Assignment: Test the functionality of a fitness tracking mobile app.

Comprehensive Test Plan :

The below features are able to make look easier for a fitness tracking mobile app

INTRODUCTION

Purpose of the test plan - To ensure that the features and functionalities that are considered for the app work as expected . To Verify that the app is user-friendly and easy to navigate.

Overview -
- Logging Workouts
- Tracking Progress
- Goal Setting
- Social parameters/ Features - Connections
- List of exercises & workouts (library)
- Integration with other fitness apps & devices

Scope of Testing - This fitness mobile app should cover all features and functionalities that are included.
- User Interface
- Functional Testing
- Performance Testing
- Security Testing
- Compatibility Testing
- Usability Testing
- Regression Testing
- Localization Testing

Assumptions -
- This app is designed to work on all major platforms , including Android & iOS.
- Designed to work with wide range of fitness devices and apps (fitness trackers, heart rate monitoring, nutrition tracking apps)
- Designed to be user-friendly with a simple interface , to be secure and protect the data , to be scalable and handle a large number of users.

Limitations -
- This app may not work properly on older devices.
- This app may not be able to connect with all apps or devices , depending on device compatibility.
- Accuracy varies with the environmental conditions.

OBJECTIVES :

Functional Testing Objectives -

- ➢ To ensure that the app is able to accurately log and display user workouts , including sets, exercises and reps.
- ➢ And also to be able to track user progress overtime and to provide relevant and accurate feedback to users based on their goals and performance.

1) Log in and signup functionality -
The following code is for testing the login and sign up functionality of the fitness tracking mobile app

```
import unittest
from appium import webdriver

class FitnessAppTests(unittest.TestCase):

    def setUp(self):
        # Set up desired capabilities for Appium server
        desired_caps = {}
        desired_caps['platformName'] = 'Android'
        desired_caps['platformVersion'] = '11.0'
        desired_caps['deviceName'] = 'emulator-5554'
        desired_caps['appPackage'] = 'com.fitness.app'
        desired_caps['appActivity'] = 'com.fitness.app.MainActivity'

        # Set up Appium server and launch the app
        self.driver = webdriver.Remote('http://localhost:4723/wd/hub', desired_caps)
        self.driver.implicitly_wait(10)

    def test_sign_up(self):
        # Click on the sign up button
        sign_up_button = self.driver.find_element_by_id('sign_up_button')
        sign_up_button.click()

        # Enter user details in the sign up form
        name_field = self.driver.find_element_by_id('name_field')
        name_field.send_keys('John Doe')

        email_field = self.driver.find_element_by_id('email_field')
        email_field.send_keys('johndoe@gmail.com')
        password_field = self.driver.find_element_by_id('password_field')
        password_field.send_keys('test1234')

        # Submit the sign up form
        submit_button = self.driver.find_element_by_id('submit_button')
        submit_button.click()

        # Assert that the user is redirected to the home page
        home_page_title = self.driver.find_element_by_id('home_page_title')
        self.assertEqual(home_page_title.text, 'Welcome, John!')

    def test_log_in(self):
        # Click on the log in button
        log_in_button = self.driver.find_element_by_id('log_in_button')
        log_in_button.click()

        # Enter user credentials in the log in form
        email_field = self.driver.find_element_by_id('email_field')
        email_field.send_keys('johndoe@gmail.com')

        password_field = self.driver.find_element_by_id('password_field')
        password_field.send_keys('test1234')

        # Submit the log in form
        submit_button = self.driver.find_element_by_id('submit_button')
        submit_button.click()

        # Assert that the user is redirected to the home page
        home_page_title = self.driver.find_element_by_id('home_page_title')
        self.assertEqual(home_page_title.text, 'Welcome back, John!')

    def tearDown(self):
        # Quit the driver and close the app
        self.driver.quit()

if __name__ == '__main__':
    unittest.main()
```

2) Logging workouts , including exercises , sets & reps -
The following code is for testing the Logging workouts , including exercises , sets & reps

```
import unittest
from appium import webdriver
```

```
class FitnessAppTests(unittest.TestCase):

  def setUp(self):
    # Set up desired capabilities for Appium server
    desired_caps = {}
    desired_caps['platformName'] = 'Android'
    desired_caps['platformVersion'] = '11.0'
    desired_caps['deviceName'] = 'emulator-5554'
    desired_caps['appPackage'] = 'com.fitness.app'
    desired_caps['appActivity'] = 'com.fitness.app.MainActivity'

    # Set up Appium server and launch the app
    self.driver = webdriver.Remote('http://localhost:4723/wd/hub', desired_caps)
    self.driver.implicitly_wait(10)

  def test_log_workout(self):
    # Log in to the app (assumes valid credentials)
    email_field = self.driver.find_element_by_id('email_field')
    email_field.send_keys('johndoe@gmail.com')

    password_field = self.driver.find_element_by_id('password_field')
    password_field.send_keys('test1234')

    submit_button = self.driver.find_element_by_id('submit_button')
    submit_button.click()

    # Navigate to the log workout page
    log_workout_button = self.driver.find_element_by_id('log_workout_button')
    log_workout_button.click()

    # Enter details of the workout
    exercise_field = self.driver.find_element_by_id('exercise_field')
    exercise_field.send_keys('Bench press')

    set_field = self.driver.find_element_by_id('set_field')
    set_field.send_keys('3')

    rep_field = self.driver.find_element_by_id('rep_field')
    rep_field.send_keys('10')

    # Submit the workout log
    submit_button = self.driver.find_element_by_id('submit_button')
    submit_button.click()
```

```
        # Assert that the workout is logged successfully
        workout_summary = self.driver.find_element_by_id('workout_summary')
        self.assertEqual(workout_summary.text, 'Bench press - 3 sets x 10 reps')

    def tearDown(self):
        # Quit the driver and close the app
        self.driver.quit()

if __name__ == '__main__':
    unittest.main()
```

