Respond.io Assessment

Test Strategy Development Question

My test strategy for a highly complex feature with multiple integrations and complex data flow and intricate user interactions is explained below. This strategy is designed based on the principles of ISTQB-CTFL course:-

1. Reviewing the Documents & Understanding the Feature:

I'll start by comprehensively understanding the feature to guide the testing process. Key activities include:

a. Grasping the feature's purpose, intended solution, and objectives.
b. Reviewing PRDs, Epics, user stories, designs, and related documents for clarity.
c. Engaging in detailed discussions with stakeholders to gather requirements and resolve queries.
d. Documenting/reviewing entry and exit criteria for testing and creating a Requirements Traceability Matrix (RTM).
e. Summarizing my understanding and seeking approval from Product/Project managers before dynamic testing, including reviewing criteria and the RTM with managers/leads.

2. Test Design:

I'll structure the testing in 3 phases: Data Flows, Integrations, and User Interactions. My approach includes:

a. Creating high-level test cases covering each functional requirement.
b. Prioritizing critical paths and key functionalities for in-depth testing across these phases.
c. To cover the testing of above mentioned 3 phases, I will be applying three Functional testing techniques:- System Testing, Integration Testing & Usability Testing. I will also apply three Non Functional testing techniques to assess the quality of the feature's non functional aspects:- Performance testing, Compatibility testing, Security testing.

Functional Testing Techniques:

1. System Testing
    a. Developing test cases to cover each functional requirement.
    b. Prioritizing critical paths and high-impact functionalities for detailed testing.

2. Integration Testing:
    a. Identifying integration points and create test scenarios to validate data flows and communication between systems.
3. Usability Testing:
    a. Designing test cases simulating real user interactions for intuitive usage.

Non Functional Testing techniques:

1. Performance Testing:
    a. Createing performance test scenarios to assess load handling, response times, and system behavior under stress.
2. Compatibility Testing:
    a. Testing across multiple browsers, mobile devices, and OS versions
3. Security Testing:
    a. Defining security test cases to validate authentication, authorization, and data protection measures.

3. Test Environment Setup:

For web:

a. Requesting the Dev Team or Project Lead to configure the Staging Environment and deploy the build.

For mobile apps:

a. Requesting Android and iOS builds pointing to staging environments to replicate real-world conditions.

4. Test Data Management:

a. Preparing diverse datasets for various scenarios and edge cases.
b. Collaborating with devs and leads to ensure comprehensive data coverage for thorough feature testing.

5. Risk Analysis and Mitigation:

a. Identifying high-risk areas via risk analysis and dedicating extra testing efforts to critical integrations and complex data flows.
b. Developing contingency plans and strategies to address identified risks proactively.

6. Execution & Reporting:

a. Executing test cases using and monitoring the results
b. Systematically logging issues, prioritizing them based on severity, and generating detailed test reports, leveraging the RTM for coverage assessment.

c. Verifying exit criteria and feature quality, conducting regression testing on staging, and requesting the DevOps team to deploy the feature on the Production environment for smoke testing.
d. Communicating testing progress and outcomes to stakeholders effectively.

7. Test Closure Cycle:

a. Sharing a comprehensive final testing summary report with stakeholders, including testing details, coverage, bugs, and residual risks.
b. Archiving staging-related data and resources used for feature testing.
c. Conducting demonstrations and training sessions for the support team, creating user manuals for sales and support purposes.
d. Have a retrospective with all people involved in testing to discuss the highs and lows of the testing

NOTE: The testing of such complex features may be spread over several weeks or months. So Aside from the daily stand ups, I will schedule regular meetings with relevant stakeholders and managers to share the test progress and clear any confusions and get answers for any queries to ensure everything is aligned perfectly.

Test Case Design Question

NOTE: There is most likely a mistake in the flow diagram. we usually test the maximum character limit allowed and maximum file size allowed and NOT the minimum character limit and minimum file size allowed. I think the 'Yes' & 'No' decisions in the given flowchart are inverted. But for the assessment I will go with what we have.

