COURSEWORK 2

COMP0034

Mar 2024

Table of Contents

I. App Code

1. Code structure

To improve code readability and overall quality, the latter was structured/divided into multiple packages and modules. The code's structure can be seen on Figure 1.

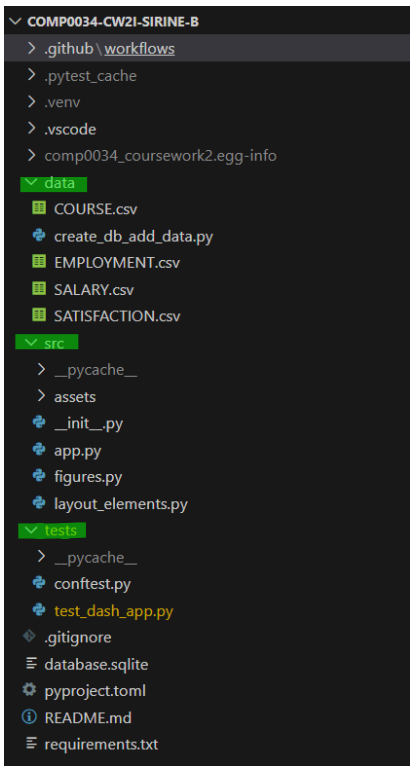


Figure 1: Structure of application code

The "data" package contains the CSV files for my dataset and a code to create a database and import data from the CSV to it using sqlite3.

The "src" package contains various modules that make up the Dash application. I decided to separate the code application into 3 different files so as to prevent having one cluttered file which would be hard to read, understand and modify if needed (i.e. to improve code readability and quality).

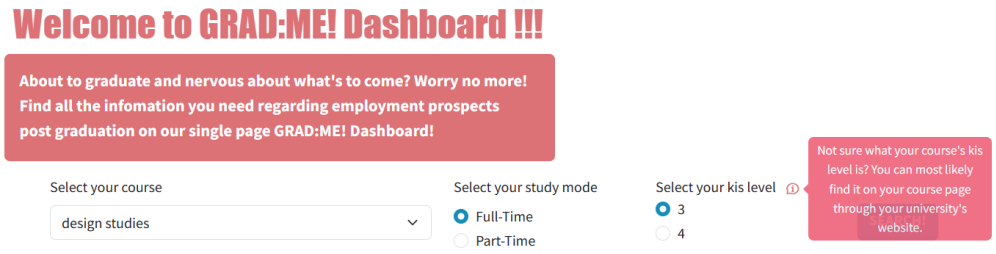
- An "assets" folder containing a CSS stylesheet and the logo of the info tooltip
- A "figures.py" file where were defined the functions which generate all the figures needed for the web app.
- A "layout.py" file in which I set out the layout of the web app (define each row, column etc).
- A "app.py" file which uses the previous 3 files to create the Dash app, in which the callback functions and error handling features are defined.

The "tests" package contains a "conftest.py" and a "test_dash_app.py" files. More details on the testing performed can be found in the "Testing" section of this report.

2. Extra web app features

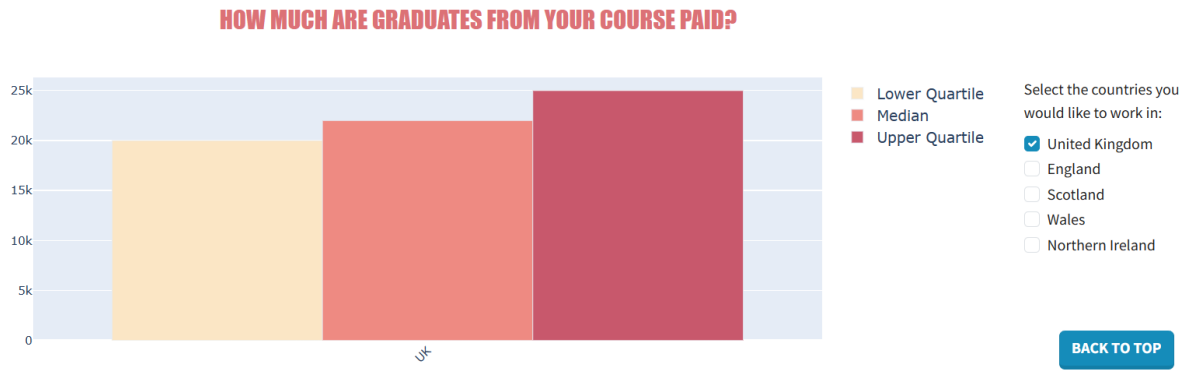
a. Info Tooltip

I added an info tooltip as an additional feature as the users (i.e. students) may not necessarily be familiar with the term “kis_level”.



b. Back to Top button

To view the salary information (i.e. the bar chart), the users need to scroll a little, making the filter section not visible anymore and making it necessary for them to scroll back up. To make this easier, and more practical (especially if we’re thinking of further improving the web app and making it more exhaustive/longer), a back to the top button was added to the web app.