3) Tracking progress overtime and setting goals for fitness and nutrition -
The following code is for testing the tracking progress overtime and setting goals for fitness and nutrition

```
import unittest
from appium import webdriver

class FitnessAppTests(unittest.TestCase):

    def setUp(self):
        # Set up desired capabilities for Appium server
        desired_caps = {}
        desired_caps['platformName'] = 'Android'
        desired_caps['platformVersion'] = '11.0'
        desired_caps['deviceName'] = 'emulator-5554'
        desired_caps['appPackage'] = 'com.fitness.app'
        desired_caps['appActivity'] = 'com.fitness.app.MainActivity'

        # Set up Appium server and launch the app
        self.driver = webdriver.Remote('http://localhost:4723/wd/hub', desired_caps)
        self.driver.implicitly_wait(10)

    def test_track_progress_and_set_goals(self):
        # Log in to the app (assumes valid credentials)
        email_field = self.driver.find_element_by_id('email_field')
        email_field.send_keys('johndoe@gmail.com')

        password_field = self.driver.find_element_by_id('password_field')
        password_field.send_keys('test1234')

        submit_button = self.driver.find_element_by_id('submit_button')
        submit_button.click()
        # Navigate to the progress tracking page
        progress_tracking_button = self.driver.find_element_by_id('progress_tracking_button')
        progress_tracking_button.click()

        # Set a fitness goal
        fitness_goal_field = self.driver.find_element_by_id('fitness_goal_field')
        fitness_goal_field.send_keys('Run a 5K in under 30 minutes')

        set_goal_button = self.driver.find_element_by_id('set_goal_button')
        set_goal_button.click()

        # Verify that the goal is set successfully
        fitness_goal_summary = self.driver.find_element_by_id('fitness_goal_summary')
        self.assertEqual(fitness_goal_summary.text, 'Run a 5K in under 30 minutes')

        # Log progress toward the goal
        log_progress_button = self.driver.find_element_by_id('log_progress_button')
        log_progress_button.click()

        progress_field = self.driver.find_element_by_id('progress_field')
        progress_field.send_keys('Ran 3 miles in 28 minutes')

        submit_button = self.driver.find_element_by_id('submit_button')
        submit_button.click()

        # Verify that the progress is logged successfully
        progress_summary = self.driver.find_element_by_id('progress_summary')
        self.assertEqual(progress_summary.text, 'Ran 3 miles in 28 minutes')

    def tearDown(self):
        # Quit the driver and close the app
        self.driver.quit()

if __name__ == '__main__':
    unittest.main()
```

4) Connecting with friends and sharing progress and achievements -
The following code is for Connecting with friends and sharing progress and achievements . With valid credentials , navigates to the “ Friends “ section , searches for a friend and adds them and is able to share the progress.

```
import time
```

```
from appium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

# Set desired capabilities and create driver
desired_caps = {
    "platformName": "Android",
    "deviceName": "emulator-5554",
    "appPackage": "com.fitnessapp",
    "appActivity": "com.fitnessapp.MainActivity",
    "noReset": True
}
driver = webdriver.Remote("http://localhost:4723/wd/hub", desired_caps)
wait = WebDriverWait(driver, 10)

# Log in with valid credentials
username_field = wait.until(EC.presence_of_element_located((By.ID, "username_field")))
password_field = wait.until(EC.presence_of_element_located((By.ID, "password_field")))
login_button = wait.until(EC.presence_of_element_located((By.ID, "login_button")))
username_field.send_keys("exampleuser")
password_field.send_keys("password123")
login_button.click()

# Navigate to "Friends" section
friends_button = wait.until(EC.presence_of_element_located((By.ID, "friends_button")))
friends_button.click()

# Search for a friend and add them
search_field = wait.until(EC.presence_of_element_located((By.ID, "search_field")))
search_button = wait.until(EC.presence_of_element_located((By.ID, "search_button")))
search_field.send_keys("friendusername")
search_button.click()
add_friend_button = wait.until(EC.presence_of_element_located((By.ID, "add_friend_button")))
add_friend_button.click()

# Share progress with a friend
progress_button = wait.until(EC.presence_of_element_located((By.ID, "progress_button")))
progress_button.click()
share_button = wait.until(EC.presence_of_element_located((By.ID, "share_button")))
share_button.click()
friend_list = wait.until(EC.presence_of_element_located((By.ID, "friend_list")))
friend_username = "friendusername"
```

```
friend_element = friend_list.find_element_by_xpath("//android.widget.TextView[@text='" +
friend_username + "']")
friend_element.click()
share_progress_button = wait.until(EC.presence_of_element_located((By.ID,
"share_progress_button")))
share_progress_button.click()

# Log out
logout_button = wait.until(EC.presence_of_element_located((By.ID, "logout_button")))
logout_button.click()

# Quit driver
driver.quit()
```

