# Code Review for Cypress Project

## Found Errors

1) **Allure reports are pushed to the main branch instead of gh-pages.**

The Allure reports are being pushed to the master branch, which is not the best practice. This can clutter the main branch and mix the source code with generated files.

2) **baseUrl is hardcoded in the config.**

The baseUrl inside the config file looks like this:

```
baseUrl: 'http://automationpractice.com/index.php'
```

This approach reduces flexibility and makes it harder to reuse the test suite on multiple environments (e.g staging, dev, production).

3) **Tight coupling between pages.**

Using loginPage.signinLink in the validateSuccessfulLogout method of the MyAccountPage class breaks the encapsulation principle and leads to tight coupling between pages. This can cause issues with scalability and maintainability, as changes on one page may affect another.

4) **Login and logout combined in one test.**

In the test file login.test.ts, login and logout are combined into a single test case. This makes it difficult to identify which part failed in case of an error, and goes against the principle of testing one functionality per test case.

5) **Duplicate tests with different data sources.**

In the test file login.test.ts, two test cases perform the same validation logic, but with data coming from different sources. This results in duplicate test logic. Maintaining both test cases increases codebase size without adding value.

6) **Hardcoded data is used in tests instead of dynamic test data generation tools.**

In “Login with valid credentials” test, static credentials are hardcoded directly in the test logic, such as:

```
loginPage.login("testautomation@cypresstest.com",
"Test@1234")
```

This makes the test suite harder to maintain. It also reduces randomness and limits the ability to detect issues that might arise with different input values.

7) **Unnecessary comments in the commands.js file.**

The default comment block generated by Cypress provides explanations about how to create custom commands, but it adds clutter in the project once the file is customized.

8) **Test name does not reflect the functionality.**

The test is named 'Sample Test', but it does not indicate what functionality is being tested. The test name should clearly describe the functionality being tested (e.g., "Verify that login works with valid credentials").

9) **Test does not perform any verification.**

The test does not contain any assertions or checks. A test should include verification to confirm that the functionality works as expected.

10) **Inconsistent test design.**

In some tests, cy.visit() is used directly in the test cases, while in others, a custom method launchApplication() in the LoginPage class is used. This inconsistency reduces reusability. Directly calling cy.visit() in individual tests goes against the principles of the Page Object Model and reduces maintainability.

**11) Incorrect error message in test assertion.**

The test fails because the expected error message `'Invalid email addresssss.'` does not match the actual error message returned by the application. The mismatch between the expected and actual message causes the test to fail.

**12) Non-standard configuration file naming.**

The configuration file is named `config.config.ts`, which is non-standard. Typically, configuration files are named `cypress.config.ts` for clarity and consistency. The redundant `.config` in the filename may cause confusion and does not follow common naming rules for configuration files.

**13) Unnecessary use of wildcards in .gitignore**

In the .gitignore file, some folder entries include wildcards (*) that are unnecessary. node_modules, videos, reports, screenshots, and cypress/downloads do not require wildcards, as the folders themselves will be ignored without needing to specify /*.

**14) The lib option in the TypeScript configuration includes unnecessary libraries.**

Cypress typically only needs es6 as the target library for its execution environment. Including "dom" and "es7" may lead to issues with TypeScript, as it could introduce extra types that are irrelevant for Cypress, or cause type conflicts when not properly managed. Simplifying this option will ensure that the configuration is cleaner and more efficient.

**15) Unnecessary usage of "include" option in typescript configuration.**

By manually specifying paths to node_modules, you're introducing potential redundancy in the configuration. TypeScript already knows to look inside node_modules for type definitions and other resources. This extra configuration can make the setup more complicated and harder to maintain.

## Suggestions for improvements

1) It is recommended to use the gh-pages branch for deploying static content such as test reports. Create a new branch named gh-pages in the repository. In the GitHub Actions workflow file replace: `BRANCH: master` with `BRANCH: gh-pages`. All reports will be pushed to a separate branch, leaving the master branch clean and focused on the source code.
2) It would be better to externalize the baseUrl into an environmental variable via .env file or CI environmental variables. This allows running the same tests on different environments without modifying the source code. To do this, you need to install the dotenv plugin using `npm install dotenv -D`. Then create a .env file: BASE_URL="http://automationpractice.com/index.php".
   Import dotenv into config file: `import dotenv from 'dotenv';`
   Update config file: `baseUrl: process.env.BASE_URL`.
3) To ensure that the Page Object Model principles are followed correctly, the MyAccountPage should not depend on elements or methods from the LoginPage. Instead, each page should be self-contained and abstract interactions with other pages internally.

4) Separate login and logout into two independent test cases. By doing this, each test validates only one behaviour and failures can be traced more clearly.
5) Keep only the test that uses data from fixtures file, as it provides better flexibility and reusability. Hardcoded test data should be avoided.
6) Use test data generation plugins such as faker.js to dynamically create test data. For using faker.js, install the plugin with `npm install @faker-js/faker --save-dev` and import `import { faker } from '@faker-js/faker'` in tests.
7) It's better to remove the default comments to avoid unnecessary clutter.
8) Rename the test using descriptive title such as "Verify that the user is able to visit the homepage successfully".
9) Add assertions to the test. The test should verify that the expected outcome occurs. This can be checking if elements are present on the page, if the title is correct, or if an action has been completed. Logging to the console, like using console.log, is not generally recommended in tests. It doesn't validate whether the test passes or fails. It's typically used for debugging purposes, but should not be part of final test code. Tests should focus on assertions that check the expected outcomes, rather than just logging information.
10) Create a BasePage class with a general open() method. This method can be used across all pages to navigate to different parts of the site. Instead of writing cy.visit() directly in each test, you can call open() from the page object.
11) Check the actual error message displayed by the application when an invalid email format is entered. It is likely that the message does not contain the extra "s" in "address". Then modify the assertion to

```
loginPage.validateLoginError('Invalid email
address.')
```

12) Rename the configuration file to a more standard and clear name, such as `config.ts` or `cypress.config.ts`. This will improve readability and maintain consistency with standard practices.
13) Remove unnecessary wildcards. node_modules, videos, reports, screenshots, and cypress/downloads should be listed without the /*.
14) Modify the lib setting to only include es6 instead of dom, es6, and es7. This will optimize the configuration and ensure compatibility with Cypress.
15) Remove the unnecessary inclusion of @shelex/cypress-allure-plugin/reporter and cypress from the "include" section. Only include the relevant TypeScript files, such as your source code.