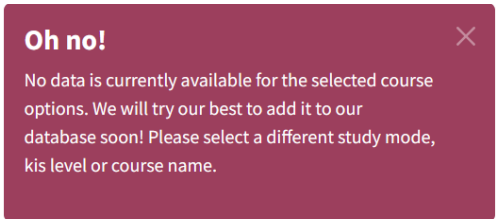
3. Error handling

Error handling is necessary, especially when designing a web-app as it is what enables us to spot an error when it happens, understand its origin and put in place a method to handle it when it happens again.

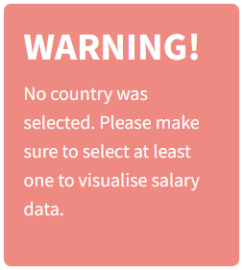
To prevent any bugs from occurring whilst the user is using the app, I came up with different scenarios which could raise an error (e.g. if the user selects a combination of course filters for which there is no data available) and added sections in the figures.py and callback functions to output error messages to the users whenever they would occur.

The two main errors handled in this web application are the following:

1. When the user selects a combination of course options for which no data is available.
   In this case, the following alert message is displayed:



2. When the user deselects all the countries from the countries filter.



II. Test Code

A comprehensive testing strategy was implemented to ensure the robustness of my web application. My testing strategy includes the following components:

1. Pytest and Selenium testing

I have used both pytest and selenium throughout my code. The performed tests cover both the correct, and error cases to ensure that the web application behaves as expected in all scenarios.

- **Two basic tests** were performed to make sure the web app server works correctly, and basic elements of the app could be detected.
  - **test_server_live()**
  - **test_app_header()**

- **Two tests were written to simulate a sequence of actions** performed by users and check that the web application behaves correctly by outputting the expected graphs.
  - **test_select_course_filters()** → simulates user selecting course filters + verifies output
  - **test_select_countries()** → simulates user selecting countries of interest for the bar chart + verifies output

- **Two tests were written to check my error handling**. They simulate users selecting (or deselecting) a combination of filter options for which no data is available and for which therefore, an error message should be displayed.

- **test_course_selection_error_message()**
- **test_countries_selection_error()**

The purpose of testing these error cases is to check that all errors that can potentially be made by users are handled correctly both from the developer's point of view (i.e. no bug in the system/web app) and from the users' perspective by having clear error messages displayed that help them understand what went wrong and what they can do to solve it.

A breakdown of the tests performed is shown on the figure below. All of them passed and more details can be found on the code's docstring.

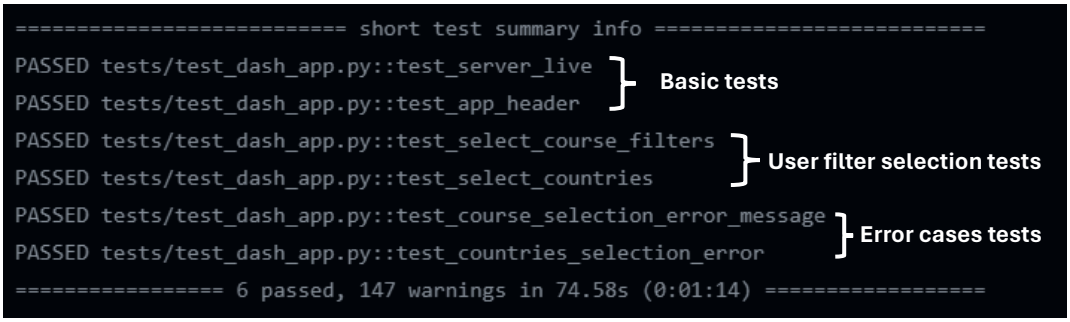


Figure 2: Screenshot of the short test summary info

NOTE:

In addition to the six above-described tests, I also wanted to test the info tooltip and back-to-top button but faced some errors that I tried to debug various times (can be seen on GitHub Actions) but was not able to. As such, I wanted to include the logic I thought of for these 2 tests.