5) Accessing a library of exercises and workout plans -
The following code is for Accessing a library of exercises and workout plans

```
import unittest
from appium import webdriver

class FitnessAppTests(unittest.TestCase):

    def setUp(self):
        desired_caps = {
            "platformName": "Android",
            "deviceName": "emulator-5554",
            "appPackage": "com.example.fitnessapp",
            "appActivity": "MainActivity"
        }
        self.driver = webdriver.Remote("http://localhost:4723/wd/hub", desired_caps)
        self.driver.implicitly_wait(10)

    def tearDown(self):
        self.driver.quit()

    def test_access_exercises_library(self):
        # Log in to the app
        self.driver.find_element_by_id("com.example.fitnessapp:id/login_button").click()

self.driver.find_element_by_id("com.example.fitnessapp:id/username_field").send_keys("userna
me")

self.driver.find_element_by_id("com.example.fitnessapp:id/password_field").send_keys("passwo
rd")
```

```
        self.driver.find_element_by_id("com.example.fitnessapp:id/login_button").click()

        # Navigate to the exercises library
        self.driver.find_element_by_id("com.example.fitnessapp:id/menu_button").click()
        self.driver.find_element_by_id("com.example.fitnessapp:id/library_button").click()
        self.driver.find_element_by_id("com.example.fitnessapp:id/exercises_button").click()

        # Check that the exercises are displayed
        exercises_list =
self.driver.find_elements_by_id("com.example.fitnessapp:id/exercise_name")
        self.assertGreater(len(exercises_list), 0, "Exercises list is empty")

    def test_access_workout_plans_library(self):
        # Log in to the app
        self.driver.find_element_by_id("com.example.fitnessapp:id/login_button").click()

self.driver.find_element_by_id("com.example.fitnessapp:id/username_field").send_keys("userna
me")

self.driver.find_element_by_id("com.example.fitnessapp:id/password_field").send_keys("passwo
rd")
        self.driver.find_element_by_id("com.example.fitnessapp:id/login_button").click()

        # Navigate to the workout plans library
        self.driver.find_element_by_id("com.example.fitnessapp:id/menu_button").click()
        self.driver.find_element_by_id("com.example.fitnessapp:id/library_button").click()
        self.driver.find_element_by_id("com.example.fitnessapp:id/workout_plans_button").click()

        # Check that the workout plans are displayed
        workout_plans_list =
self.driver.find_elements_by_id("com.example.fitnessapp:id/workout_plan_name")
        self.assertGreater(len(workout_plans_list), 0, "Workout plans list is empty")
```

In this one I have created two test cases named test_access_exercises_library() and test_access_workout_plans_library() , which both tests log into the app , navigate to the library section.

6) Integrating with other fitness apps and devices -
The following code is for integrating with other fitness apps and devices. In this the ability of the fitness tracking app to integrate with other fitness apps and devices will be tested.

```
import unittest
from appium import webdriver
```

```
class FitnessAppIntegrationTests(unittest.TestCase):

    def setUp(self):
        desired_caps = {
            'platformName': 'Android',
            'deviceName': 'device',
            'appPackage': 'com.fitness.app',
            'appActivity': '.MainActivity'
        }
        self.driver = webdriver.Remote('http://localhost:4723/wd/hub', desired_caps)

    def test_integration_with_other_apps(self):
        # Launch the fitness app and navigate to the settings screen
        self.driver.find_element_by_id('com.fitness.app:id/btn_settings').click()

        # Click on the 'Connect to other apps' option
        self.driver.find_element_by_id('com.fitness.app:id/btn_connect_to_apps').click()

        # Assert that the app can connect to other fitness apps and devices
        # For example, if the app can integrate with a Fitbit device, we can assert that the Fitbit
device is connected
        connected_device =
self.driver.find_element_by_id('com.fitness.app:id/connected_device').text
        self.assertEqual(connected_device, 'Fitbit')

    def tearDown(self):
        self.driver.quit()

if __name__ == '__main__':
    unittest.main()
```

Usability Testing Objectives -
Evaluate the ease of the application use like key features , such as logging workouts and tracking progress , to ensure users can accomplish their goals efficiently.
Test the app’s visual design and layout to ensure it is clear , consistent and visually appealing , promoting a positive vibe for users.

