Explain your framework architecture:

- I developed a Selenium automation framework using Java, TestNG, Cucumber BDD, and Maven.

- The framework follows Page Object Model design pattern to maintain clean separation between test logic and UI elements.

- The framework supports Chrome and Microsoft Edge execution in headless mode, suitable for Jenkins CI pipelines.

- Test execution is controlled using TestNG XML and reports are generated using Allure plugins.

Explain folder structure

1. base package

[Contains driver initialization logic]

- DriverFactory is responsible for creating WebDriver instances.
- Load URL
- I used **ThreadLocal** to maintain separate driver instances for parallel execution.

2. pages package (Page Object Model)

- Each page class contains locators and actions related only to that specific page.

- This improves maintainability because UI changes require updates only in page classes.

3. stepDefinitions package

- Step definitions map Gherkin steps from feature files to Java methods.

- They call page class methods and perform validations using TestNG assertions.

4. Hooks class

Hooks class manages test lifecycle.

@Before → initializes driver
@After → quits browser

This ensures driver creation and cleanup is centralized.

5. runners (TestRunner)

TestRunner integrates Cucumber with TestNG.

It defines:

- feature file path

- step definition glue code

- reporting plugins

6. testng.xml / parallel.xml

TestNG XML controls execution and parallel execution strategy.

Parallel Execution Explanation

- I implemented parallel execution using **ThreadLocal WebDriver**.

- Browser type is passed as parameter from TestNG XML &

- DriverFactory initializes respective browser.

Headless Execution Explanation

Headless mode improves execution speed and is suitable for Jerkins CI.

why POM?

- Page Object Model improves code reuse, readability, and reduces maintenance effort.

- And it separate the test logic from UI pages.

Short Flow Architecture Summary

→ Feature File
→ Step Definition
→ Page Class
→ DriverFactory creates driver
→ TestNG handles execution
→ Cucumber generates reports

- First, feature files define test scenarios using Gherkin syntax.

- Step definitions map steps to automation code.

- Page classes contain locators and reusable actions.

- DriverFactory handles browser initialization using ThreadLocal for parallel execution.

- TestNG manages execution and parallel threads.

- Reports are generated using Allure plugins.

Workflow:

Test Runner (Cucumber + TestNG)

- Test Runner is the entry point of execution.
- It integrates Cucumber with TestNG and controls how tests run.

Responsibilities:

- Execute feature files

- Control test flow

- Manage parallel execution

- Generate reports

Feature Files (Gherkin Scenarios)

- Feature files contain test scenarios written in simple English using Gherkin language like Given, When, Then.

purpose : Business readable test cases.

Step Definitions

- Step definitions connect feature file steps to actual automation code.
- Each step written in Gherkin is mapped to a Java method.

purpose : Bridge between test scenarios and Selenium actions.

BaseClass (Driver Setup)

- BaseClass manages WebDriver initialization and browser setup
- Load URL

Page Objects (POM Layer):

- Each page has its own class representing UI elements and actions.
- This follows Page Object Model design pattern which separates UI locators from test logic.