Test Plan

Instagram Application

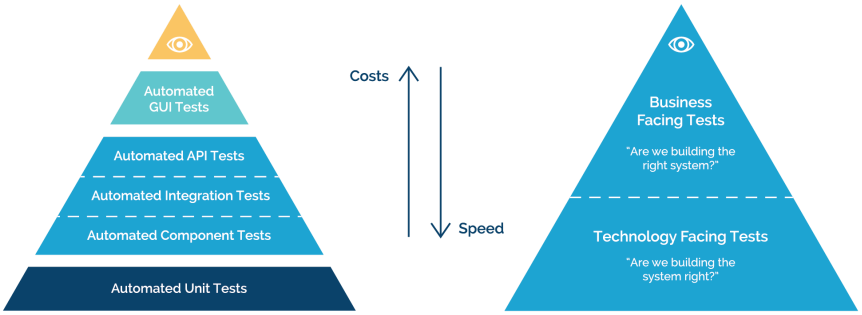
Aaqua SDET Exercise

Katerina Denisova

1. Test Methodology

This document will describe approaches and methodologies that will apply to the exploratory and system testing of the Instagram app using mobile devices.

- Login screen - here our clients simply sign in with their username/password, create a new account and login with Facebook account.
- Main app (Instagram) - on this screen we should see a user profile with many features available to use.



1.1. Test categories and 1.2. Test case creation

1. Functionality

Test the GUI of Instagram app thoroughly, testing validates the features available on the UI of an instagram app – check the functionality of the mobile app's GUI. This includes testing the menus, dropdowns, navigation buttons and gestures, landscape/portrait, languages, forms, and other features that are used by the end-user.

Test Case ID	Scenario	Test Steps	Expected Result
1	Login with valid credentials	Given I am an existing user open a login page When I login with email address and password Then I am on the home page	User is able to login successfully
2	Hide first post on homepage	Given I Am Logged In To The Home Page When I Will Hide The First Post Then The First Post Is Hidden	User will successfully hide a post
3	Like Random Post	Given I Am Logged In To The Home Page When I Will Like A Random Post Then Verify Likes On The Post	User is able to submit likes
4	Search for some varying texts	Given I Am Logged In To The Home Page When I Try To Search A Random Value Then I Will Check The Result	User is able to search anything they want and functionality is working

2.Compatibility

Test Instagram apps for cross-platform Compatibility, we have to ensure to test our application on all the possible combinations of operating systems, screen sizes and resolutions that the app users might be using. This ensures that the app will behave uniformly across all devices.

Test Case ID	Description	Test Steps	Expected Result
1	Verify the design is consistent	Given I logged in to the instagram When I explore a home page And I am using different devices Then I can see that design is consistent	Instagram app performs well on different devices, browsers, screen sizes, and OS versions, design is consistent.
2	Verify a font size	Given I logged in to the instagram When I explore a home page And I am using different devices Then I can see that font size is readable	Font size is readable, no wrapping issues
3	Verify data exchange with other applications	Given I want to make a post in Instagram When I click on create a new post Then I will see a pop up asking permission to allow using my data	I can successfully create a new post using a data from my device

3. Interruption Testing

We need to make sure that the application runs fine even when there are system generated interruptions.

Test Case ID	Description	Test Steps	Expected Result
1	Incoming phone call interruption	Given I logged in to the instagram When I explore a home page Then I have an incoming phone call	When I finish a phone call I can continue using an app without any crashes or errors
2	Low battery popup interruption	Given I logged in to the instagram When I am creating a post Then I have a popup saying that my battery is low	I expect to continue using my application and I will be able to create a new post
3	Phone shutdown	Given I logged in to the instagram When I am browsing the app Then my phone shuts down	After charging up my phone I expect to open Instagram and I will not need to login as my history will be saved and I can continue using the app

4. Back-end/database

Very important part of testing is Back-end testing and database validation, we have to make sure that our users can use all features that app has to offer.

