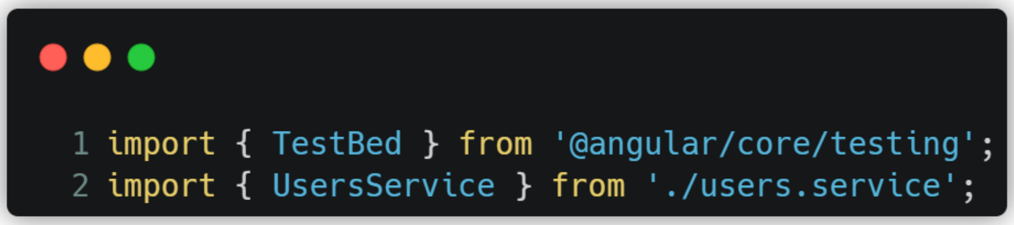
Unit Tests and Integration Tests

User Login

The following code snippets are written in JavaScript, and use the Angular framework for testing a service called `**UsersService**`

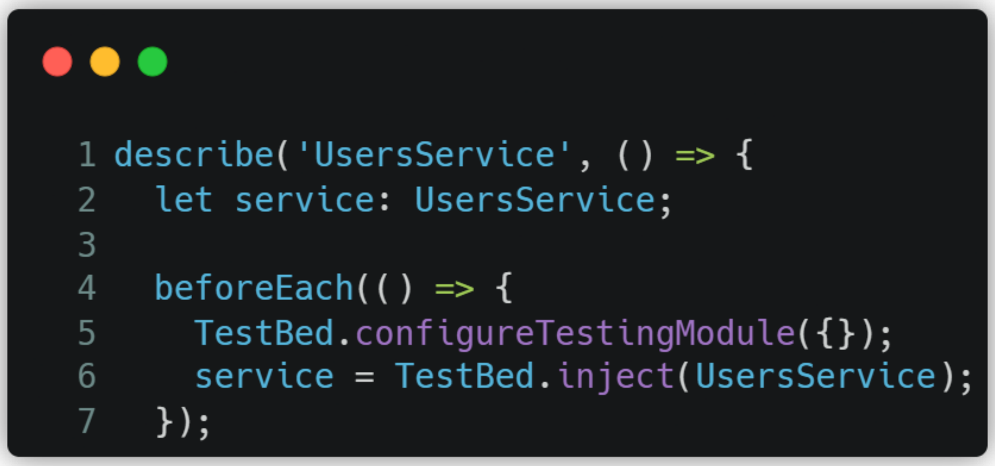
1. Importing Dependencies:



```
import { TestBed } from '@angular/core/testing';
import { UsersService } from './users.service';
```

In the first line, the `**TestBed**` module is imported from the Angular's `**core/testing**` package. This module provides utilities for configuring and creating a test module where Angular components, services, and other dependencies can be tested. The `**UsersService**` is imported from a separate file that contains the implementation of the service being tested.

2. Test Suite Initialization:



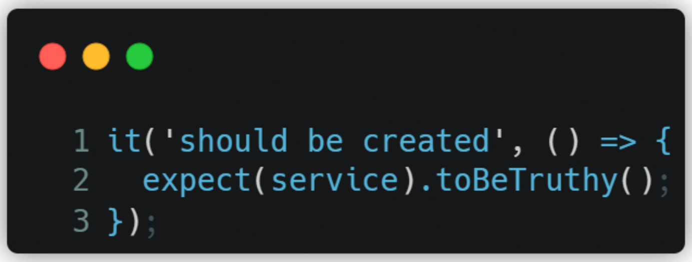
```
describe('UsersService', () => {
  let service: UsersService;

  beforeEach(() => {
    TestBed.configureTestingModule({});
    service = TestBed.inject(UsersService);
  });
```

The `**describe**` function defines a test suite with the name `**UsersService**`. Inside the test suite, a variable `**service**` of type `**UsersService**` is declared to hold an instance of the service being tested.

The `**beforeEach**` function is a setup function that is executed before each test case in the suite. In this case, it configures the TestBed module by calling `**TestBed.configureTestingModule({})**`. The empty object passed to the `**configureTestingModule**` function indicates that no additional configurations or dependencies are required for this test.

The next line `**service = TestBed.inject(UsersService);**` retrieves an instance of the `**UsersService**`from the `**TestBed**`. This allows the tests to access and manipulate the service.

