**Budgeting App Test Plan**

**Overview**

This document discusses high-level strategies for testing UI for the specified feature.

**Goals**

User can do the following:

* Add inflow/outflow items
* View categorized amounts on charts

**Non-Goals**

Backend testing is not discussed here.

**Test Strategies and Test Cases**

**Check-in Tests:** Create tests to check the basic functionality when new code is checked-in by developers, to ensure that the new code does not break existing functionality. If these tests fail, the developer won’t be allowed to check-in the new code.

**Build Verification Tests:** Create BVTs to test the basic core functionalities on every new build and ensure that the page is stable for testing.

**Comprehensive Test:** These tests are P2.

**Test Cases:** Test cases are located in BudgetingAppTestCases.xlsx. Here’s the link: https://github.com/sunkarameera/BudgetTestApp/blob/master/e2e/BudgetingAppTestCases.xlsx

**Accessibility**

Verify that the user is able to navigate around the Budgeting App using only their keyboard.

**Browser, OS & Device Matrix**

Ensure all the listed scenarios work correctly in all the company-supported browser, OS and device matrix. Below is an example of company-supported matrix.

|  |  |
| --- | --- |
| Browsers | Chrome 54, 55, 56  Firefox 50, 51, 52  IE 9, 10, 11  Safari 8, 9, 10 |
| OS | Windows 7, 8, 10  Mac OS X |
| Devices | Samsung Galaxy 5, 6, 7  Apple iPhone 5, 6, 7  Google Nexus 5X, 6P |

**Fault Tolerance**

Put a fall back mechanism in place when the budgeting data cannot be retrieved from backend.

**Error Page**

Navigate the user to appropriate error pages for different failures. Set up correct error pages.

**Automation**

Create automation for UX. Write UX browser automation tests using WebdriverIO and mobile automation using **Appium.**

Write automation in the team-supported frameworks using JavaScript or Ruby.

**Performance**

Create performance web-tests for the Budgeting feature with JMeter.

**Stress test:** Perform a stress test to analyze the system/page’s behavior on breakdown. Especially, check effects on functionality and rule out susceptibility to potential security threats under breakdown.

**Load test:** Perform a load test to understand the system’s behavior during normal and anticipated peak loads, to find the system’s breaking-point.

**Security**

Run penetration tests to verify the system and find all the security loop holes. Also, run tests to ensure there are no SQL attacks or denial of service attacks when user tries to save a malicious script. Use a tool like XSSer.

**Upgrade**

The Budgeting feature should work seamlessly on different browser updates. On every browser update, verify that all the test automation suites are run.

**Glossary**

If any, we will add here.