Laguna.rs Automation Testing Project Review

Overall Assessment

Great job, Ivana! For a beginner project, this is **really solid work**. You've implemented:

- ✅ Page Object Model pattern correctly
- ✅ Proper separation of concerns
- ✅ Reusable BasePage with helper methods
- ✅ Configuration management
- ✅ Multiple test scenarios covering key flows

Your code is clean, readable, and shows good understanding of Selenium fundamentals.

Strengths

1. Excellent BasePage Implementation

Your `BasePage` is comprehensive with methods for:

- Waits (explicit, visibility, clickability)
- Scrolling
- Alert handling
- GDPR popup handling
- JavaScript execution

This is advanced for a beginner!

2. Clean Page Object Model

Each page class is focused and has clear responsibilities. Good naming conventions throughout.

3. Configuration Management

Using a properties file for test data is a best practice. Nice work!

4. Test Coverage

You cover the main user flows: login (valid/invalid), search, cart operations, and full purchase flow.

Suggestions for Improvement

🔴 CRITICAL: Security Issue ✅ RESOLVED

Problem: Your `config.properties` file contained real credentials that were visible in your GitHub repository!

Status: ✅ You've already fixed this by:

- Adding `config.properties` to `.gitignore`
- Creating `config.properties.example` with dummy values
- The real credentials were never actually pushed to GitHub (the folder was already ignored)

Great job resolving this! 🔒

🟡 Code Improvements

1. Remove Duplicate Methods in BasePage

You have several duplicate or very similar methods:

Duplicates to remove:

- `waitForPageToLoad()` and `waitForPageLoadComplete()` - keep one
- Multiple alert handling methods - consolidate
- Two versions of `resolveHtmlAlertIfPresent()` - keep one

Example consolidation:

```
java
```

```
// Keep this one
public void handleAlert(boolean accept) {
    try {
        Alert alert = wait.until(ExpectedConditions.alertIsPresent());
        if (accept) {
            alert.accept();
        } else {
            alert.dismiss();
        }
    } catch (TimeoutException e) {
        // No alert present - that's okay
    }
}

// Remove: alertAccept(), acceptAlert(), getAlertTextAndAccept(), dismissAlertIfPresent()
```

2. Instantiate Page Objects in @Before Method ✅ DONE IN CartTest

In your tests, you should create page objects for every test using `@Before`:

Better approach (like you now have in CartTest):

```java
public class LoginTest extends BaseTest {
    private LoginPage loginPage;

    @Before
    public void setUp() throws Exception {
        super.setUp();
        loginPage = new LoginPage(driver, BaseTest.DEFAULT_TIMEOUT);
    }

    @Test
    public void validLoginTest() {
        loginPage.login(/* ... */);
        // ...
    }
}
```

Apply this pattern to all your test classes:

- ✅ CartTest (already done!)

- ⚠️ LoginTest (needs update)
- ⚠️ SearchTest (needs update)
- ⚠️ BuyFlowTest (needs update)

3. Standardize Timeout Durations ✅ PARTIALLY DONE

You've created `DEFAULT_TIMEOUT` in BaseTest, now use it everywhere:

```java
// In BaseTest - ✅ Already done
protected static final Duration DEFAULT_TIMEOUT = Duration.ofSeconds(20);

@BeforeClass
public static void beforeClass() {
    driver = new ChromeDriver();
    driver.manage().window().maximize();
    driver.manage().timeouts().pageLoadTimeout(DEFAULT_TIMEOUT);
    driver.manage().timeouts().scriptTimeout(DEFAULT_TIMEOUT);
    basePage = new BasePage(driver, DEFAULT_TIMEOUT);
}

// In tests - Use BaseTest.DEFAULT_TIMEOUT everywhere
loginPage = new LoginPage(driver, BaseTest.DEFAULT_TIMEOUT);
```

4. Improve Dynamic Locator Methods

Your dynamic locators are good, but can be improved:

Current:

```java
private By productTitle(String product) {
    return By.xpath("//*[@class=\"naslov\" and text()='" + product + "']");
}
```

Better (safer with quotes in product names):

```java
```

```
private By productTitle(String product) {
    return By.xpath("//*[@class='naslov' and text()=\"" + product + "\"]");
}
```

Or even better, use contains for partial matching:

```java
private By productTitle(String product) {
    return By.xpath("//*[@class='naslov' and contains(text(),\"" + product + "\")]");
}
```

5. Add Logging

Add simple logging to understand test flow:

```java
public void login(String email, String password) {
    System.out.println("Attempting login with email: " + email);
    click(loginButton);
    type(emailInputField, email);
    type(passwordInputField, password);
    click(submitInputField);
    System.out.println("Login form submitted");
}
```

Or use a proper logging framework like Log4j or SLF4J for more advanced projects.

