable metric for success.
        - Attributes were also checked to make sure that they were in the range of appropriate values.
    - Tests performed:
        - Receiving a small amount of job posts from a single job title.
        - Receiving a large amount of job posts from a single job title.
        - Receiving a small amount of job posts from multiple job titles.
        - Receiving a small amount of job posts from multiple job titles that don’t have a table in the database.

        - Receiving a large amount of job posts from multiple job titles that don't have a table in the database.
- Front end
    - frontend/src/frontend.test.js
        - Using Jest to create and close a new browser tab for each test case, with a timeout set to 10000 (10 seconds). Each test case is independent from the previous ones, but all rely on using local host port 3000 as the source url.
    - backend
        - The user is only allowed to interact with buttons and dropdowns, so the margin for error is slim when it comes to querying from the database. There is no unit test for the connection from user interface to database.
    - Tests performed:
        - Page title of web app to be *PathFinder*
        - New card button to be loaded
        - Initial empty pie as null
        - Initial empty card set as null