Full Solution with Explanations

1. Three Main Issues in the Original Code:

1. No explicit waits: The code uses `findElement` without waiting explicitly for elements to be available and loaded. This can cause flaky tests when the site or elements load slowly.
2. Hardcoded values: URL, username, password, and expected text are all hardcoded directly in the test, making maintenance, reuse, and running tests across different environments harder.
3.Unclear error messages: The test throws a generic error on failure without detailed assertion messages, making debugging difficult.
4. Bonus (Playwright): Use of Storage State: To avoid logging in before every test run, Playwright allows saving the authenticated session using storageState. This improves performance and stability by reusing the login state across tests.
Ideal for setups requiring authentication without repeating login flows.

2. Recommended Improvements:

1.Add explicit waits using `until.elementLocated` before interacting with elements.
2.Move all hardcoded values to a separate config file (`config.js`) for easier maintenance and flexibility.
3.Use assertions with clear, descriptive error messages to simplify debugging and test failure analysis

.3. Complete Clean and Maintainable Selenium Code:

config.js

```
module.exports = {
  BASE_URL: 'https://example.com/login',
  USERNAME: 'user',
  PASSWORD: 'pass',
  EXPECTED_WELCOME_TEXT: 'Welcome User',
  TIMEOUT: 10000,
};
```

testLogin.js

```
const { Builder, By, until } = require('selenium-webdriver');
const assert = require('assert');
const config = require('./config');

async function testLogin() {
  const driver = await new Builder().forBrowser('chrome').build();

  try {
    await driver.get(config.BASE_URL);

    // Explicit wait and fill username
    const usernameField = await driver.wait(until.elementLocated(By.id('username')), config.TIMEOUT);
    await usernameField.sendKeys(config.USERNAME);

    // Explicit wait and fill password
    const passwordField = await driver.wait(until.elementLocated(By.id('password')), config.TIMEOUT);
    await passwordField.sendKeys(config.PASSWORD);

    // Explicit wait and click login button
    const loginButton = await driver.wait(until.elementLocated(By.id('login-button')), config.TIMEOUT);
    await loginButton.click();

    // Explicit wait for welcome text element and get text
    const welcomeElement = await driver.wait(until.elementLocated(By.id('welcome')), config.TIMEOUT);
    const actualWelcomeText = await welcomeElement.getText();

    // Assert with detailed error message
    assert.strictEqual(
      actualWelcomeText,
      config.EXPECTED_WELCOME_TEXT,
      `❌ Welcome message mismatch. Expected: "${config.EXPECTED_WELCOME_TEXT}", but got: "${actualWelcomeText}"`
    );

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

testLogin();
```

4. (Bonus) Playwright globalSetup for saving storageState to avoid repeated logins

```
import { chromium } from '@playwright/test';
import config from './config';

async function globalSetup() {
  const browser = await chromium.launch({ headless: true });
  const context = await browser.newContext();
  const page = await context.newPage();

  await page.goto(config.BASE_URL);
  await page.fill('#username', config.USERNAME);
  await page.fill('#password', config.PASSWORD);
  await page.click('#login-button');

  await page.waitForSelector('#welcome', { timeout: config.TIMEOUT });

  await context.storageState({ path: 'storageState.json' });
  console.log('✅ storageState saved successfully');

  await browser.close();
}

export default globalSetup;
```