6. Improve Test Assertions

Some of your tests call methods but don't assert the result:

Current:

```java
cartPage.isProductInCart(ConfigReader.get("product.title")); // Called but not used
Assert.assertTrue("Message", cartPage.isProductInCart(ConfigReader.get("product.title")));
```

Better:

```java
```

```
Assert.assertTrue("The selected product should be in cart",
            cartPage.isProductInCart(ConfigReader.get("product.title")));
```

Remove the duplicate call.

7. Consider Test Data in Test Methods

For better readability, you could define test data as constants:

```java
public class LoginTest extends BaseTest {
    private LoginPage loginPage;

    private static final String VALID_EMAIL = ConfigReader.get("valid.email");
    private static final String VALID_PASSWORD = ConfigReader.get("valid.password");

    @Test
    public void validLoginTest() {
        loginPage.login(VALID_EMAIL, VALID_PASSWORD);
        Assert.assertTrue("Valid login proof should contain text 'Odjava'",
                    loginPage.validLoginProofCheck());
    }
}
```

8. Add Browser Selection Support

Currently hardcoded to Chrome. Add flexibility:

```java
```

```
// In config.properties
browser = chrome

// In BaseTest
@BeforeClass
public static void beforeClass() throws Exception {
    String browser = ConfigReader.get("browser");

    switch (browser.toLowerCase()) {
        case "firefox":
            driver = new FirefoxDriver();
            break;
        case "edge":
            driver = new EdgeDriver();
            break;
        default:
            driver = new ChromeDriver();
    }

    driver.manage().window().maximize();
    driver.manage().timeouts().pageLoadTimeout(DEFAULT_TIMEOUT);
    driver.manage().timeouts().scriptTimeout(DEFAULT_TIMEOUT);
    basePage = new BasePage(driver, DEFAULT_TIMEOUT);
}
```

9. Add Screenshot on Failure ✅ DONE

You've already implemented this! Great work. Just make sure:

- ✅ `screenshots/` folder is in `.gitignore`
- ✅ Apache Commons IO dependency is in `pom.xml`

10. Add Product Add-to-Cart Validation ✅ DONE IN CartTest

You've already added cart count validation in `ProductPage` and updated `CartTest`. Excellent improvement!

🟢 Optional Enhancements (For Future Learning)

1. Consider TestNG Instead of JUnit

TestNG offers more features for test management (groups, dependencies, parallel execution, better reporting).

2. Add Test Reports

Use ExtentReports or Allure for beautiful HTML test reports.

3. Parameterized Tests

Instead of multiple similar tests, use parameterized testing:

```java
@RunWith(Parameterized.class)
public class LoginTest extends BaseTest {

    @Parameters
    public static Collection<Object[]> data() {
        return Arrays.asList(new Object[][] {
            { "valid@email.com", "validpass", true },
            { "invalid@email.com", "wrongpass", false },
            { "", "", false }
        });
    }
}
```

4. Page Factory Pattern

Instead of `By` locators, you can use `@FindBy` annotations (though your current approach is perfectly fine):

```java
@FindBy(id = "pretraga_rec")
private WebElement searchInputField;
```

Specific File Feedback

BasePage.java

- ⭐ **Excellent** - Very comprehensive
- ⚠️ Remove duplicate methods (alert handlers, page load waits)
- Consider making `sleep()` method public or removing it (waiting should use explicit waits)

LoginPage.java

- ✅ Clean and focused
- Good use of normalize-space() in XPath

- ⚠️ Update to use `@Before` for page object instantiation

SearchPage.java

- ✅ Simple and effective
- Dynamic locator is good
- ⚠️ Update to use `@Before` for page object instantiation

ProductPage.java ✅ UPDATED

- ✅ Good structure
- ✅ Added cart count validation - excellent!
- ✅ `wasItemAddedToCart()` method is a great addition

CartPage.java

- ✅ Good handling of GDPR popup
- `isProductRemoved()` method is well-designed with multiple checks

BuyerPage.java

- ✅ Complex form handling done well
- Good use of dynamic locators for dropdowns
- ⚠️ Update to use `@Before` for page object instantiation

