Medifind Test Plan

Scope:

The test plan is designed to cover the following:

1. White Box Basis Path and Data Flow Testing
    a. This will be done to test the alternative paths data can take within the different endpoints
2. Black Box testing of all the endpoints
    a. All the backend endpoints shall be tested for accepting the expected input data, behaving properly (i.e. valid data to success and exception handling), and generating expected output correctly
    b. The inputs passed will also include edge cases and alternative cases to perform exhaustive blackbox testing and see if any boundary value analysis needs to be performed.
3. The test plan will eventually cover all high level methods and the individual units composing those methods in the backend endpoints

Approach:

1. Each backend endpoint of the ASP.NET Core application will be treated as an individual unit. Since the structure of the codebase is such that controller methods are invoked to eventually call CRUD methods on the Postgresql database, testing the controllers will ensure that all the subsequent methods (units) called are working properly.
2. Driver classes to be created for each type of unit test. These driver classes will then pass data (including edge cases) to test the unit under different possible scenarios. Stubs to be used wherever possible and/or necessary.
3. Integration tests to be performed on subunits that form clusters. May not be necessary for every endpoint in our usecase due to the specific type of structure of code achieved.
4. Regression testing to be performed after any modification to a unit/endpoint. Tests to be adjusted accordingly.
5. After the aforementioned tests have been completed and the product is in a state suitable for validation testing, a series of validation tests to be performed on the deployed system (Alpha testing).
6. Beta testing to be considered at a later stage to gauge User Experience reviews and identify any missed test cases.
7. System testing of minimally the following type to be performed:
    a. Script injections to test security
    b. Performance testing using Chrome Dev Tools and identifying areas of performance improvement
8. Interface testing for front-end by a group of users to test content, navigation and front-end components.
9. The order listed here to be followed as the schedule to be followed and the order of priority.
10. Backend endpoints to be tested by the development team. Frontend tests, Beta tests, and Interface testing to be performed by a set of users.

Validation Tests for 3 main use cases:

1. **Search by QR code**
    a. Scanning QR code invokes the endpoint api/drugs/{drugid}
    b. Therefore, is equivalent to search by drug id
    c. Test the unit of searchDrugById in the Drug Controller class. This would include edge cases such as invalid IDs and invalid datatypes.
    d. Test the functionality by scanning the QR code and inspecting the path the data takes
    e. Passing criteria to be set at properly handling edge cases, exceptions, and providing the correct Drug information page upon scan
2. **Search by name, ingredients, and/or manufacturer**
    a. The search invokes SearchDrug method in the drug controller
    b. Test the unit using driver and stubs
    c. Test the following cases:
        i. All parameters present
        ii. Only one parameter present, rest null
        iii. Handling case-insensitivity
        iv. Wrong datatypes (non-string)
    d. Test the functionality by using the web interface as well as making a get request directly from the browser and inspect the path taken and the response payload
    e. Passing criteria to be set at properly handling edge cases, exceptions, and providing the correct Drug informations
3. **Add drugs to database**
    a. This invokes an eventual Create operation on the database if the prerequisites are met
    b. Method should only perform a create operation if the following prerequisites are met:
        i. User is logged in
        ii. User has admin permissions
        iii. Required fields of a drug are non-null in the data passed
    c. Multiple paths may be invoked within a method depending on the state of the system.
    d. If the user is not logged in i.e. session_id does not exist:
        i. Raise a badhttprequest exception
    e. If the user does not have admin permissions:
        i. Raise a badhttprequest exception
    f. If the authorization header contains garbage or invalid value:
        i. Raise a badhttprequest exception
    g. If the user is logged in and has admin privileges:
        i. Check if drug to be added has non-null values for required fields
        ii. Raise an exception if not, create drug in database otherwise
    h. The above paths to be tested using post requests by CLI as well as the deployed application

i. Passing criteria to be set at properly handling edge cases, exceptions, and providing the correct Drug informations

In addition to the above mention 3 main usecases, validation and unit tests to be performed in a similar way for all the endpoints in the UserController and DrugController classes.

Current Status:

- A detailed log of the executed test plan to be available in the Github repository by the deadline of project submission so that the final version is tested and the logs are up to date.
- The current status is logged below:

```
Starting test execution, please wait...
A total of 1 test files matched the specified pattern.

Passed!  - Failed:     0, Passed:    39, Skipped:     0, Total:    39, Duration: 1 s - MedifindTests.dll (net6.0)
```

- With all the tests written up until now and the different scenarios, a total of 35 tests performed. All tests pass as of now and were passing when they were written.
- The only issue encountered was while testing the following:
    - SignUpTest
    - SaveDrug to user profile test
    - Both of these tests would create an entry in the database and then raise an exception when the same tests were run again because of unique_value constraints being violated by creating the same entry again.
    - This was resolved by removing the newly created entry for testing purposes as soon as the test was done.
- The tests can be viewed in the UserTests directory from the root of the project repository.