1) User interface design and layout -
The following code is for user interface design and layout.

```
import unittest
from appium import webdriver
from appium.webdriver.common.touch_action import TouchAction
```

```
class FitnessAppUITest(unittest.TestCase):

    def setUp(self):
        desired_caps = {}
        desired_caps['platformName'] = 'Android'
        desired_caps['platformVersion'] = '10'
        desired_caps['deviceName'] = 'Android Emulator'
        desired_caps['appPackage'] = 'com.example.fitnessapp'
        desired_caps['appActivity'] = 'com.example.fitnessapp.MainActivity'
        self.driver = webdriver.Remote('http://localhost:4723/wd/hub', desired_caps)

    def test_ui_elements(self):
        # Verify the presence and visibility of various UI elements on the home screen
        home_screen = self.driver.find_element_by_id('com.example.fitnessapp:id/home_screen')
        self.assertTrue(home_screen.is_displayed())

        workout_button =
self.driver.find_element_by_id('com.example.fitnessapp:id/workout_button')
        self.assertTrue(workout_button.is_displayed())

        progress_button =
self.driver.find_element_by_id('com.example.fitnessapp:id/progress_button')
        self.assertTrue(progress_button.is_displayed())

        friends_button = self.driver.find_element_by_id('com.example.fitnessapp:id/friends_button')
        self.assertTrue(friends_button.is_displayed())

        library_button = self.driver.find_element_by_id('com.example.fitnessapp:id/library_button')
        self.assertTrue(library_button.is_displayed())

        # Verify the layout of the workout log screen
        workout_button.click()
        workout_log = self.driver.find_element_by_id('com.example.fitnessapp:id/workout_log')
        self.assertTrue(workout_log.is_displayed())

        exercise_list = self.driver.find_element_by_id('com.example.fitnessapp:id/exercise_list')
        self.assertTrue(exercise_list.is_displayed())

        add_exercise_button =
self.driver.find_element_by_id('com.example.fitnessapp:id/add_exercise_button')
        self.assertTrue(add_exercise_button.is_displayed())

        # Verify the layout of the progress tracking screen
        progress_button.click()
        progress_screen =
self.driver.find_element_by_id('com.example.fitnessapp:id/progress_screen')
        self.assertTrue(progress_screen.is_displayed())

        goal_list = self.driver.find_element_by_id('com.example.fitnessapp:id/goal_list')
        self.assertTrue(goal_list.is_displayed())

        add_goal_button =
self.driver.find_element_by_id('com.example.fitnessapp:id/add_goal_button')
        self.assertTrue(add_goal_button.is_displayed())

        # Verify the layout of the friends screen
        friends_button.click()
        friends_screen =
self.driver.find_element_by_id('com.example.fitnessapp:id/friends_screen')
        self.assertTrue(friends_screen.is_displayed())

        friend_list = self.driver.find_element_by_id('com.example.fitnessapp:id/friend_list')
        self.assertTrue(friend_list.is_displayed())

        add_friend_button =
self.driver.find_element_by_id('com.example.fitnessapp:id/add_friend_button')
        self.assertTrue(add_friend_button.is_displayed())

        # Verify the layout of the exercise library screen
        library_button.click()
        library_screen = self.driver.find_element_by_id('com.example.fitnessapp:id/library_screen')
        self.assertTrue(library_screen.is_displayed())

        exercise_library =
self.driver.find_element_by_id('com.example.fitnessapp:id/exercise_library')
        self.assertTrue(exercise_library.is_displayed())

        search_box = self.driver.find_element_by_id('com.example.fitnessapp:id/search
```

2) Navigation and User flow -
The following code is for navigation and flow of user

```
import unittest
from appium import webdriver

class UsabilityTest(unittest.TestCase):
```

```
def setUp(self):
    desired_caps = {}
    desired_caps['platformName'] = 'Android'
    desired_caps['platformVersion'] = '10.0'
    desired_caps['deviceName'] = 'emulator-5554'
    desired_caps['appPackage'] = 'com.example.fitnessapp'
    desired_caps['appActivity'] = '.MainActivity'
    self.driver = webdriver.Remote('http://localhost:4723/wd/hub', desired_caps)

def tearDown(self):
    self.driver.quit()

def test_navigation_and_user_flow(self):
    # Test login
    username_field = self.driver.find_element_by_id('username_field')
    password_field = self.driver.find_element_by_id('password_field')
    login_button = self.driver.find_element_by_id('login_button')
    username_field.send_keys('example_user')
    password_field.send_keys('password123')
    login_button.click()

    # Test workout logging
    workouts_button = self.driver.find_element_by_id('workouts_button')
    workouts_button.click()
    log_workout_button = self.driver.find_element_by_id('log_workout_button')
    log_workout_button.click()
    exercise_field = self.driver.find_element_by_id('exercise_field')
    sets_field = self.driver.find_element_by_id('sets_field')
    reps_field = self.driver.find_element_by_id('reps_field')
    exercise_field.send_keys('Bench press')
    sets_field.send_keys('3')
    reps_field.send_keys('10')
    save_button = self.driver.find_element_by_id('save_button')
    save_button.click()

    # Test progress tracking
    progress_button = self.driver.find_element_by_id('progress_button')
    progress_button.click()
    set_goal_button = self.driver.find_element_by_id('set_goal_button')
    set_goal_button.click()
    goal_field = self.driver.find_element_by_id('goal_field')
    goal_field.send_keys('Run 5 miles in 30 minutes')
    save_button = self.driver.find_element_by_id('save_button')
```

```
        save_button.click()

        # Test connecting with friends
        friends_button = self.driver.find_element_by_id('friends_button')
        friends_button.click()
        add_friend_button = self.driver.find_element_by_id('add_friend_button')
        add_friend_button.click()
        friend_username_field = self.driver.find_element_by_id('friend_username_field')
        friend_username_field.send_keys('friend_user')
        send_request_button = self.driver.find_element_by_id('send_request_button')
        send_request_button.click()

        # Test accessing library of exercises and workout plans
        library_button = self.driver.find_element_by_id('library_button')
        library_button.click()
        exercises_button = self.driver.find_element_by_id('exercises_button')
        exercises_button.click()
        back_button = self.driver.find_element_by_id('back_button')
        back_button.click()
        workout_plans_button = self.driver.find_element_by_id('workout_plans_button')
        workout_plans_button.click()
        back_button = self.driver.find_element_by_id('back_button')
        back_button.click()

        # Test integration with other fitness apps and devices
        integration_button = self.driver.find_element_by_id('integration_button')
        integration_button.click()
        connect_fitbit_button = self.driver.find_element_by_id('connect_fitbit_button')
        connect_fitbit_button.click()
        username_field = self.driver.find_element_by_id('fitbit_username_field')
        password_field = self.driver.find_element_by_id('fitbit_password_field')
        username_field.send_keys('fitbit_user')
        password
```