PaymentPage.java & ConfirmationPage.java

- ✅ Clean implementations
- Consistent use of GDPR handling

CheckoutPage.java

- ✅ Simple URL verification - appropriate for the use case

BaseTest.java ✅ UPDATED

- ✅ Good setup/teardown structure
- ✅ `DEFAULT_TIMEOUT` constant added
- ✅ Screenshot on failure implemented
- ⚠️ Consider adding browser selection option

Test Classes

- ✅ CartTest - Updated with `@Before` and cart validation
- ⚠️ LoginTest - Move page object instantiation to `@Before` method
- ⚠️ SearchTest - Move page object instantiation to `@Before` method
- ⚠️ BuyFlowTest - Move page object instantiation to `@Before` method
- Remove duplicate method calls before assertions

ConfigReader.java

- ✅ Simple and effective
- Consider adding error handling for missing keys

Priority Action Items

✅ Completed:

1. 🔴 Security: Config file protection with `.gitignore`
2. 🟡 Screenshot on test failure functionality
3. 🟡 Cart count validation in ProductPage
4. 🟡 `@Before` method in CartTest
5. 🟡 `DEFAULT_TIMEOUT` constant

🔄 Remaining:

1. 🟡 Remove duplicate methods from BasePage
2. 🟡 Move page object instantiation to `@Before` in LoginTest, SearchTest, BuyFlowTest
3. 🟡 Update all tests to use `BaseTest.DEFAULT_TIMEOUT`
4. 🟢 Add basic logging (optional)
5. 🟢 Add browser selection support (optional)

Updated Files Summary

Files You've Already Updated:

- ✅ `.gitignore` (security)

- ✅ `config.properties.example` (template)
- ✅ `BaseTest.java` (screenshot, DEFAULT_TIMEOUT)
- ✅ `ProductPage.java` (cart validation)
- ✅ `CartTest.java` (@Before, validation)

Files That Still Need Updates:

- ⚠️ `BasePage.java` (remove duplicates)
- ⚠️ `LoginTest.java` (add @Before)
- ⚠️ `SearchTest.java` (add @Before)
- ⚠️ `BuyFlowTest.java` (add @Before)

Example: LoginTest with @Before

Here's how to update `LoginTest.java`:

```
java
```

```java
package appTests;

import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;
import pages.LoginPage;
import util.ConfigReader;

public class LoginTest extends BaseTest {

//   Objects:
    private LoginPage loginPage;

//   Setup:
    @Before
    public void setUp() throws Exception {
        super.setUp();
        loginPage = new LoginPage(driver, BaseTest.DEFAULT_TIMEOUT);
    }

//   Tests:
    @Test
    public void validLoginTest() {
        loginPage.login(ConfigReader.get("valid.email"), ConfigReader.get("valid.password"));
        Assert.assertTrue("Valid login proof should contain text 'Odjava'",
                loginPage.validLoginProofCheck());
    }

    @Test
    public void invalidLoginTest() {
        loginPage.login(ConfigReader.get("invalid.email"), ConfigReader.get("invalid.password"));
        Assert.assertTrue("Invalid login proof should contain text 'Prijava'",
                loginPage.invalidLoginProofCheck());
    }

    @Test
    public void emptyFieldsLoginTest() {
        loginPage.login("", "");
        Assert.assertTrue("Failed login proof should contain text 'Prijava' when fields are empty",
                loginPage.invalidLoginProofCheck());
    }
}
```

Apply the same pattern to `SearchTest` and `BuyFlowTest`.

Final Thoughts

This is **excellent work for a beginner**! Your project demonstrates:

- Good understanding of automation principles
- Clean code organization
- Practical test scenarios
- Best practices like POM and configuration management
- Willingness to learn and improve

The suggestions above will make your project even better, and you've already implemented several of them!

Current Grade: 9/10 (up from 8.5/10 after your improvements!)

The remaining items are mostly polish and consistency improvements. What you have now would definitely impress in a beginner-level exam or interview.

Keep up the great work! 🚀

Resources for Further Learning

- Selenium official docs: https://www.selenium.dev/documentation/
- Page Object Model: https://www.selenium.dev/documentation/test_practices/encouraged/page_object_models/
- TestNG framework: https://testng.org/
- Extent Reports: https://www.extentreports.com/
- Log4j logging: https://logging.apache.org/log4j/2.x/