Use Case Testing

Title: Create New Account
Actors: User
Main Scenario:

1. User chooses the “Create New Account” button.
2. System transfers to the “Create New Account” interface; display “First Name”, “Last Name”, “Username”, “Create Password”, and “Verify Password” text input fields.
3. User inputs information into necessary text fields.
4. System verifies password.
5. Inputted information saved to “Login” repository in database.
6. System transfers to the “Travel Logs” interface.

Alternatives:

2a. System does not transfer to "Create New Account" interface
2a1. User cannot create new account and receives an error message
3a. User does not input necessary fields
3a1. User cannot create new account and empty fields are highlighted
4a. System produces a false negative in password verification
4a1. User cannot create new account and receives an error message
5a. System fails to store account information
5a1. User cannot create a new account and receives an error message
6a. System fails to redirect user to "Travel Logs" interface
6a1. User is not signed in and receives an error message

Test Situations:

1. User can create account successfully
2. User does not fill out all required fields
3. User does not enter a valid password
4. User's first password entry does not match second (verification) entry

Test Coverage:

4/5 = 80% test coverage

Title: Login
Actors: User
Main Scenario:

1. User chooses the "Login" button.
2. System transfers to the "Login" interface; display "username" and “password” text input fields.
3. User inputs information into text input fields
4. System verifies inputted information matches stored information from the database.

5. System transfers to the "Travel Logs" interface.

Alternatives:

2a. System does not transfer to "Login" interface
2a1. User cannot login and receives an error message
3a. User does not input necessary information
3a1. User cannot login and receives an error message
4a. Username cannot be found in database
4a1. User cannot login and receives an error message
4b. Password does not match username
4b1. User cannot login and receives an error message
5a. System fails to redirect to "Travel "Travel Logs" interface
5a1. User cannot login and receives an error message

Test Situations:

1. User can login successfully
2. User does not fill out all required fields
3. User enters invalid username
4. User enters valid username and invalid password

Test Coverage:

4/5 = 80% test coverage

Title: Search Based on Criteria
Actors: User
Main Scenario:

1. User selects the "Search" button.
2. System transfers to the "Exploratory" interface; display the search field and search criteria options.
3. User inputs item they're searching for.
4. System displays options that fit the type of item the user entered.
5. User checks additional criteria options.
6. System removes items from display that don't meet new criteria.

Alternatives:

2a. System does not transfer to "Search Based on Criteria" interface
2a1. User cannot search based on criteria and receives an error message
3a. User does not input any criteria
3a1. User cannot search based on criteria and receives an error message
4a. System does not display filtered items
4a1. User cannot search based on criteria and receives an error message
6a. System does not update to display items with new filter
6a1. User cannot search based on criteria and receives an error message

Test Situations:

1. User can successfully filter based on criteria

2. User can successfully edit filter based on criteria
3. User does not input any criteria

Test Coverage:

3/4 = 75% test coverage

Title: Add Log
Actors: User
Main Scenario:

1. User selects “New Log” button
2. New travel log created in the "Travel Log" directory in the database.
3. System transfers to the “New Log” interface; displays empty travel log with the option to add events

Alternatives:

2a. System fails to store new log in database
3a1. User cannot add travel log and receives an error message
3a. System does not transfer to "New Log" interface
3a1. User cannot add travel log and receives an error message

Test Situations:

1. User can successfully add travel log
2. User does not enter travel log details

Test Coverage:

1/2 = 50% test coverage

Title: Add Media
Actors: User
Main Scenario:

1. User chooses "Add Photo/Video" in a travel log.
2. System requests media permission from the user.
3. User grants permission to the system.
4. System accesses media on the device.
5. User selects media to add.
6. Selected media is stored in the "Travel Log" directory in the database.

Alternatives:

2a. System does not request media permission from user
2a1. User cannot add media and receives an error message
3a. User does not grant media permission to system
3a1. User cannot add media and receives an error message
4a. System does not access media on device
4a1. User cannot add media and receives an error message
5a. User does not select media to add

5a1. User cannot add media and receives an error message
6a. System fails to store selected media in database
6a1. User cannot add media and receives an error message

Test Situations:

1. User can successfully add photo to travel log
2. User can successfully add video to travel log
3. User does not grant media permission
4. User does not select media too add to travel log

Test Coverage:

4/5 = 80% test coverage

Unit Case Testing

Title: Verify Login

Description:

This test will determine that a user can login successfully with a valid username and password

Partitions:

Login (valid):
- Username exists
- Password == password associated w/ username

Login (invalid):
- Username does not exist
- Password =/= password associated w/ username

Test Outline:

```
Public Class test() {
    @Test
    Public Void testValidLogin(){
        User testUser = User();
        testUser.setPassword("testPassword1");
        testUser.setUsername("testUsername");

        String enteredPassword = "testPassword1";
        String enteredUsername = "testUsername";

        assertSametestUser.getPassword(), enteredPassword);
        assertSame(testUser.getUsername(), enteredUsername);
    }

    @Test
    Public Void testInvalidLogin(){
        User testUser = User();
        testUser.setPassword("testPassword");
        testUser.setUsername("testUsername");

        String enteredPassword = "testPassword2";
        String enteredUsername = "testUsername2";

        assertNotSame(testUser.getPassword(), enteredPassword);
        assertNotSame(testUser.getUsername(), enteredUsername);
    }
}
```

Title: Valid Email
Description:
This test will determine if an email follows the format of a valid email address.
Partitions:

Email (valid):	Email (invalid):
one '@'	no '@'
one '.'	no '.'