3) Accessibility -
The following code is for Accessibility

```
from appium.webdriver.common.touch_action import TouchAction
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.by import By

# set up Appium driver
```

```
desired_caps = {
  "platformName": "Android",
  "deviceName": "device",
  "appPackage": "com.example.fitnessapp",
  "appActivity": "MainActivity"
}

driver = webdriver.Remote('http://localhost:4723/wd/hub', desired_caps)

# locate the element to test accessibility
element = driver.find_element_by_id("com.example.fitnessapp:id/button_workout")

# check if element is accessible
if element.is_enabled():
    print("Element is accessible")
else:
    print("Element is not accessible")

# perform accessibility actions on element
TouchAction(driver).tap(element).perform()

# wait for new screen to load
new_screen = WebDriverWait(driver, 10).until(EC.presence_of_element_located((By.ID,
"com.example.fitnessapp:id/workout_list")))

# check if new screen is accessible
if new_screen.is_enabled():
    print("New screen is accessible")
else:
    print("New screen is not accessible")

# close the driver
driver.quit()
```

In this I first set up the Appium driver and located the element to test accessibility. Then we can check if the element is enabled and perform a tap action on it using ‘TouchAction’. After that new screen loads , then we check if it's accessible or not . If it's accessible , then we can close the driver.

Performance Testing Objectives -
To measure the response time of the fitness tracking app under various scenarios , such as logging in ,accessing the exercise library, and submitting workout logs. To identify and measure

the maximum load the fitness tracking app can handle and how it performs under high traffic conditions.

1) Load testing for concurrent users -
The following code is for load testing for concurrent users . Load testing in appium can be done using various tools such as JMeter or Gatling.

```
from locust import HttpUser, TaskSet, task, between

class UserBehavior(TaskSet):
    @task
    def log_workout(self):
        self.client.post("/log_workout", json={
            "exercise": "Bench Press",
            "sets": 3,
            "reps": 10
        })

class WebsiteUser(HttpUser):
    tasks = [UserBehavior]
    wait_time = between(5, 15)
```

Here, I defined a ‘UserBehavior’ class that includes a task to log a workout with the ‘/log_workout’ endpoint. We then define a ‘WebsiteUser’ class that uses the ‘UserBehavior’ tasks and specifies a wait time between 5 and 15 seconds. We also can run this script using the ‘locust’ command in the terminal and accessing the Locust web interface at ‘http://localhost:8089’. From there, we can specify the number of concurrent users and the hatch rate (ie., how quickly new users should be spawned) to simulate
load on the server.

2) Response time and Latency testing -
The following code is for response time and latency testing .
In this I used the ‘time’ module to start and end a timer before and after the actions to be tested, then calculate the response time by subtracting the end time from the start time . We can also wait for a set amount of time before performing the actions again to calculate the latency.

```
import time
from appium import webdriver

desired_caps = {
    'platformName': 'Android',
    'deviceName': 'device',
    'appPackage': 'com.example.myapp',
    'appActivity': '.MainActivity'
}

driver = webdriver.Remote('http://localhost:4723/wd/hub', desired_caps)

# Start the timer
start_time = time.time()

# Perform actions to be tested
# ...

# End the timer
end_time = time.time()

# Calculate the response time
response_time = end_time - start_time
print('Response time:', response_time)

# Wait for a set amount of time
time.sleep(10)

# Start the timer again
start_time = time.time()

# Perform actions to be tested
# ...

# End the timer again
end_time = time.time()

# Calculate the latency
latency = end_time - start_time - 10
print('Latency:', latency)

driver.quit()
```

3) Resource utilization and memory consumption -
The following code is for resource utilization and memory consumption.

```
import psutil
import time
from appium import webdriver

# set desired capabilities
desired_caps = {
    "platformName": "Android",
    "platformVersion": "10",
    "deviceName": "emulator-5554",
    "appPackage": "com.example.fitnessapp",
    "appActivity": ".MainActivity"
}

# start Appium server and create driver instance
driver = webdriver.Remote("http://localhost:4723/wd/hub", desired_caps)

# perform some actions and measure resource utilization and memory consumption
start_time = time.time()
cpu_percentages = []
memory_usages = []
for i in range(100):
    # do some actions, e.g. log workouts
    driver.find_element_by_id("log_workout_button").click()
    driver.find_element_by_id("exercise_input").send_keys("pushups")
    driver.find_element_by_id("sets_input").send_keys("3")
    driver.find_element_by_id("reps_input").send_keys("10")
    driver.find_element_by_id("submit_button").click()

    # measure resource utilization and memory consumption
    cpu_percentages.append(psutil.cpu_percent())
    memory_usages.append(psutil.virtual_memory().percent)
end_time = time.time()

# print results
print("Elapsed time:", end_time - start_time, "seconds")
print("Average CPU utilization:", sum(cpu_percentages) / len(cpu_percentages), "%")
print("Average memory consumption:", sum(memory_usages) / len(memory_usages), "%")

# close driver and stop Appium server
driver.quit()
```