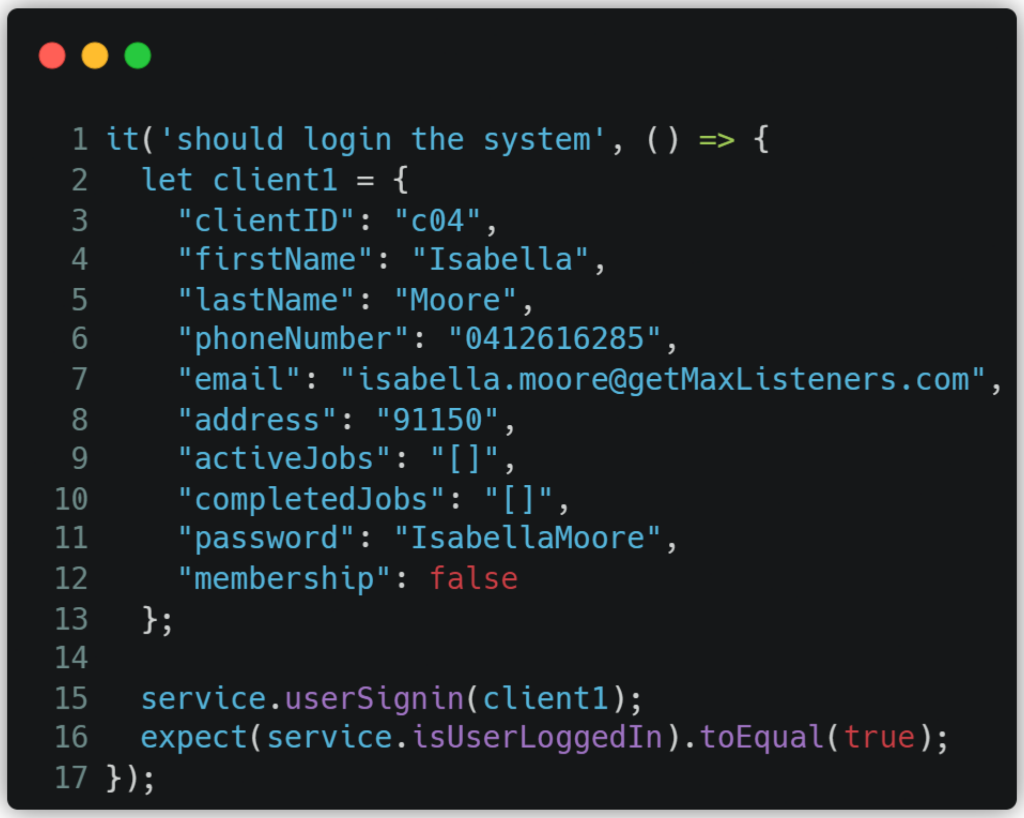
3. Unit Test: 'should be created'



```
it('should be created', () => {
  expect(service).toBeTruthy();
});
```

This is a unit test case that verifies whether the `**UsersService**` is created successfully. The `**it**` function defines an individual test case with a description. In this case, it checks if the `**service**` object is truthy (i.e., not null or undefined), indicating that the service was successfully instantiated. The `**expect**` statement with `**toBeTruthy()**` matcher verifies this condition.

4. Integration Test: 'should login the system'



```
it('should login the system', () => {
  let client1 = {
    "clientID": "c04",
    "firstName": "Isabella",
    "lastName": "Moore",
    "phoneNumber": "0412616285",
    "email": "isabella.moore@getMaxListeners.com",
    "address": "91150",
    "activeJobs": "[]",
    "completedJobs": "[]",
    "password": "IsabellaMoore",
    "membership": false
  };

  service.userSignin(client1);
  expect(service.isUserLoggedIn).toEqual(true);
});
```

This is an integration test case that checks the behavior of the `**userSignin**` method in the `**UsersService**`. It simulates a user login by creating a `**client1**` object representing user data. The `**userSignin**` method of the service is then called with `**client1**` as an argument.

The `**expect**` statement with `**toEqual(true)**` verifies that the `**isUserLoggedIn**` property of the `**UsersService**`is set to `**true**` after calling `**userSignin**`. If the login is successful, the expectation will pass.

These are the explanations of the tests conducted in the provided code. Unit tests focus on verifying the behavior of individual units (functions, methods) in isolation, while integration tests check how different components work together as a whole system.

Test Cases

Test Case ID: #TC001
Test Scenario: Client/Professional Login
Test Steps:
- The user navigates to the login page
- The user enters a registered email address as the username
- The user enters the registered password
- The user clicks 'Sign In'

Prerequisites:
- A registered account with a unique username and password.

Test Data: Legitimate username and password
Expected/Intended Results: Once the username and password are entered correctly, the user is successfully logged into their account.
Actual Results: As Expected
Test Status: Pass

Test Case ID: #TC002
Test Scenario: Client Creates Service Request
Test Steps:
- The client logs into their account
- The client navigates to the service request page
- The client fills in the necessary details for the service request
- The client submits the service request

Prerequisites:
- Client logged into their account

Test Data: Valid service request details
Expected/Intended Results: The service request is successfully created and submitted.
Actual Results: As Expected
Test Status: Pass

Test Case ID: #TC003
Test Scenario: Professional Accepts Service Request
Test Steps:
- The professional logs into their account
- The professional navigates to the service request section
- The professional views the available service requests
- The professional selects a service request to accept
- The professional clicks 'Accept'

Prerequisites:
- Professional logged into their account
- Existing service requests

Test Data: Valid service request to accept
Expected/Intended Results: The selected service request is successfully accepted by the professional.
Actual Results: As Expected
Test Status: Pass