Test Case ID	Description	Test Steps	Expected Result
1	Verify API Schema	Given I mapped an end points	I expect to see a matching

		When I perform schema validation Then the mapping formats of tables/views/columns are compatible with mapping formats of UI	mapping formats and no errors (SQL Query)
2	Verify that long running operation are running on the backend and not blocking our UI	Given I am uploading a video When it is loading Then long accrued operations are running synchronously on the background	User is able to see a loading message and the video will be uploaded successfully.
3	Verify each microservice is working as expected	Given I am checking an API responses When I use separate microservice and hit different end points Then I get expected responses	We have a good documentation to follow and a descriptive structure of microservices

5. Performance and security

Security is a very important part of testing, we have to stop data breaches and attacks from unauthorized users to sensitive information. Not less important is performance testing. We have to ensure that our users can use our application without unexpected errors or degradation of performance beyond acceptable limits.

Test Case ID	Description	Test Steps	Expected Result
1	Verify that the loading time for the app is not too long.	Given I am browsing an app When I click on back button Then I can see a page during specified duration	I expect to navigate between pages without a delay.
2	Verify that the app is Secured from an external network	Given I use an external network When I try to get an access to sensitive information Then I will be blocked	I expect that all security validations were tackled and external users cannot access anything
3	Using JMeter verify signin and signout API time	Given I have performance tests When I run signin and signout Then results are within performance limits	I expect that all token are working as expected without exceeding

1.3. Dependencies

I will need to request access to the source code. I need to know a version of the app that I will be testing, documentation for microservices and a high overview on technologies used.

1.4. Defect reporting

For defect reporting I will use JIRA, where I will include screenshots, video recordings etc. It will also include Steps how to reproduce the issue, information about the environment that was used during testing, expected and actual results together with a severity and a priority.

1.5. Tooling

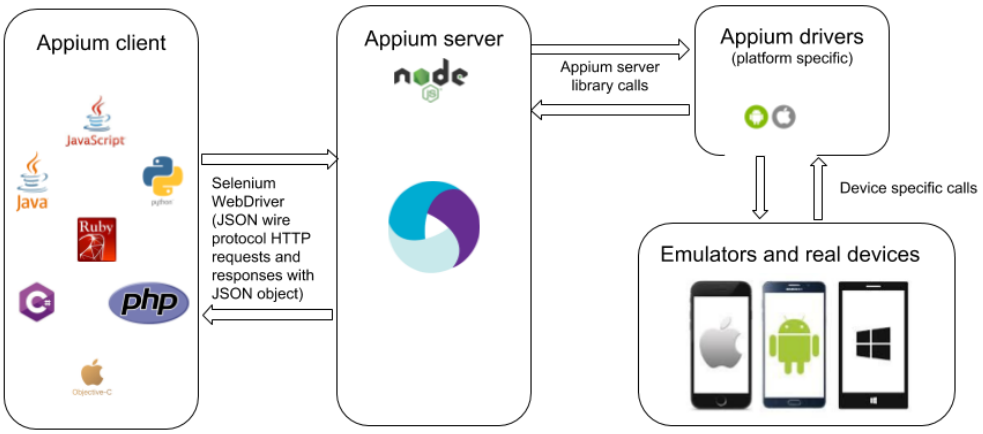
Tooling you would use for testing:

Criteria	Appium	Detox
Devices Supported	All	Do not support testing on real devices for iOS platforms

Flaky tests	Flaky	Automatically Synchronized: Stops flakiness at the core by monitoring asynchronous operations in your app.
Dev/test Environment	only black box testing available	Gray-box testing allows easy debugging and testing
Program languages supported	Java, Python, Ruby, C# etc.	JavaScript, TypeScript
Speed	Slow	Fast
Complexity	Complicated setup, API and maintenance	a little less reliable due to network and server issues
CI integration	Jenkins	Made For CI: Allows execution for E2E tests on CI platforms like Travis or TeamCity
Webdriver	Yes	No
Integration	Mocha, Robot Framework	Mocha,Espresso, Jest and Earl Grey

Recommendation: Instagram is a React Native app and based on the comparison above it is clear to see that it is best to use Detox even though it does not support testing on real devices for iOS platforms, as a work around this problem we can use Xcode to create an emulator but for this case we will need the application source code, which we do not have for Instagram. My personal laptop is Windows based, therefore I will need to install a MacOS image and create a virtual environment to be able to use Detox for automation, not an ideal solution.

