Test Automation as a Service (TAAS)

Author: Imran Abbas Satti

Overview

I have designed TAAS as centralized automation framework for performing automated testing different application deployment environments (test, stage, prod) over different platforms (web, android, IOS) over different testing layers (UI, API, Databases). All in one service.

I have automated some API test cases using python and BDD approach

Strategy and Plan

1. We have segregated the automation of application or system into on three layers.
   a. API layer
   b. Database layer
   c. UI (User Interface) layer
      i. Web
      ii. Android
      iii. IOS
      iv. Desktop

   Reason between segregating the system/app/service under test into these layer is that most of the time issues lie underneath API or database layer rather than UI layer. In addition, automation cost (time/efficiency) of UI layer is more than API.

2. We have followed BDD (Behavior Driven Development) approach using Gherkin to write the test cases in GIVEN, WHEN THEN form.

3. Each Test case is defined over three files
   a. Feature File
   b. Step File
   c. Logic File

   Feature file contains definition of test case in Gherkin format. In steps file we have function associated to feature file whereas the core logic is written in logic files.

   Reason behind this is to increase reusability and to make the architecture more flexible in terms of expansion or we can say “separation of concerns “

4. TASS architecture is based on **SOLID** principle of object orientation to avoid redundancy and maintenance cost

5. Xpaths and other dependencies are kept in common place. In same way we have placed the page logic by defining a class for each page, to follow the **page object model** approach

6. We have utilized exception handling and stored all of the exceptions in the centralized repository like a files or database.

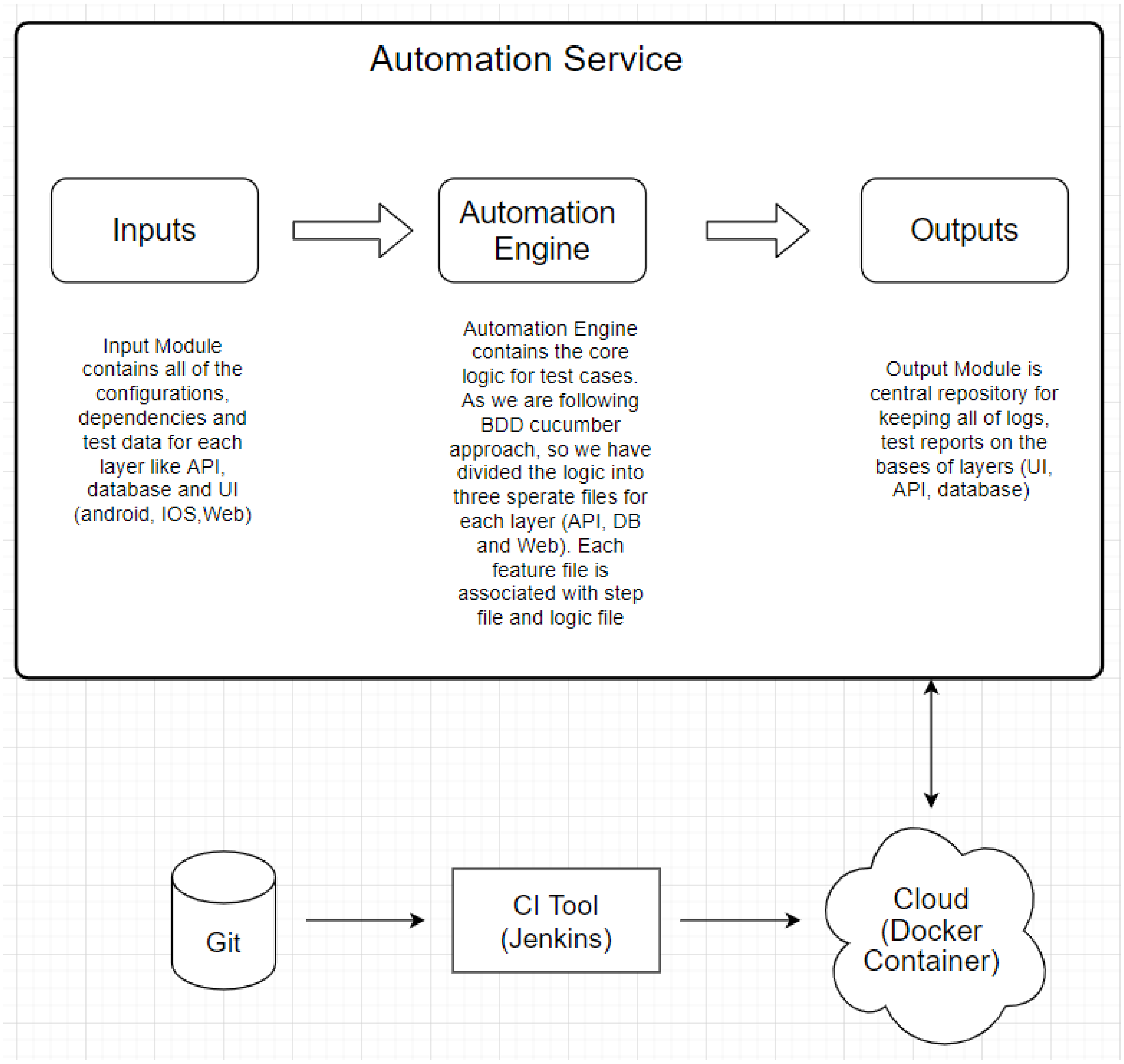
7. We are maintaining framework logs in centralized location for not only backtracking and debugging of failed test cases but also monitoring the health of TAAS.
   Reason behind is that suppose when we have to ran thousands of test cases simultaneously and a single test case gets failed then we have to back track either its due functionality crash or automation failure. This approach will make backtracking easier.

