**Test Plan and Results**

**Overall Test Plan**

In testing the design, we will first test each component individually before combining the components and testing them together, e.g: testing user inputs, then system results, then testing user inputs and system results together. When testing our design we want to ensure that users have a straightforward way to use the application along with documentation and instruction.

Then, the following tests aim to ensure the successful integration and hosting of the database and website. The tests cover aspects including the selection of an appropriate SQL library for Python, validating database updates after user searches, verifying website hosting on a cloud platform, specifying a data caching method, and creating an HTML page handler using Python. These tests will contribute to the goal of achieving a well-integrated and functional database and website hosting solution with efficient data handling and caching capabilities.

The last set of test attempt to verify the data fetched, stored, and returned is in the format we anticipate, and that the data searched for is what is expected. This includes tests to ensure we are only retrieving the necessary data we want, and storing it in such a way that when the data has returned from the database it is still usable. Ultimately these tests will support the expected behavior of our program and ensure no erroneous behavior occurs during runtime within tolerance.

**Test Case Description**

UIUX1.1 **User Interface and Experience Test 1**

UIUX1.2 To ensure that the frontend design of the application is accessible.

UIUX1.3 Wireframes are the first step of our design process; designs and their respective tests will focus on page layout and user flow.

UIUX1.4 Inputs: Wireframes for application components

UIUX1.5 Outputs: Clean component layout and accessibility for users

UIUX1.6 Normal

UIUX1.7 Both

UIUX1.8 Functional

UIUX1.9 Integration

UIUX2.1 **User Interface and Experience Test 2**

UIUX2.2 This test will ensure that Eleventy’s performance is quick and lightweight.

UIUX2.3 Eleventy, by default, will throw a warning if pieces of the build take longer than 8% of the total build time. UIUX2.1 will look for components of the build that take the longest to complete; these results will then be analyzed and fixed.

UIUX2.4 Inputs: The built application.

UIUX2.5 Outputs: Performance measurements and aggregate benchmarks.

UIUX2.6 Normal

UIUX2.7 Whitebox

UIUX2.8 Performance

UIUX2.9 Integration

UIUX3.1 **User Interface and Experience Test 3**

UIUX3.2 This test will ensure that user interactions with the frontend work as expected.

UIUX3.3 The functionality of our search and return functions, in particular, work as expected within the layout of our application. For example, when a user interacts with the search component.

UIUX3.4 Inputs: Appropriate search data

UIUX3.5 Outputs: Appropriate result data

UIUX3.6 Normal

UIUX3.7 Whitebox

UIUX3.8 Functional

UIUX3.9 Unit

UIUX4.1 **User Interface and Experience Test 4**

UIUX4.2 This test will ensure that the results of a query, if the user wants, can be saved into an easily accessible frontend component for the user to refer back to.

UIUX4.3 The design and functionality of our cached results are easy for a user to enable as well as refer to.

UIUX4.4 Inputs: Appropriate result data stored in the cache.

UIUX4.5 Outputs: Cached data is displayed and accessible to user.

UIUX4.6 Normal

UIUX4.7 Whitebox

UIUX4.8 Functional

UIUX4.9 Integration

DIWH1.1 **Database Integration/Website Hosting Test 1**

* DIWH1.2 To ensure the selected SQL library for Python is successfully researched and identified.
* DIWH1.3 Research various SQL libraries for Python, such as SQLAlchemy, PyMySQL, and sqlite3, and choose the most suitable one.
* DIWH1.4 Inputs: List of potential SQL libraries for Python.
* DIWH1.5 Outputs: The finalized choice of the SQL library for Python.
* DIWH1.6 Normal
* DIWH1.7 Blackbox
* DIWH1.8 Performance
* DIWH1.9Integration Test
* DIWH1.10 Results: The chosen SQL library is successfully identified based on research and suitability for the project.

DIWH2.1 **Database Integration/Website Hosting Test 2**

* DIWH2.2 To validate that the database is updated with new records on each search.
* DIWH2.3 Perform search queries using a set of test data to simulate user searches and confirm that new records are added to the database.
* DIWH2.4 Inputs: Set of test data for search queries.
* DIWH2.5 Outputs: Confirmation that new records are added to the database after each search.
* DIWH2.6 Normal
* DIWH2.7 Whitebox
* DIWH2.8 Functional
* DIWH2.9 Integration
* DIWH2.10 Results: The database is successfully updated with new records after each search, as expected.