Test Outline:
```
Public Class test() {
    @Test
    Public Void testValidEmail() {
        String testEmail = "test@email.com"

        assertTrue(testEmail.contains("@"));
        assertTrue(testEmail.contains("."));
    }

    @Test
    Public Void testInvalidEmail() {
        String testEmail = "testemailcom"

        assertFalse(testEmail.contains("@"));
        assertFalse(testEmail.contains("."));
    }

}
```

Title: Valid Password
Description:
This test will determine if the password entered by a user is valid. A valid password will be at least eight characters, at least one character, and at least one number.
Partitions:

Password (valid):	Password (invalid):
length >= 8	length < 8
at least one number	no number
at least one letter	no letter

Test Outline:

```
Public Class test() {
    @Test
    Public Void testValidPassword(){
        String password = "password1";

        assertTrue(password.contains([a-zA-Z]);
        assertTrue(password.contains([0-9]);
        assertTrue(password.len >= 8);
}
```

Title: Password Match

Description:

This test will determine if the first password entry and second password entry matches.

Partitions:

Password Match (valid): first password == second password	Password Match (invalid): first password =/= second password

Test Outline:

```
Public Class test() {
    @Test
    Public Void testPasswordMatch() {
        String password1 = "testPassword1";
        String password2 = "testPassword1";

        assertSame(password1, password2);
    }

    @Test
    Public Void testNotPasswordMatch() {
        String password1 = "testPassword1";
        String password2 = "testPassword2";

        assertNotSame(password1, password2);
    }
}
```

Title: Add new log

Description:

This test will determine that a user can add a log to the log list

Partitions:

Log information(valid):	Log information(invalid):
Log name >= 1	length = 0, NULL
Log details >= 1	Log name = 0, NULL
	Log name = 0, NULL

Test Outline:

```
Public Class test () {
@Test
Public Void testAddLog(){
    User testUser = user();
    testUser.addLog("testLogName","testLogDetails");
}
```

Title: Remove a log

Description:

This test will determine that a user can remove an existing log from the log list

Partitions:

Log information(valid):	Log information(invalid):
Log name >= 1	length = 0, NULL
Log details >= 1	Log name = 0, NULL
	Log name = 0, NULL

Test Outline:

```
Public Class test () {
@Test
Public Void testAddLog(){
    User testUser = user();
    testUser.addLog("testLogName","testLogDetails");

    testUser.removeLog(0);
}
```

Title: Create new account

Description:

This test will determine that a user can create a new account with a valid email address, username and password

Partitions:

Password (valid):	Password (invalid):
length >= 8	length < 8
at least one number	no number
at least one letter	no letters
Email (valid):	Email (invalid):
one '@'	no '@'
one '.'	no '.'

Test Outline:

```
Public Class test() {
    @Test testValidSignUp() {
        User testUser = User();

        String enterPassword = testUser.enterPassword("testPassword1");
        String confirmPassword =
        testUser.enterConfirmPassword("testPassword1");
        String enteredEmail = testUser.enter("testEmail");

        assertSame(enterPassword, confirmPassword);
        assertTrue(password.contains([a-zA-Z]);
        assertTrue(password.contains([0-9]);
        assertTrue(password.len >= 8);
        assertTrue(testEmail.contains("@"));
        assertTrue(testEmail.contains("."));
    }

}
```

Title: Search for criteria

Description:

This test will determine that a user can complete a search

Partitions:

Search(valid):	Search(invalid):
Search query length >= 1	length = 0, NULL
Search query length >= 1	Search query length = 0, NULL
	Search query length = 0, NULL

Test Outline:

```
Public Class test() {
@Test
Public Void testValidSearch(){
    User testUser = User();
    testUser.search(“testQuery”);
}
```

Title: Set budget

Description:

This test will determine that a user can set a budget for a trip.

Partitions:

Budget (valid):	Password (invalid):
value >= 1, <= 2147483647	Value <= 0, > 214748364

Test Outline:

```
Public Class test() {
@Test
Public Void testValidLogin(){
    User testUser = User();
    testUser.addLog(“testLogName”,”testLogDetails”);
    testUser.setBudget(“testLogName”,“999”);
}
```

Title: Update budget

Description:

This test will determine that a user can update a budget for a trip.

Partitions:

Budget (valid):
value >= 1, <= 2147483647

Password (invalid):
Value <= 0, > 2147483647

Test Outline:

```
Public Class test() {
@Test
Public Void testValidLogin(){
    User testUser = User();
    testUser.addLog(“testLogName”,”testLogDetails”);
    testUser.updateBudget(“testLogName”,“999”)
}
```