8. Output of test cases can be stored in database or CSV files. Based on test result we can be develop centralized UI dashboard (node.js), which will show output in different visualization. We can provide filters to create custom reports based on number of passed test cases, failed one, high priority, low priority, passed regression test cases, failed lower priority test cases etc.

9. We can integrate our test case with CI/CD pipeline. The Jenkins will trigger a testing job while pushing the development code from development environment to test environment. Testing job will get the latest automated code from git repo and it will deploy it to the testing server and after job completion, it will return pass or fail flag. If the flag is true then developer code would be pushed to staging environment or otherwise revert.

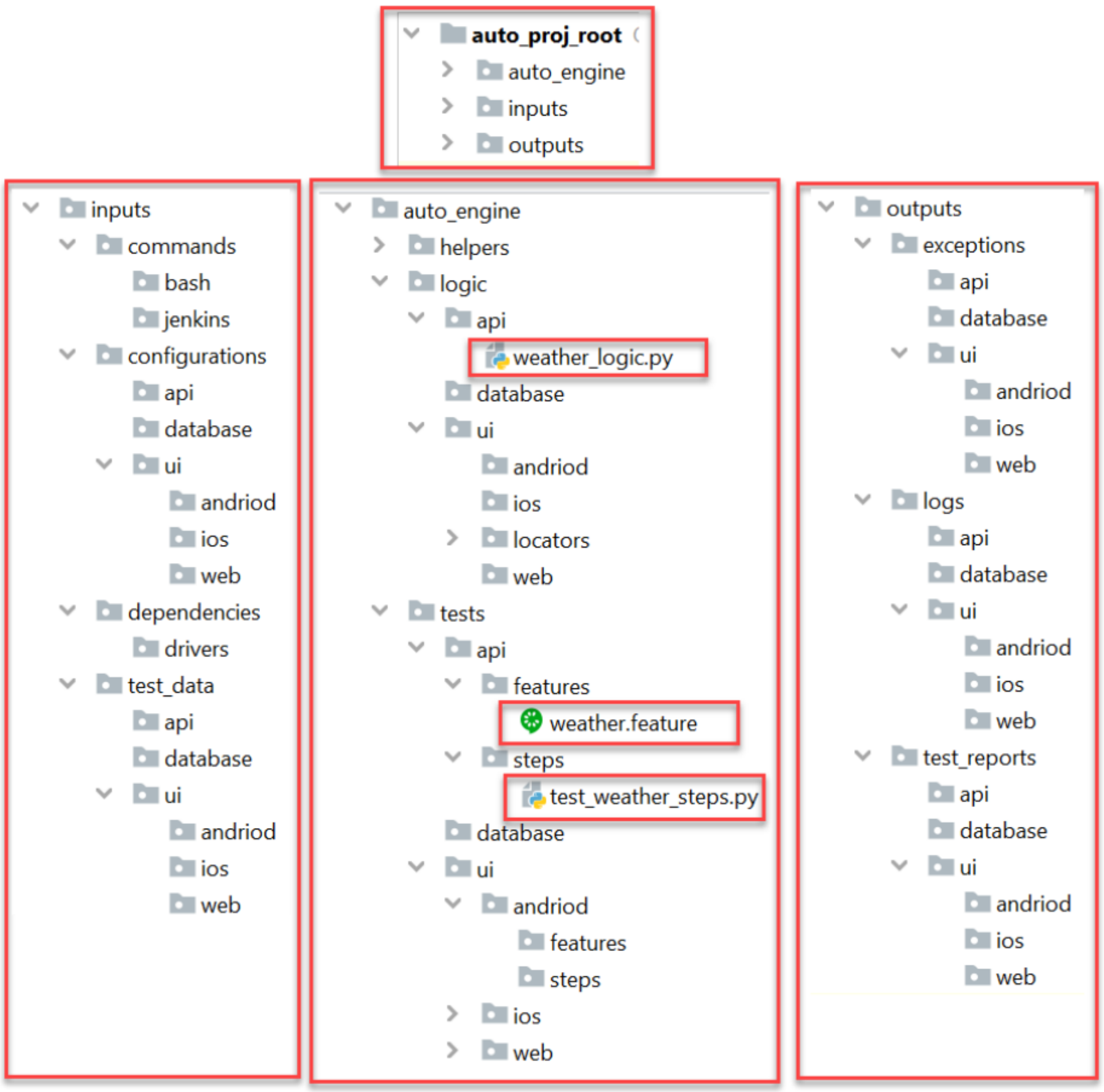
Modular Level Architecture

TAAS is divided into three modules namely inputs, automation engine and output. Each module have its own unique responsibility and a defined dependency, as shown in figure below


Automation Service
Inputs
Automation Engine
Outputs
Input Module contains all of the configurations, dependencies and test data for each layer like API, database and UI (android, IOS,Web)
Automation Engine contains the core logic for test cases. As we are following BDD cucumber approach, so we have divided the logic into three sperate files for each layer (API, DB and Web). Each feature file is associated with step file and logic file
Output Module is central repository for keeping all of logs, test reports on the bases of layers (UI, API, database)
Git
CI Tool (Jenkins)
Cloud (Docker Container)

Code Level Architecture

Below is my proposed architecture. In this architecture, we will have three core components inputs, automation engine and outputs. Automation engine will take inputs and process them to produce outputs.



```
auto_proj_root
    auto_engine
    inputs
    outputs
```

```
inputs
    commands
        bash
        jenkins
    configurations
        api
        database
        ui
            andriod
            ios
            web
    dependencies
        drivers
    test_data
        api
        database
        ui
            andriod
            ios
            web
```

```
auto_engine
    helpers
    logic
        api
            weather_logic.py
        database
        ui
            andriod
            ios
