📘 Cucumber BDD Automation Guide

Selenium + Java + Maven + Eclipse

Beginner to Job-Ready | Real Project Based

📑 Table of Contents

1️⃣ Why Cucumber BDD? (Recruiter Perspective)

Cucumber enables **Behavior Driven Development (BDD)** where:

- Test cases are written in **plain English**
- Non-technical stakeholders can understand tests
- Automation code follows **business behavior**

✅ Recruiters value Cucumber because it shows:

- Strong **automation framework design**
- Collaboration mindset
- Industry-aligned testing approach

2️⃣ Tools & Technologies Used

Tool	Purpose
Java	Programming language
Selenium WebDriver	Browser automation
Cucumber	BDD framework
Maven	Dependency management
JUnit	Test execution
Eclipse IDE	Development environment

ChromeDriver	Browser driver
Page Object Model	Framework design pattern

3️⃣ What You Will Build

A **real-time automation framework** that:

- Opens Chrome browser
- Navigates to a form page
- Validates title & URL
- Fills user details
- Handles alert popups
- Generates **HTML & JSON reports**

👉 This mimics **real company test frameworks**

4️⃣ Maven Project Creation (Step-by-Step)

Create Maven Project

```
File → New → Other → Maven → Maven Project
→ Next → Next
→ Catalog: Internal
→ Group ID: ProjectName
→ Artifact ID: ProjectName
→ Finish
```

⏳ Wait for dependency download
When prompted → **Type Y and press Enter**

Cleanup Default Structure

Delete these auto-created packages:

```
src/main/java
src/test/java
```

(We will recreate clean structure manually)

5️⃣ Required Dependencies (pom.xml)

These dependencies support:

- Cucumber execution
- Selenium browser automation
- Reporting & assertions

Core Dependencies Used

- cucumber-core
- cucumber-java
- cucumber-junit
- selenium-java
- junit
- gherkin
- hamcrest-core
- cucumber-html
- cucumber-reporting

- cobertura
- com.sun tools

📌 **Recruiter Note:**
Using Maven dependencies shows **professional automation setup**

6️⃣ Recommended Folder Structure

```
ProjectName
│
├── Features
│   └── Form.feature
│
├── src/test/java
│   ├── pageObjects
│   │   └── FormPage.java
│   │
│   ├── stepDefinitions
│   │   └── Steps.java
│   │
│   ├── testRunner
│   │   └── testRun.java
│   │
│   └── Utilities
│
├── Drivers
│   └── chromedriver.exe
│
├── target
│   └── htmlreport.html
│
└── pom.xml
```

7️⃣ Adding Browser Drivers

Add required drivers inside **Drivers** folder.

🔗 Driver Repository Used:

```
https://github.com/RouthKiranBabu/Mini_Major--Projects./tree/main/Projects/Year_Equals_2026/01_Jan/02_%5BCucumber%2C%20Junit%2C%20%26%20Selenium%5D%20-%20Feature%20Requirements%20to%20Executable%20Tests/project_id/Drivers
```

This ensures **cross-system execution readiness**

8️⃣ Writing Your First Feature File

📄 **Features/Form.feature**

```
Feature: Form

Scenario: Single User Fills the Form
  Given Maximized Chrome Browser
  And URL is "https://v1.training-support.net/selenium/simple-form"
  Then Check Title is "Simple Form"
  And Check URL is
"https://v1.training-support.net/selenium/simple-form"
  And Enter first name as "Routh"
  And Enter last name as "Kiran Babu"
  And Enter email as "routhfamily123@gmail.com"
  And Enter contact number as "1234322343"
  And Enter message as "Good Form!"
  And Click button named "cleaR"
  Then Check alert text as "Thank You for reaching out to us, Routh
Kiran Babu"
  And Click button named OK
  And Wait "3" seconds then close browser
```

📌 **Why recruiters like this:**
Readable + Business-focused + Parameterized steps

9️⃣ Page Object Model (FormPage.java)

✔ Separates **UI logic** from test logic
✔ Improves reusability and maintenance

Key responsibilities:

- Locate elements
- Perform actions
- Validate UI behavior
- Handle alerts

📌 You implemented:

- `@FindBy`
- `PageFactory`
- Alert handling
- Button decision logic
- Implicit waits

This is **industry-grade POM usage**

🔟 Step Definitions (Steps.java)

Steps act as the **bridge** between:

```
Gherkin steps → Java methods → Selenium actions
```

Key concepts used:

- `@Given`, `@Then`

- Parameterized steps
- WebDriver lifecycle
- Thread waits
- Conditional logic

📌 Real-world logic:

- Handling clear vs submit
- Skipping alert when reset is clicked

1️⃣1️⃣ Test Runner (testRun.java)

```
@RunWith(Cucumber.class)
@CucumberOptions(
    features = ".//Features/Form.feature",
    glue = "stepDefinitions",
    dryRun = false,
    monochrome = true,
    plugin = {
      "pretty",
      "html:target/htmlreport.html",
      "json:target/cucumber.json"
    }
)
```

✔ Executes all scenarios
✔ Generates reports
✔ Connects feature + steps

1️⃣2️⃣ Executing the Tests

Method 1: Run Feature File

```
Right Click → Form.feature → Run As → Cucumber Feature
```

Method 2: Run via JUnit

```
Right Click → testRun.java → Run As → JUnit Test
```

1️⃣3️⃣ Dry Run (Best Practice)

```
dryRun = true
```

✔ Validates step mappings
✔ No browser execution
✔ Fast debugging

Then switch back to:

```
dryRun = false
```

1️⃣4️⃣ Reports & Result Analysis

📄 Location:

```
target/htmlreport.html
```

✔ Open with System Editor
✔ Visual execution summary
✔ Recruiter-friendly evidence

1️⃣5️⃣ Common Errors & Fixes

Issue	Fix

@RunWith error	Hover → Import Cucumber
Steps not found	Run dryRun = true
Driver error	Check driver path
Alert exception	Validate clear vs submit

---

16 Why Recruiters Like This Project

✔ BDD framework
✔ Page Object Model
✔ Maven + JUnit
✔ Reporting
✔ Real UI validation
✔ Clean structure
✔ Readable tests

This is **placement-ready automation**

---

17 Key Skills Gained

- Cucumber BDD
- Selenium automation
- Framework design
- Test execution
- Debugging
- Reporting
- Industry standards