Approach For writing Test Cases

The essence of my approach lies in the *'Decision Table Testing'* technique mentioned in ISTQB-CTFL. The idea behind choosing this technique is that this technique is usually the most suitable when it comes to testing flow diagrams. A Slight deviation from the technique here is that I have omitted writing conditions one by one for a single test case. This is done to avoid complications and keep things simple and effective as we have to keep in mind the time constraints as well while writing the test cases.

Considerations For test cases

In a real scenario(aside from assessment) Some of the consideration I have to do while writing test cases are

1. The target audience of feature
2. The devices usually used by my target audience - Android/IOS and their versions
3. The browsers usually used by my target audience - Chrome/Firefox/Edge etc

TC#	Section / Area	Platform	Test Case	Status (pass/ fail/ Blocked/ etc	Bug
Positive Cases For Web & Mobile (Happy flows)					
1	Valid Upload	Web	Verify the upload button on the page is available with label/copy *'upload file'*		
2	Valid Upload	Web	Verifying clicking on upload button open the the system directory/gallery		
3	Valid Upload	Web	Verify Uploading a file with name exceeding 50 characters, size exceeding 20 MB & file format is 'CSV' results in uploading to storage		
4	Valid Upload	Web	Verify Uploading a file with name exceeding 50 characters, size exceeding 20 MB & file format is 'JSON' results in file uploading to storage		
5	Valid Upload	Web	Verify I get the success message when file is successfully uploaded		
6	Valid Upload	Mobile	Verify the upload button on the page is available with label/copy *'upload file'*		
7	Valid Upload	Mobile	Verifying clicking on upload button opens the the mobile directory/gallery		
8	Valid Upload	Mobile	Verify Uploading a file with name exceeding 50 characters, size exceeding 20 MB & file format is 'CSV' results in file uploading to storage		

9	Valid Upload	Mobile	Verify Uploading a file with name exceeding 50 characters, size exceeding 20 MB & file format is 'JSON' results in file uploading to storage		
10	Valid Upload	Mobile	Verify I get the success message when file is successfully uploaded		
Negative Cases for Web & Mobile					
11	InValid Upload	Web	Verify uploading a file with name less than 50 characters & size exceeding 20 MB & format CSV/JSON throws error message		
12	InValid Upload	Web	Verify uploading a file with name greater than 50 characters but size less than 20 MB & format CSV/JSON throws error message		
13	InValid Upload	Web	Verify uploading a file with name exceeding 50 characters & size exceeding 20 MB but format other than CSV/JSON throws error message		
14	InValid Upload	Mobile	Verify uploading a file with name less than 50 characters & size exceeding 20 MB & format CSV/JSON throws error message		
15	InValid Upload	Mobile	Verify uploading a file with name greater than 50 characters but size less than 20 MB & format CSV/JSON throws error message		
16	InValid Upload	Mobile	Verify uploading a file with name exceeding 50 characters & size exceeding 20 MB but format other than CSV/JSON throws error message		
Executing the above cases will result in 100% patch coverage. Following are some *additional cases* which can be executed(if time, budget etc permits) to test the quality of features in more detail.					
17	Others	Web	Verify uploading an empty file fails file upload & throws an error		
18	Others	Web	Verify uploading multiple file(allowed/not allowed based on the expected behavior)		
19	Others	Web	Verify Canceling the upload process during file upload does NOT upload file		
20	Others	Web	Verify refreshing the page during file upload does NOT upload file		

21	Others	Web	Verify the copy of error message is correct & informative		
22	Others	Web	Verify the copy of success message is correct & informative		
23	Others	Web	Verify Canceled/interruption during upload throws an informative error		
24	Others	Mobile	Verify uploading an empty file fails file upload & throws an error		
25	Others	Mobile	Verify uploading multiple file(allowed/not allowed based on the expected behavior)		
26	Others	Mobile	Verify Canceling the upload process during file upload does NOT upload file		
27	Others	Mobile	Verify refreshing the page during file upload does NOT upload file		
28	Others	Mobile	Verify the copy of error message is correct & informative		
29	Others	Mobile	Verify the copy of success message is correct & informative		
30	Others	Mobile	Verify file upload is unsuccessful if mobile internet is disconnected during upload		
31	Others	Mobile	Verify Canceled/interruption during upload throws an informative error		