            locators
            web
    tests
        api
            features
                weather.feature
            steps
                test_weather_steps.py
        database
        ui
            andriod
                features
                steps
            ios
            web
```

```
outputs
    exceptions
        api
        database
        ui
            andriod
            ios
            web
    logs
        api
        database
        ui
            andriod
            ios
            web
    test_reports
        api
        database
        ui
            andriod
            ios
            web
```

Implementation

For implementation, we can choose any technology stack like python, java, C# or JavaScript frameworks. Currently I have implemented it using python.

According to my experience python and java have more support regarding to automation libraries but again it also depends on the application/system under test, in some cases we can achieve more efficient results by using cypress over selenium and vice versa.

Considering python as implementation language then we can utilize following technology stack for TAAS.

- **selenium** library for web automation
- **appium** library for android and IOS app automation
- **request** and **pact-test** library for API testing
- **pandas** and **sql libraries** for data testing
- **py-test** library as test runner and **pytest-bdd** library for following gherkin approach
- **bash** scripting for creation of testing job
- **linux** server (Cent-OS/Ubuntu) with **docker** for deployment of automation script
- **git** repository for holding automation code
- **Jenkins** for scheduling and triggering automation job
- **Node.js** for creating dashboard page for displaying and filtering test results
- **Python smptp** module for sending emails to stake holders
- **Pycharm** as IDE (Integrated Development Environment)

Exploring Code

Find the code in directory named **auto_service**

As I have implemented the complete architecture for **TASS**. It will contain some empty python modules.

In this assignment I have automated two API test cases, so you will find implementation files [feature, steps, logic] by looking directly into following directories with below sequence

1. auto_engine/tests/api/features/weather.feature
2. auto_engine/tests/api/steps/test_weather_steps.py
3. auto_engine/logic/api/weather_logic.py