**What is Behavior-Driven Development?**

Behavior-Driven Development (BDD) is a software development methodology, often used for software testing, that aims to improve communication and collaboration between various stakeholders involved in the software development process. BDD came out as an extension of Test-Driven Development (TDD) and emphasizes the behavior of the software from the end-user’s perspective. BDD is a methodology that encourages cross-functional teams, including developers, testers, business analysts, and product owners, to work together more effectively. BDD focuses on bridging the gap between technical teams and non-technical stakeholders, achieved by using common language that everyone can understand in order to describe the desired behavior of the software.

**What is Behave?**

Behave is specifically designed for a Python-based framework used for BDD. The framework allows you to write and execute behavior-driven tests in a structured way that can easily be read by humans. Behave encourages collaboration between the technical and non-technical team members by providing common language to define and automate tests. Behave integrates smoothly with Python, uses Gherkin language for writing tests, and can be used with selenium.

**Required Packages**

Python - version 3.7 or higher

Open a command line interface inside of the ‘WebApp’ folder

Run the requirements.txt file to install all required packages

pip install -r requirements.txt

**Setting up Test Environment**

* Setting Up the Python Virtual Environment:
  + Navigate to the ‘WebApp’ folder and open a command line. Initiate your Python virtual environment by running:

python -m venv env

**Run the Flask Application**:

* Navigate to your ‘WebApp’ folder in the command line and run:

flask --debug --app main run

* If you receive the error ‘flask is not a recognized command’, alternatively run:

**python -m** flask --debug --app main run

* The application should now be running at <http://127.0.0.1:5000/> and will respond to changes you make in the code when you refresh the page. The flask application will need to be running for the lab to interact with it.

Learning Resources

**Official Behave Documentation**:

* + [Behave Documentation](https://behave.readthedocs.io/en/latest/): The official docs for Behave and how to use it.
  + [Official Behave GitHub Repo](https://github.com/behave/behave): The official Behave GitHub repository

**Community Forums and Support**:

* + [Stack Overflow](https://stackoverflow.com/questions/tagged/python-behave): Community Q&A on Behave-related topics.

**Writing Behave Tests**

**Understanding the file structure and Behave**:

* + In your *WebApp* folder, you will have a sub-folder called *Features*, this is where all Gherkin-based tests are stored for Behave in files noted as *.feature* files.
  + Inside the *Features* folder is another sub-folder called *Steps*, this is where the actual tests’ functionality goes in the form of Python scripts.
  + While multiple features can be put into a single file, they can also be split into multiple.
  + Similarly, steps can be combined into single files or split into multiple such as steps focusing on a specific action that will be reused often. (Ex: A step that opens the website to be tested).
  + For this lab, each scenario in the feature file will be broken down into its own step file.
  + Behave is a BDD **Python** Testing Framework. This means for web-interaction, a framework such as [Selenium Web-Driver](https://www.selenium.dev/documentation/webdriver/) or [Splinter](https://splinter.readthedocs.io/en/latest/) will need to be utilized. For this lab, Selenium Web-Driver will be used for all step-functionality.

**Basic Behave Test Structure**:

* + A base Behave feature written in Gherkin *given-when-then* looks like this:

A white background with black text

Description automatically generated

* + This feature’s corresponding steps would be written as:

A screen shot of a computer code

Description automatically generated

Test Examples:

Test 1 - Validate that only partial input for a new book shows and error:

A screenshot of a computer program

Description automatically generated

Test 2 – Sorting by the page column shows the smallest page count book:

This test sorts the page column, leaving the results from smallest to largest then ensures the smallest page count is the one at the top of the list.

A screenshot of a computer program

Description automatically generated

**Incomplete Test Scenarios (Exercises YOU need to Complete):**

**Test 3 (Incomplete) – Ensure the search bar only filters by titles**:

* + - Write a test that uses the search bar to search for something other than a title of a book and get no results.
    - Instruction: Fill in the search input with a value other than something present in a title of a book and ensure there are no results.

**Test 4 (Incomplete) – Add a new book**:

* + - Write a test that adds a new book and find it in the list after submission.
    - Instruction: Fill in the new book contents, submit the form, and check to see if the new book is present.

**Expected Results from Testing**

* Test 1: After partial input of a book, it will display an error instead of submitting the new book.

A screenshot of a computer

Description automatically generated

* Test 2: The smallest page book will be the top row of the list.

A screenshot of a computer

Description automatically generated

* Test 3: After searching for a value not in a title, there will be no results in the books list.
* Test 4: The newly added book will be present in the list.

**Running the Tests**

Execute your tests using the command:

behave

It will automatically run all feature files in the *Features* sub-folder. Observe the results to ensure all tests are functioning as expected. Your command line should indicate:

A black text on a white background

Description automatically generated

**Conclusion**

This lab should provide a good insight into the workings of the Behave BDD testing framework and how to implement it into a workflow. Behave allows business partners, QA testers, and development teams all get involved in the testing process to allow everyone to better understand how the program should function.