This one performs a 100 iterations cycle of logging workouts in the fitness app and measures the CPU utilization and memory consumption using the ‘psutil’ library. Also we can adjust the number of iterations and the specific actions to perform based on your performance testing objectives.

General Outline of steps to test the functionality of the app using Appium :

1. Set up the testing environment: Install and configure the necessary software components, including the Appium server, the mobile device emulator/simulator, and the test automation framework (e.g. TestNG, JUnit).

2. Create test cases: Based on the features and requirements of the app, write test cases to verify that the app functions as expected. Your test cases should cover all the features of the app and simulate user interactions, such as logging workouts, tracking progress, and connecting with friends.

3. Record test scripts: Use Appium's recorder feature or a similar tool to record test scripts that simulate user actions. This will allow you to generate code for your test cases without having to write it from scratch.

4. Customize and optimize test scripts: Edit the generated test scripts to make them more robust and efficient. For example, you may want to add assertions to verify that certain elements appear on the screen or that data is saved correctly.

5. Execute the tests: Run the test scripts on the emulator/simulator or on a real mobile device. Monitor the test execution and look for any errors or failures.

6. Report and analyze test results: After the tests have finished running, generate a report that summarizes the test results. Analyze any failures or errors to determine their root cause and work with the development team to fix any issues that were uncovered.

7. Iterate and improve: Use the insights gained from the test results to improve the testing process and the app itself. Repeat the testing cycle until all the features of the app have been thoroughly tested and any issues have been resolved.

TEST DELIVERABLES :

Test cases and Test scripts -

Test Case : Logging a Workout

Test Steps -

- ➢ Launch the fitness tracking app.
- ➢ Navigate to the “Log Workout” Screen.
- ➢ Verify that the “Exercise” field is present.
- ➢ Select an exercise from the drop-down menu.
- ➢ Verify that the “Sets” and “Reps” fields are present.
- ➢ Enter a value for “Sets” Field.
- ➢ Enter a value for “Reps” Field.
- ➢ Tap the “Save” button.
- ➢ Verify that the workout was saved successfully by checking that the workout appears in the user’s workout history.

Expected Results :

The “Exercise” , “Sets” and “Reps” fields are present and can be selected or edited.
The “Save” button is enabled and can be tapped.
The workout is saved successfully and appears in the user’s workout history.

Code Representation :

```
import unittest
from appium import webdriver

class LogWorkoutTest(unittest.TestCase):

    def setUp(self):
        desired_caps = {
            'platformName': 'Android',
            'platformVersion': '9.0',
            'deviceName': 'emulator-5554',
            'app': 'path/to/your/app.apk',
            'automationName': 'UiAutomator2'
        }
        self.driver = webdriver.Remote('http://localhost:4723/wd/hub', desired_caps)

    def tearDown(self):
        self.driver.quit()

    def test_log_workout(self):
        # Navigate to the "Log Workout" screen
        log_workout_button = self.driver.find_element_by_id('log_workout_button')
        log_workout_button.click()

        # Verify that the "Exercise" field is present
        exercise_field = self.driver.find_element_by_id('exercise_field')
        self.assertIsNotNone(exercise_field)

        # Select an exercise from the drop-down list
        exercise_dropdown = self.driver.find_element_by_id('exercise_dropdown')
        exercise_dropdown.click()
        exercise_option =
self.driver.find_element_by_xpath('//android.widget.TextView[@text="Bench Press"]')
        exercise_option.click()

        # Verify that the "Sets" and "Reps" fields are present
        sets_field = self.driver.find_element_by_id('sets_field')
        reps_field = self.driver.find_element_by_id('reps_field')
        self.assertIsNotNone(sets_field)
        self.assertIsNotNone(reps_field)

        # Enter a value for the "Sets" field
        sets_field.send_keys('3')

        # Enter a value for the "Reps" field
        reps_field.send_keys('10')

        # Tap the "Save" button
        save_button = self.driver.find_element_by_id('save_button')
        save_button.click()

        # Verify that the workout was saved successfully by checking that the workout appears in
the user's workout history
        workout_history = self.driver.find_element_by_id('workout_history')
        workout_list = workout_history.find_elements_by_class_name('workout_item')
        self.assertNotEqual(len(workout_list), 0)

if __name__ == '__main__':
    unittest.main()
```

Test Case : Tracking progress over time

Test Steps -

- ➢ Launch the fitness tracking app.
- ➢ Navigate to the “Progress” Screen.

- ➢ Verify that the user’s progress data is displayed correctly, including weight , body fat percentage, and muscle mass.
- ➢ Enter a new weight value.
- ➢ Tap the “Save” button.
- ➢ Verify that the weight value is saved correctly and appears in the user’s progress chart.