DIWH3.1 **Database Integration/Website Hosting Test 3**

* DIWH3.2 To investigate and verify the process of hosting the website on a cloud platform.
* DIWH3.3 Explore the chosen cloud platform (e.g., AWS, Azure, Google Cloud) and ensure successful website hosting.
* DIWH3.4 Inputs: Information about the selected cloud platform.
* DIWH3.5 Outputs: Successful hosting of the website on the chosen cloud platform.
* DIWH3.6 Normal
* DIWH3.7 Blackbox
* DIWH3.8 Functional
* DIWH3.9 Integration
* DIWH3.10 Results: The website is hosted successfully on the specified cloud platform, meeting hosting requirements.

DIWH4.1 **Database Integration/Website Hosting Test 4**

* DIWH4.2 To design and validate a method to cache data after a user has searched for it.
* DIWH4.3 Implement the caching method and test it with sample search data to ensure correct data caching.
* DIWH4.4 Inputs: Designed caching method and sample search data.
* DIWH4.5 Outputs: Confirmation that data is correctly cached after a user search.
* DIWH4.6 Normal
* DIWH4.7 Whitebox
* DIWH4.8 Functional
* DIWH4.9: Unit Test
* DIWH4.10 Results: The caching method is successfully designed and implemented, ensuring data is cached after user searches.

DIWH5.1 **Database Integration/Website Hosting Test 5**

* DIWH5.2 To create and verify an HTML page handler using Python.
* DIWH5.3 Develop a Python script for handling HTML pages and test it with sample HTML pages to ensure effective handling.
* DIWH5.4 Inputs: Python script for handling HTML pages and sample HTML pages.
* DIWH5.5 Outputs: Successful handling of HTML pages using the Python script.
* DIWH5.6 Normal
* DIWH5.7 Blackbox
* DIWH5.8 Functional
* DIWH5.9 Integration Test
* DIWH5.10 Results: The HTML page handler is successfully created and effectively handles HTML pages as intended.

DM1.1 **Data Management Test 1**

DM1.2 To verify the data we scrape off the web is parsed and formatted correctly

DM1.3 A test with the result of a simple search that is returned in a manner we expect. (No random characters / spliced links / unexpected html)

DM1.4 Method for grabbing results

DM1.5 A test result returned in a predetermined manner

DM1.6 Normal

DM1.7 Whitebox

DM1.8 Functional

DM1.9 Unit

DM2.1 **Data Management Test 2**

DM2.2 To check that filters can be applied to fetched results before being saved to database

DM2.3 A test the verify that we can, within a tolerance, filter out results based on filter criteria

DM2.4 A given filter for the search funciton

DM2.5 Out of a set amount of results returned less than 10% will contain the filtered content

DM2.6 Abnormal

DM2.7 Blackbox

DM2.8 Performance

DM2.9 Intergration

DM3.1 **Data Management Test 3**

DM3.2 To ensure that data retrieved from the database is in the format we expect it to be

DM3.3 Testing the results of set data retrieved from the database that it will be the way we expect it to be

DM3.4 The data from the database

DM3.5 The test data is returned fromatted in a way the frontend expects to receive it

DM3.6 Normal

DM3.7 Whitebox

DM3.8 Functional

DM3.9 Unit

DM4.1 **Data Management Test 4**

DM4.2 To ensure only necessary data is saved to the database and discarded when no longer needed.

DM4.3

DM4.4 Data the crosses beyond specific boundaries that marks it as valid for removal from the database

DM4.5 The data no longer exist in the database

DM4.6 Boundary

DM4.7 Blackbox

DM4.8 Functional

DM4.9 Integration

**Test Case Matrix**

|  | **Normal/Abnormal** | **Blackbox/Whitebox** | **Functional/Performance** | **Unit/Integration** |
| --- | --- | --- | --- | --- |
| **UIUX1** | Normal | Both | Functional | Integration |
| **UIUX2** | Normal | Whitebox | Performance | Integration |
| **UIUX3** | Normal | Whitebox | Functional | Unit |
| **UIUX4** | Normal | Whitebox | Functional | Integration |
| **DIWH1** | Normal | Blackbox | Performance | Integration |
| **DIWH2** | Normal | Whitebox | Functional | Integration |
| **DIWH3** | Normal | Blackbox | Functional | Integration |
| **DIWH4** | Normal | Whitebox | Functional | Unit |
| **DIWH5** | Normal | Blackbox | Functional | Integration |
| **DM1** | Normal | Whitebox | Functional | Unit |
| **DM2** | Abnormal | Blackbox | Performance | Integration |
| **DM3** | Normal | Whitebox | Functional | Unit |
| **DM4** | Boundary | Blackbox | Functional | Integration |