Cross-Browser Testing Question

Cross-browser testing is crucial to ensure that a web application functions seamlessly across different browsers and versions. Here's a comprehensive strategy for ensuring compatibility with the latest versions of popular browsers (Chrome, Firefox, Safari, and Edge):

1. Define Browser and Version Support:

a. I'll identify the browser versions prevalent among our customers and in our target market. Analyzing usage statistics or conducting surveys will help align our support with the browsers commonly used by our audience.

b. I'll prioritize the latest stable versions of Chrome, Firefox, Safari, and Edge based on market trends and customer usage patterns.

2. Establish Test Environments:

a. I'll utilize cloud-based services like BrowserStack or Sauce Labs to access multiple browser versions and platforms, enabling comprehensive testing across various configurations without the need for maintaining physical hardware.
b. Additionally, I'll maintain an in-house testing lab equipped with physical devices for more precise testing, ensuring detailed examination and debugging in a controlled environment.

3. Automation Tools and Frameworks:

a. Using browser automation tools such as Selenium or Playwright, I'll automate critical test scenarios across different browsers.
b. Regularly updating these scripts will accommodate changes in browser behavior or application updates.

4. Manual Testing:

a. I'll conduct manual testing to evaluate user interfaces, navigation, and responsiveness on each browser. This human-centric approach helps identify issues that might not be captured by automated tests.

5. Test Coverage:

a. Critical Features: I'll Prioritize core functionalities that must work seamlessly across all browsers.
b. Visual Consistency: I'll Ensure consistent appearance and layout in different browsers.
c. Responsiveness: I'll Test on various screen sizes and devices for optimal display and behavior.
d. Forms and User Interactions: I'll Verify correct functioning of forms, buttons, and other interactive elements.

6. Regularly Update Browser Versions:

a. I'll keep testing environments updated with the latest stable versions of browsers. Monitoring release schedules and retiring support for older versions based on usage statistics and industry trends will be part of my routine.

7. Reporting and Documentation:

a. I'll generate detailed test reports for each browser, highlighting any issues or deviations from expected behavior. Additionally, maintaining comprehensive

documentation on known browser-specific issues, workarounds, and resolutions will be a priority.

This refined cross-browser testing strategy integrates both cloud-based services and in-house testing facilities to ensure comprehensive coverage, precise testing, and alignment with customer usage patterns, ultimately ensuring a consistent and robust user experience across different browsers and versions.

Test Automation Framework Question

1. **Tool Selection:**

Playwright: I'd recommend Playwright for its modern capabilities and suitability for the application's characteristics:

a. Cross-browser support (Chrome, Firefox, Safari, Edge)
b. Reliable element interactions and waiting strategies
c. Fast execution speed
d. Efficient handling of complex web features (iframes, shadow DOM)
e. Built-in support for visual regression testing

2. **Framework Architecture:**

a. Language Selection:
I'd opt for a programming language that suits the team's expertise and application requirements. JavaScript (with Playwright) would be a strong choice

b. Page Object Model (POM):
I'd adopt a modular approach using the Page Object Model (POM) design pattern. This involves creating separate classes for each web page or component to encapsulate their functionalities. It helps in reusability, maintainability, and readability of test code

c. Data-Driven Testing:
I'd separate test data from test logic for flexibility and ease of use.

d. Test Case Types:

i. **Smoke Tests:** Essential workflows that verify critical functionalities across user roles, ensuring basic operability of the application.
ii. **Regression Tests:** Comprehensive tests covering key user workflows for each role, ensuring that new changes don't break existing functionalities

e. **Execution:**
   Run tests locally and on a Continuous Integration (CI) server like Jenkins for regular automated test runs.

f. **Reporting and Logging:**
   Implementation of detailed logging and reporting mechanisms using tools like Allure or ExtentReports. This allows for comprehensive test result analysis, aiding in debugging and identifying issues quickly.