Expected Results :
The “Progress” screen displays the user’s progress data correctly.
The user can enter a new weight value and save it successfully.
The weight value is saved correctly and appears in the user’s progress chart.

Code Representation :

```
import unittest
from appium import webdriver

class TrackProgressTest(unittest.TestCase):

    def setUp(self):
        desired_caps = {
            'platformName': 'Android',
            'platformVersion': '9.0',
            'deviceName': 'emulator-5554',
            'app': 'path/to/your/app.apk',
            'automationName': 'UiAutomator2'
        }
        self.driver = webdriver.Remote('http://localhost:4723/wd/hub', desired_caps)

    def tearDown(self):
        self.driver.quit()

    def test_track_progress(self):
        # Navigate to the "Track Progress" screen
        track_progress_button = self.driver.find_element_by_id('track_progress_button')
        track_progress_button.click()

        # Verify that the user's progress is displayed
        progress_chart = self.driver.find_element_by_id('progress_chart')
        self.assertIsNotNone(progress_chart)

        # Set a fitness goal for the user
        set_goal_button = self.driver.find_element_by_id('set_goal_button')
        set_goal_button.click()
        goal_field = self.driver.find_element_by_id('goal_field')
        goal_field.send_keys('Run a 5K in under 30 minutes')
        save_button = self.driver.find_element_by_id('save_button')
        save_button.click()
```

Test Case : Connecting with friends

Test Steps -

- ➢ Launch the fitness tracking app.
- ➢ Navigate to the “Friends” Screen.
- ➢ Verify that the user’s friend list is displayed correctly.
- ➢ Tap the “Add Friend” button.
- ➢ Enter the friend’s name or username.
- ➢ Tap the “Send Request” button.
- ➢ Verify that the friend request is sent successfully.
- ➢ Wait for the friend to accept the request.
- ➢ Verify that the friend is added to the user’s friend list.

Expected Results :
The “Friends” screen displays the user’s friend list correctly.
The user can send a friend request successfully.
The friend request is accepted successfully.
The friend is added to the user’s friend list.

Code Representation :

```
 import unittest
from appium import webdriver

class ConnectWithFriendsTest(unittest.TestCase):

    def setUp(self):
        desired_caps = {
            'platformName': 'Android',
            'platformVersion': '9.0',
            'deviceName': 'emulator-5554',
            'app': 'path/to/your/app.apk',
            'automationName': 'UiAutomator2'
        }
        self.driver = webdriver.Remote('http://localhost:4723/wd/hub', desired_caps)

    def tearDown(self):
        self.driver.quit()

    def test_connect_with_friends(self):
        # Navigate to the "Connect with Friends" screen
        connect_with_friends_button =
self.driver.find_element_by_id('connect_with_friends_button')
        connect_with_friends_button.click()

        # Verify that the user's friend list is displayed
        friend_list = self.driver.find_element_by_id('friend_list')
        self.assertIsNotNone(friend_list)

        # Select a friend to view their profile
        friend_profile_button = friend_list.find_element_by_class_name('friend_profile_button')
        friend_profile_button.click()

        # Verify that the friend's profile is displayed
        friend_profile = self.driver.find_element_by_id('friend_profile')
        self.assertIsNotNone(friend_profile)

        # Send a message to the friend
        message_button = friend_profile.find_element_by_id('message_button')
        message_button.click()
        message_field = self.driver.find_element_by_id('message_field')
        message_field.send_keys('Hey, great job on your progress!')
        send_button = self.driver.find_element_by_id('send_button')
        send_button.click()

        # Verify that the message was sent successfully
        sent_messages = self.driver.find_element_by_id('sent_messages')
        message_list = sent_messages.find_elements_by_class_name('message_item')
        self.assertNotEqual(len(message_list), 0)

        # Go back to the friend list
        back_button = self.driver.find_element_by_id('back_button')
        back_button.click()

        # Select another friend to view their profile
        friend_profile_button = friend_list.find_elements_by_class_name('friend_profile_button')[1]
        friend_profile_button.click()

        # Verify that the friend's profile is displayed
        friend_profile = self.driver.find_element_by_id('friend_profile')
        self.assertIsNotNone(friend_profile)
        # Like the friend's progress update
        like_button = friend_profile.find_element_by_id('like_button')
        like_button.click()

        # Verify that the like was recorded
        liked_updates = self.driver.find_element_by_id('liked_updates')
        update_list = liked_updates.find_elements_by_class_name('update_item')
        self.assertNotEqual(len(update_list), 0)

if __name__ == '__main__':
    unittest.main()
```

Test Case : Accessing a library of exercises and workout plans

- ➢ Launch the fitness tracking app.
- ➢ Navigate to the “Library” screen.
- ➢ Verify that the list of exercises and workout plans is displayed correctly.
- ➢ Tap on an exercise or workout plan.
- ➢ Verify that the details of the exercise or workout plan are displayed correctly.

Expected Results :
The “Library” screen displays the list of exercises and workout plans correctly.
The user can tap on an exercise or workout plan and view its details correctly.

