Mobile Automation Framework 2
1. Technology Stack 2
2. Automation Requirements 2
3. Implementation Steps 2
4. Sample Test Script 3
5. Best Practices 4
6. Version Control and Reporting 4

Mobile Automation Framework

1. Technology Stack

Scripting Language:

JavaScript and TypeScript:

Utilize TypeScript for its static typing features, which enhance code quality and maintainability.

Automation Tool:

Playwright:

A powerful tool for automating web and mobile applications, supporting TypeScript and allowing for mobile device emulation.
Alternatively, Appium can be used for native app testing.

2. Automation Requirements

Test Scenarios to Cover:

Login with Registered User: Automate the login process to verify user credentials.
Adding a Recipient: Script the steps to add a recipient in the app.
Logout: Automate the logout process to ensure users can exit securely.

Logging and Reporting:

Capture screenshots and logs at each critical step to facilitate debugging and verification.
Store test results and reports in a GitHub repository for version control and collaboration.

3. Implementation Steps

Project Initialization:

```bash
```

```
npm init -y
npm install playwright typescript ts-node @types/node
```

Create `tsconfig.json`:

```json
{
  "compilerOptions": {
    "target": "ES6",
    "module": "commonjs",
    "outDir": "./dist",
    "rootDir": "./src",
    "strict": true,
    "esModuleInterop": true
  },
  "include": ["src//*"]
}
```

4. Sample Test Script

Create a file named `mobileTest.ts` in the `src` directory with the following content:

```typescript
import { chromium, devices } from 'playwright';

const iPhone11 = devices['iPhone 11'];

(async () => {
  const browser = await chromium.launch({ headless: false });
  const context = await browser.newContext({
    ...iPhone11,
    locale: 'en-US',
    geolocation: { latitude: 37.7749, longitude: -122.4194 },
    permissions: ['geolocation'],
  });

  const page = await context.newPage();

  // Login Test
  await page.goto('https://scopex.in/login');
  await page.fill('#username', 'registeredUser');
  await page.fill('#password', 'userPassword');
  await page.click('#submit');

  // Verify Login Success
  const successMessage = await page.textContent('.success-message');
  if (successMessage?.includes('Welcome')) {
    console.log('Login successful');
  } else {
    console.error('Login failed');
```

```
    await page.screenshot({ path: 'login-failed.png' });
  }

  // Adding a Recipient
  await page.goto('https://scopex.in/add-recipient');
  await page.fill('#recipientName', 'John Doe');
  await page.fill('#recipientAccount', '1234567890');
  await page.click('#addRecipient');

  // Capture Screenshot after adding recipient
  await page.screenshot({ path: 'recipient-added.png' });

  // Logout Test
  await page.click('#logout');

  // Close the browser
  await browser.close();
})();
```

5. Best Practices

Ensure structured and reusable test scripts by organizing code into functions or classes.
Use descriptive naming conventions for test cases and variables.
Implement error handling to capture failures gracefully.

6. Version Control and Reporting

Use GitHub Actions to automate the process of running tests and storing results in the repository.
Generate reports using tools like Allure or Mocha for better visibility of test outcomes.