**Test Plan /Approach**

**PetStore**

**Microservices application**

Author: Rohan Jamdar

The Petstore microservice Business Requirements can be divided into 3 broad categories:

1. **Operations about a pet – microservice (MS1)**

Application should allow the following functionalities:

1. Adding a new pet
2. Updating details about an existing pet
3. Search / Find pets by their status
4. Search / Find a pet by its ID
5. Updating name and status about an existing pet
6. Delete the pet ID from the data store
7. Uploads/updates photos of pet
8. **Store (Admin) operations – microservice (MS2)**

Application should allow the following functionalities:

1. Return all pet inventories by status
2. Admin to place an order for a pet
3. Search / Find a PO by its ID
4. Delete the PO by its ID
5. **User operations – microservice (MS3)**

Application should allow the following functionalities:

1. Creates a single user
2. Create list of users (list and array)
3. Login functionality
4. Logout functionality
5. Search user by username
6. Update user details functionality
7. Delete user functionality

**Acceptance Criteria**

**MS1: Operations about a pet**

**Acceptance Criteria:**

1. User should be able to add a new pet successfully with valid inputs of category, name, photo-URL, tag and status.
2. User should update details of an existing pet successfully. For e.g. details like category, name, photo-URL, tag and status
3. User should be able to successfully search pets by its status in the application. Status values can be ‘Available’, ‘Pending’ and ‘Sold’
4. User should be able to successfully search a pet by its ID. Assumption is pet ID is known to the user, as it is a required field.
5. User should be able to successfully update name and status of an existing pet. Assumption is pet ID is known to the user, as it is a required field.
6. User should be able to successfully delete the pet information from the data store using the pet ID. Assumption is pet ID is known to the user, as it is a required field.
7. User should be able to successfully upload the photo of a pet. Again, assumption is pet ID is known to the user, as it is a required field.

In all the above cases we have to ensure that if invalid inputs are given then the appropriate error codes and messages are returned.

In all above cases we have to ensure that if no inputs are given then the appropriate error codes and messages are returned.

**MS2: Store (Admin) operations**

**Acceptance Criteria:**

1. Verify that admin user is able to get all the pet inventories by status
2. Verify that an admin is able to successfully place an order for a pet, by providing all the required fields.
3. Verify that and admin user is able to Search / Find a purchase order by ID. Assumption is order ID is known to the user, as it is a required field.
4. Verify that an admin user is able to Delete the purchase order by ID. Assumption is order ID is known to the user, as it is a required field.

In all the above cases we have to ensure that if invalid inputs are given then the appropriate error codes and messages are returned.

In all above cases we have to ensure that if no inputs are given then the appropriate error codes and messages are returned.

**MS3: User operations**

**Acceptance Criteria:**

1. Verify that a single user can be successfully created by an already logged in user.
2. Verify that a list of users can be created.
3. Verify that a valid user can login successfully.
4. Verify that a logged in user can logout out successfully.
5. Verify that a logged in Admin user is able to search for a user by its by username. Assumption is that the username of the person is already known, as it is a required field.
6. Verify that a logged in Admin user is able to successfully update user details. Assumption is that the username of the person is already known, as it is a required field.
7. Verify that if a logged in user is able to successfully delete an existing user. Assumption is that the username of the person is already known, as it is a required field.

In all the above cases we have to ensure that if invalid inputs are given then the appropriate error codes and messages are returned.

In all above cases we have to ensure that if no inputs are given then the appropriate error codes and messages are returned.

**Based on the architecture of each microservice, our testing strategy should include the following:**

1. Verify that the application protocol exposed by the service, does its job of checking the request and providing a protocol specific response according to the outcome of the business transaction.
2. All our microservices have to be tested for the logic of how they communicate with the external services - the communication through the gateway and the http client.
3. Also, our microservices should be tested for connections out to external services and should be resilient to outages of remote components.
4. The microservice will also have to be tested for the logic on how the connection is made with the data store as it exists over a network boundary incurring latency and risk of outage.
5. **RISK:** The presence of network partitions affects the style of testing employed. Tests of these modules can have longer execution times and may fail for reasons outside of the team's control.

**Types of testing that should be undertaken:**

1. **Unit testing:**

The intention of unit tests should be to test the behaviour of the microservices class or method separately. We should have as many unit tests as possible to cover all the paths that the feature is going to provide. From a cost perspective these are the cheapest to fix and should be as exhaustive as possible.

1. **Integration testing:**

An integration test will verify the communication paths and interactions between components to detect interface defects.

1. Verify that the calling service terminates its call and provides appropriate message if the response from the called service takes too long. Also, the called service sends a response that indicates overload and signals that the calling service should reduce or redirect its load.

Verify that the calling services have a graceful way to handle a nonresponsive called service. Error handling should also be tested to ensure the service and protocol client employed respond as expected in exceptional circumstances. If the information that the called service is supposed to return is unavailable, then the calling service should still be able to accept that the called service will not respond, and continue to serve up useful, if incomplete, information. It can be beneficial to use a stub version of the external component as a test harness which can be configured to fail in predetermined ways.

1. Services must be able to spawn and kill new service instances as needed to accommodate traffic load variations through auto-scaling.
2. Gateway integration tests allow any protocol level errors such as missing HTTP headers, incorrect SSL handling or request/response body mismatches.
3. **RISK:** State management can be difficult when testing against external components. For this we have to agree on a fixed set of data that is guaranteed to be available in every environment.
4. Since most data stores exist across a network partition, they are also subject to timeouts and network failures. Integration tests should attempt to verify that the integration modules handle these failures gracefully.

**3) Component testing:**

We will descope the exercised software to a portion of system under test.

In our case the components are the services MS1, MS2 and MS3. By writing tests at this granularity, the contract of the API can be driven through tests from the perspective of a consumer. Isolation of the service can be achieved by replacing external collaborators with test doubles and by using internal API endpoints to configure the service.

While it is possible to configure test doubles and setup data directly when writing acceptance tests, routing all requests through privileged internal resources allows the service to be tested as more of a black box. This allows changes in external service communications to take place without impacting the component test suite.

**4) End-To-End testing:**

End-to-end tests will provide value by adding coverage of the gaps between the services. This gives additional confidence in the correctness of messages passing between the services and also ensures that firewalls and proxies are correctly configured.

Over a period of time, end-to-end tests will help us to evolve our microservice architecture as more is learnt about the problem domain.

**Few Gherkin test cases:**

**1) User creation:**

**Given: User wants to register a user**

**When: api is requested for user creation**

**Then: create user AND set ‘username’ on new user**

**2) Login functionality**

**Given: user is already on login page**

**When: when title of login page is petstore**

**Then: user enters username and password**

**Then: user clicks on login button**

**And: user is on home page**

1. **Logout functionality**

**Given: user is already on logged in on homepage**

**When: when title of login page is petstore**

**Then: user clicks on logout button**

**And: user is on login page**