Code Representation :

```
import time
from appium import webdriver

desired_caps = {
    'platformName': 'Android',
    'deviceName': 'Android Emulator',
    'appPackage': 'com.example.fitnessapp',
    'appActivity': 'com.example.fitnessapp.MainActivity'
}

driver = webdriver.Remote('http://localhost:4723/wd/hub', desired_caps)

# Navigate to the library of exercises and workout plans
driver.find_element_by_id('btn_library').click()

# Scroll down to view more options
```

```
driver.swipe(0, 1000, 0, 500, 400)

# Select an exercise from the library
driver.find_element_by_xpath('//android.widget.ListView/android.widget.LinearLayout[3]').click()

# Add the exercise to a workout plan
driver.find_element_by_id('btn_add_workout_plan').click()
driver.find_element_by_id('workout_plan_name').send_keys('Monday Workout Plan')
driver.find_element_by_id('btn_add').click()

# Save the workout plan and navigate back to the main screen
driver.find_element_by_id('btn_save_workout_plan').click()
driver.find_element_by_id('btn_back').click()

# Close the app
driver.quit()
```

Test Case : Integrate with other fitness apps and devices

- ➢ Launch the fitness tracking app and navigate to the settings menu.
- ➢ Click on the "Connect with Other Apps" option.
- ➢ devices that can be connected with the fitness tracking app.
- ➢ Click on the fitness app or device that you want to connect with.
- ➢ Perform a task or action on the connected app or device.
- ➢ Disconnect the connected app or device.
- ➢ Verify that the data and progress from the connected app or device are still visible in the
- ➢ fitness tracking app.
- ➢ Repeat the steps with different fitness apps and devices to verify compatibility and seamless integration.

Expected Result:
The user should be able to view the list of available fitness apps and
The fitness app or device should be connected with the fitness tracking appland the user should be able to access its data and functionalities.
The action should be reflected in the fitness tracking app and the user should be able to view the data and progress in real-time.
The connection between the fitness tracking app and the connected app or
device should be terminated.
The fitness tracking app should be compatible with multiple fitness apps
and devices and should seamlessly integrate with them to provide accurate data and

Code Representation :

```
import unittest
from appium import webdriver
from time import sleep

class TestFitnessAppIntegration(unittest.TestCase):

    def setUp(self):
        # Set up Appium desired capabilities and initialize driver
        desired_caps = {
            'platformName': 'Android',
            'platformVersion': '11',
            'deviceName': 'emulator-5554',
            'appPackage': 'com.fitnessapp',
            'appActivity': '.MainActivity'
        }
        self.driver = webdriver.Remote('http://localhost:4723/wd/hub', desired_caps)

    def tearDown(self):
        # Quit the driver
        self.driver.quit()

    def test_integration_with_google_fit(self):
        # Open the app and navigate to the settings page
        settings_btn = self.driver.find_element_by_id('com.fitnessapp:id/settings_button')
        settings_btn.click()

        # Navigate to the integrations page
        integrations_btn = self.driver.find_element_by_id('com.fitnessapp:id/integrations_button')
        integrations_btn.click()

        # Connect with Google Fit
        google_fit_btn = self.driver.find_element_by_id('com.fitnessapp:id/google_fit_button')
        google_fit_btn.click()

        # Log in to Google Fit account
        # ...

        # Grant permission to access data
        # ...

        # Verify successful integration
        success_msg = self.driver.find_element_by_id('com.fitnessapp:id/success_message')
        self.assertEqual(success_msg.text, 'Google Fit connected successfully!')
    def test_integration_with_fitness_tracker_device(self):
        # Open the app and navigate to the settings page
        settings_btn = self.driver.find_element_by_id('com.fitnessapp:id/settings_button')
        settings_btn.click()

        # Navigate to the integrations page
        integrations_btn = self.driver.find_element_by_id('com.fitnessapp:id/integrations_button')
        integrations_btn.click()

        # Connect with fitness tracker device
        device_btn = self.driver.find_element_by_id('com.fitnessapp:id/device_button')
        device_btn.click()

        # Turn on device and enable Bluetooth
        # ...

        # Verify successful integration
        success_msg = self.driver.find_element_by_id('com.fitnessapp:id/success_message')
        self.assertEqual(success_msg.text, 'Device connected successfully!')


if __name__ == '__main__':
    unittest.main()
```

Improvements & Recommendations :

Based on the testing results, here are some recommendations for improving the user experience and functionality of the fitness tracking app:

1. Improve the user interface design and layout to make it more intuitive and user-friendly. This can include simplifying the navigation, using clearer labels and icons, and improving the visual hierarchy of the app.

2. Enhance the performance of the app by optimizing its response time, reducing latency, and improving resource utilization and memory consumption.

3. Add more features to the app, such as the ability to track and log meals, monitor sleep

patterns, and receive personalized recommendations based on user data. 4. Implement better security measures to protect user data, such as encryption and two-factor authentication.

5. Make it easier for users to connect with friends and share progress by integrating the app

with popular social media platforms and fitness communities.

6. Provide more options for customization and personalization, such as allowing users to create their own workout plans and set their own fitness goals.

7. Incorporate gamification elements to make the app more engaging and motivating, such as challenges, badges, and rewards for achieving milestones.
Overall, these recommendations can help improve the user experience and functionality of the fitness tracking app, making it more effective at helping users achieve their fitness goals.