For the purposes of this task, I am limited with the devices and as I have Windows based OS, to accomplish this task I chose to go with Appium and Robot Framework, targeting only Android. Even if I was able to test iOS, it is not possible to install Instagram onto the emulated device, as access to the Apple app store is not granted, and I do not have the app source code.



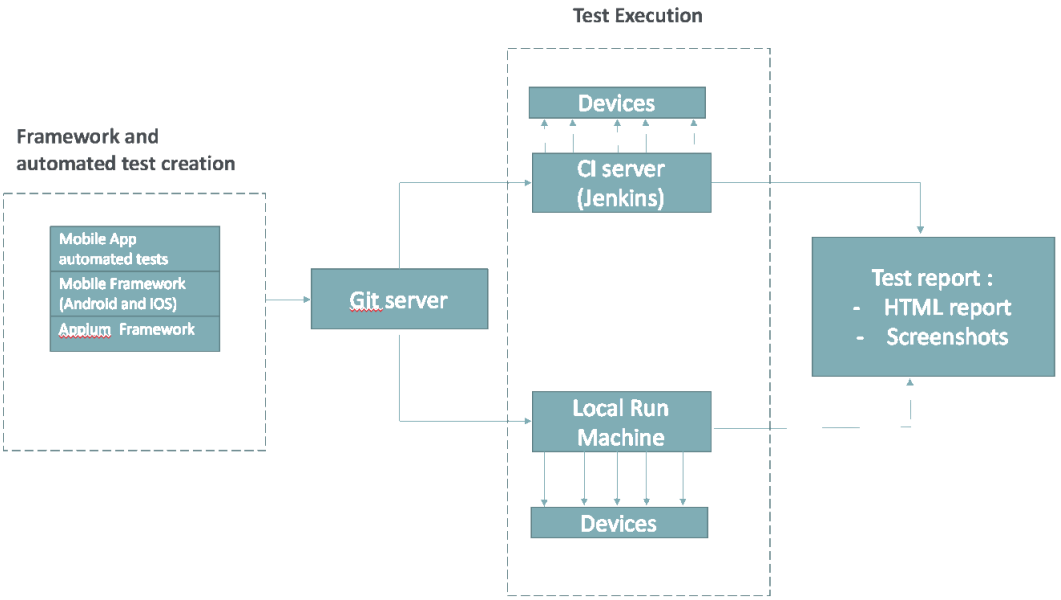
1.6. Test management

In the long term I will keep testing efforts organized through well-defined processes, also a very important part is to maintain good working relationships with all the colleagues and stakeholders.Keep test cases which need to be reused, maintained and updated, try to automate the most riskiest parts of applications and critical flows, have a good representation in test reports this will help to make informed decisions.

3. CI/CD pipeline

3.1 What does your essential CI/CD build pipeline(s) look like?

Very important part is that the entire CI feedback loop should run in less than 10 minutes, also all these processes are fully automated, with each run fully logged and visible to the entire team, it is easy to use. My favourite approach would be to use a 3rd generation CI tool such a Github Actions (or any other CI tooling), all tests we can put in a Docker container, and configurations passed in a yaml file. I will attach an example file I have built for my current company.



3.2 What causes the build pipeline(s) to start?

To trigger the build pipeline to start we will need to change a source code, manually starting the pipeline, or using a schedule configuration, the pipeline detects the change and starts an execution.

3.3 Briefly describe each step in your pipeline. Which actions are executed?

Workflow starts, we will open our CI tool and we will see the status for in-progress actions, in case of dependency or a conflict we will need to resolve it, the pipeline builds code, then our test cases will be triggered and run, and safely deploys a new version of the application. After the build has finished our reports will be downloaded to the Artifacts and we will receive an email and a Slack notification with a report.

3.4 What happens in case of success and failure?

In case of failure, we will see a failing build and we will not be able to deploy our changes to the master branch until we investigate the problem and fix the issue.The concerned team will be notified.
In case of success, we will see a green build and we will be able to push the latest changes to the master branch then deployment will take place.