For test_info_tooltip(dash_duo):

"""
GIVEN the app is running
WHEN the user hovers over the info tooltip
THEN a text with some additional information regarding kis_level is displayed
"""

1. Find the help tooltip on web page

tool_tip=dash_duo.find_element("#pic")

2. Extract/Read the tooltip text

help_tooltip_text=tool_tip.text()

3. Check the tooltip text is correct

```
assert help_tooltip_text=="Not sure what your course's kis level is?\
                You can most likely find it on your course page\
                through your university's website."
```

For test_back_to_top_button(dash_duo):

"""
GIVEN the app is running
WHEN the user clicks on the 'Back to Top' button
THEN the web page should automatically scroll back up to the top of the page
"""

1. **Get y-offset of top of the page**
2. **Assert that it is equal to 0 (since we have not yet scrolled down)**
3. **Use Selenium Actions to scroll down the page**
4. **Get y-offset of top of the page**
5. **Assert that it is not equal to 0 anymore**
6. **Find the back to top button (using find_elements and the button's CSS selector) + click on it**
7. **Get y-offset of top of the page**
8. **Assert that it is equal to 0 (since we would have scrolled back up to the top of the page)**

For the tooltip test function, the main error was that I could not find the tooltip within the web app despite having tried using various methods: its CSS selector, X-path, ID name, even considering the fact that it could be inside of an iframe and wrote a piece of code that would find it within the iframe but that also did not work.

For the 'back to top' button, in addition to not being to locate some elements, the biggest concern was that I was not able to use the execute_script functions along with dash_duo to get the page's location etc (e.g. execute_script("window.scrollTo(0, document.body.scrollHeight)).

All of my attempts at debugging these test functions were documented through CI and can be found on GitHub Actions.

2. Test Coverage

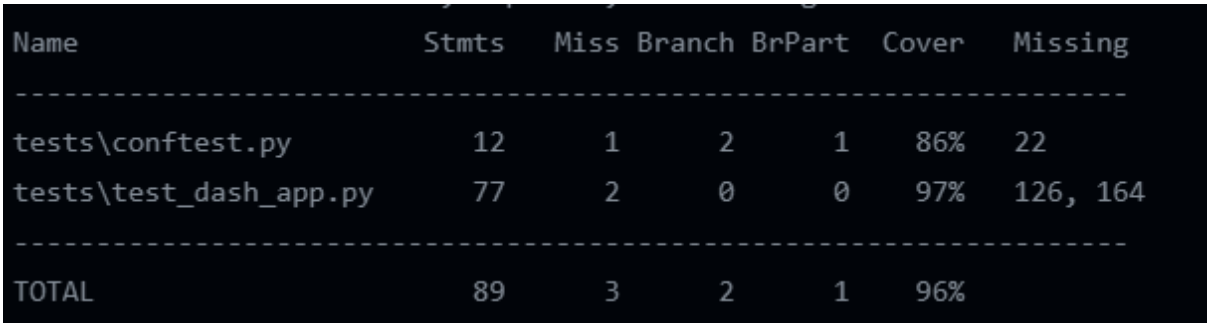
Using the pytest coverage command, I obtained a report of my code's coverage.

It is important to conduct code coverage as, having all the tests pass (i.e. successful) doesn't tell us anything about whether those tests are actually representative of/running on the whole code.

I have achieved a test coverage of 96% in total and the detailed results can be found on Figure 3. This high coverage entails that the large majority of my code is exercised/covered by the performed tests, thus reducing the likelihood of undetected bugs.

However, it is important to keep in mind that having a high coverage is not necessarily good (on its own) since just covering code doesn't mean that all cases/code sections are tested adequately. As such, it is important to have a

good balance between the tests' coverage and their relevant/adequacy to the application at hand.



Name	Stmts	Miss	Branch	BrPart	Cover	Missing
tests\conftest.py	12	1	2	1	86%	22
tests\test_dash_app.py	77	2	0	0	97%	126, 164
TOTAL	89	3	2	1	96%	

Figure 3: Screenshot of coverage report (html format)

3. Continuous Integration (CI)

CI was implemented through GitHub actions and enabled me to run my tests and coverage reports on each commit/push request. This enabled me to immediately notice whether a change in my code introduced an error and thus facilitated and sped up the debugging process.

III. Tools and Techniques

URL of GitHub repository: https://github.com/ucl-comp0035/comp0034-cw2i-sirine-b.git

IV. References

Reference for the dataset used for in this Dashboard (GRAD:ME! Web app)I:

Name: Unistats dataset - employment trends (DV)

Author: HESA

License: CC-BY-4.0 You must cite the author in your work.

Location: https://www.hesa.ac.uk/support/tools-and-downloads/unistats

Use: Data visualisation

Description: Provides comparable sets of information about full- and part-time undergraduate courses. It is run by the Office for Students and is designed to meet the information needs of prospective students.