Problems Identified

No Explicit Waits

- The test calls `findElement()` without verifying that the element is actually present in the DOM.
- This causes flakiness and makes the test prone to failures due to race conditions.

Lack of Page Object Model (POM)

- UI element locators are hardcoded within the test logic.
- If the UI changes, it requires modifying the test logic directly, violating the Single Responsibility Principle.

No Logging for Debugging Steps

- The test does not log any specific steps or actions being performed (opening URL, clicking login).
- This makes it hard to trace failures or analyze behavior in reports.

Unclear Logging and Error Messages

- The test throws a generic error like "Login failed" with no context.
- This makes it harder to debug, especially in CI/CD environments.

Tight Coupling of Concerns

- Driver setup, navigation, interaction, and validation are all handled inside one function.
- This approach is not scalable or maintainable.

No Reusability or Extensibility

- The current structure does not allow easy reuse of components like login steps or element actions.

No Global Timeout Configuration

- There's no centralized setup for default timeouts, so every wait must be defined manually (if used at all).

Missing Headless Browser Setup

- The original test runs the browser in full GUI mode.
- This is not suitable for CI/CD or headless server environments.
- Adding `--headless` enables background execution without a display.
- Additionally, `--disable-notifications` prevents browser popups from interfering with test execution.

Improvements:

- Chrome notifications were disabled (`--disable-notifications`) to ensure tests are not interrupted by browser-level popups.

Driver Factory

- Sets up the WebDriver instance with default timeouts.
- Allows central management of browser configuration.

Base Page Class

- Implements reusable utility methods like `click()`, `type()`, `waitForElement()`, and `getText()`.
- Encourages DRY (Don't Repeat Yourself) principles.

Page Object Model

- Separates page-specific logic and selectors from the test logic.
- Makes the test more maintainable and scalable.

Clean and Informative Error Handling

- Provides meaningful error messages including actual vs. expected results.
- Helps speed up debugging during failures.

Added Debugging Logs

- Added logging for each test step (e.g., navigating to login page, submitting credentials).
- Enables clear visibility into what the test is doing at every stage.

Test Logic Separation

- The test only calls high-level methods like `loginPage.login()` and `loginPage.getWelcomeMessage()`.
- Ensures maximum readability and ease of modification.

Reusable Across CI/CD Pipelines

- The new structure can be easily integrated with Jest, Mocha, GitLab CI, GitHub Actions.
- Headless mode (`--headless`) and notification suppression (`--disable-notifications`) were added to ensure compatibility and stability in headless CI environments.

Resulting File Structure

```
project/
│
├── pages/
│   ├── basePage.js
│   ├── browserFactory.js
│   └── loginPage.js
```

```
│
└── test/
    └── testLogin.js
```

Code Implementation

pages/browserFactory.js

```
const { Builder } = require('selenium-webdriver');
const chrome = require('selenium-webdriver/chrome');

async function createDriver(browser = 'chrome', headless = true) {
 let options;

 if (browser === 'chrome') {
  options = new chrome.Options();

  options.addArguments('--disable-notifications');

  if (headless) {
   options.addArguments('--headless');
  }
 }

 const driver = await new Builder()
  .forBrowser(browser)
  .setChromeOptions(options)
  .build();

 await driver.manage().setTimeouts({ implicit: 0, pageLoad: 10000 });

 return driver;
}

module.exports = { createDriver };
```

---

pages/basePage.js

```
const { until } = require('selenium-webdriver');

class BasePage {
  constructor(driver) {
    this.driver = driver;
  }
```

```
  async waitForElement(locator, timeout = 10000) {
    return await this.driver.wait(until.elementLocated(locator), timeout);
  }

  async click(locator) {
    const element = await this.waitForElement(locator);
    await element.click();
  }

  async type(locator, text) {
    const element = await this.waitForElement(locator);
    await element.sendKeys(text);
  }

  async getText(locator) {
    const element = await this.waitForElement(locator);
    return await element.getText();
  }

  async getTitle() {
    return await this.driver.getTitle();
  }
}

module.exports = BasePage;
```

---

pages/loginPage.js

```
const { By } = require('selenium-webdriver');
const BasePage = require('./basePage');

class LoginPage extends BasePage {
  constructor(driver) {
    super(driver);
    this.url = 'https://example.com/login';
    this.selectors = {
      username: By.id('username'),
      password: By.id('password'),
      loginButton: By.id('login-button'),
      welcomeMessage: By.id('welcome'),
    };
  }

  async open() {
    console.log('Navigating to login page...');
    await this.driver.get(this.url);
  }
```

```
  async login(username, password) {
    console.log('Filling in login form...');
    await this.type(this.selectors.username, username);
    await this.type(this.selectors.password, password);
    await this.click(this.selectors.loginButton);
  }

  async getWelcomeMessage() {
    console.log('Retrieving welcome message...');
    return await this.getText(this.selectors.welcomeMessage);
  }
}

module.exports = LoginPage;
```

---

test/testLogin.js

```
const { createDriver } = require('../pages/browserFactory');
const LoginPage = require('../pages/loginPage');

async function testLogin() {
  const driver = await createDriver();
  const loginPage = new LoginPage(driver);

  try {
    console.log('Starting login test...');
    await loginPage.open();
    await loginPage.login('user', 'pass');
    const message = await loginPage.getWelcomeMessage();

    if (message !== 'Welcome User') {
      throw new Error(`Unexpected welcome message: "${message}"`);
    }

    console.log('Login successful!');
  } catch (error) {
    console.error('Test failed:', error.message);
  } finally {
    await driver.quit();
  }
}

testLogin();
```