

Cucumber - reports

Here's a beginner-friendly guide on **Cucumber Reports** from scratch with code explanation. This will cover:

✓ What is Cucumber?

Cucumber is a testing tool that supports **Behavior-Driven Development (BDD)**. It allows writing test cases in **Gherkin** syntax (plain English) and automating them using **Java + Selenium + JUnit/TestNG**.

✓ What is a Cucumber Report?

Cucumber Reports are **test execution reports** that display the status (passed/failed/skipped) of each scenario and step in a structured format. Types of reports:

- **Pretty/Default Report** (Console)
- **HTML Report**
- **JSON Report**
- **JUnit Report**
- **Advanced Reports** (e.g., Extent Report, Cucumber Report Plugin)

✓ Basic Cucumber Project Setup (Maven + Java)

1. Folder Structure

```
bash

src
├── test
│   ├── java
│   │   ├── stepDefinitions
│   │   │   └── LoginSteps.java
│   └── resources
│       ├── features
│       │   └── Login.feature
```

✓ 2. Maven pom.xml Dependencies

```
xml

<dependencies>
  <!-- Cucumber Dependencies -->
  <dependency>
    <groupId>io.cucumber</groupId>
    <artifactId>cucumber-java</artifactId>
    <version>7.14.0</version>
```

```

</dependency>
<dependency>
  <groupId>io.cucumber</groupId>
  <artifactId>cucumber-junit</artifactId>
  <version>7.14.0</version>
  <scope>test</scope>
</dependency>

<!-- JUnit -->
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.13.2</version>
  <scope>test</scope>
</dependency>
</dependencies>

```

✓ 3. Feature File: Login.feature

gherkin

Feature: Login Functionality

Scenario: Successful login with valid credentials

Given User is on Login page

When User enters valid username and password

Then User should be navigated to the Dashboard

✓ 4. Step Definition: LoginSteps.java

java

```

package stepDefinitions;

import io.cucumber.java.en.*;

public class LoginSteps {

    @Given("User is on Login page")
    public void user_on_login_page() {
        System.out.println("User navigates to Login page");
    }

    @When("User enters valid username and password")
    public void enter_valid_credentials() {
        System.out.println("User enters username and password");
    }

    @Then("User should be navigated to the Dashboard")
    public void user_should_see_dashboard() {
        System.out.println("User is navigated to dashboard");
    }
}

```

✓ 5. Test Runner: TestRunner.java

```
java

import org.junit.runner.RunWith;
import io.cucumber.junit.Cucumber;
import io.cucumber.junit.CucumberOptions;

@RunWith(Cucumber.class)
@CucumberOptions(
    features = "src/test/resources/features",
    glue = "stepDefinitions",
    plugin = {
        "pretty", // Console output
        "html:target/cucumber-reports/cucumber.html", // HTML Report
        "json:target/cucumber-reports/cucumber.json", // JSON Report
        "junit:target/cucumber-reports/cucumber.xml" // JUnit Report
    },
    monochrome = true
)
public class TestRunner {}
```

✓ 6. Running the Test

- Right-click on `TestRunner.java` > Run as JUnit Test.

- After execution, check the folder:

`target/cucumber-reports/`

You'll find:

- `cucumber.html` → HTML report
- `cucumber.json` → JSON report
- `cucumber.xml` → JUnit report

✓ 7. Cucumber HTML Report Preview

Open `target/cucumber-reports/cucumber.html` in a browser. You'll see:

- Feature name
- Scenario(s)
- Step-by-step result (✓ Passed or ✗ Failed)
- Execution Time

✓ 8. Optional: Pretty HTML Reports (Using Cucumber Reporting Plugin)

Add Plugin to `pom.xml`:

```
xml

<plugin>
  <groupId>net.masterthought</groupId>
  <artifactId>maven-cucumber-reporting</artifactId>
```

```
<version>5.7.0</version>
</plugin>
```

Generate Report via Maven Command:

```
sh

mvn verify
```

This will generate a pretty report at:

target/cucumber-html-reports/overview-features.html

✓ Summary

| Report Type | How to Enable | Output File |
|-------------|--|-------------------------------------|
| Pretty | "pretty" | Console output |
| HTML | "html:target/cucumber-reports/cucumber.html" | HTML report |
| JSON | "json:target/cucumber-reports/cucumber.json" | JSON file (used for custom reports) |
| JUnit | "junit:target/cucumber-reports/cucumber.xml" | XML format (CI/CD integration) |

✓ Next Step Suggestions

Would you like:

- ✓ To integrate Extent Reports?
- ✓ More real-world scenario examples?
- ✓ Cucumber + Selenium mini-project?

Let me know and I'll guide you step-by-step.

Great — let's now learn how to **read and analyze Cucumber reports** effectively. This is an essential skill for identifying failures, understanding test behavior, and debugging issues in your automation framework.

✓ Types of Reports and How to Read Them

1. Console Output (pretty)

- Shown in your IDE or terminal when you run the test.
- Shows:
 - Step-by-step execution of each scenario
 - Pass/fail indicators (✓ / ✗)
 - Exception or stack trace (if any step fails)

Sample:

```
pgsql

Feature: Login Functionality

  Scenario: Successful login with valid credentials
    Given User is on Login page
    When User enters valid username and password
    Then User should be navigated to the Dashboard
  ✓ PASSED
```

- ✓ Use this when debugging quickly during development.

2. HTML Report (cucumber.html)

- Located at: `target/cucumber-reports/cucumber.html`
- Open with any browser

Key Sections:

- **Features** and **Scenarios** listed with status (✓ / ✗)
- **Step-by-step breakdown** of each scenario
- **Execution Time**
- **Error Details:** If any step fails, it will show:
 - Exception Message
 - Stack Trace

Example:

```
pgsql

Feature: Login Functionality
  Scenario: Successful login with valid credentials
```

- ✓ Given **User is on Login** page
- ✓ When **User** enters **valid** username **and** password
- ✗ Then **User** should be navigated **to** the Dashboard

java.lang.AssertionError: Expected dashboard but **found login** page

- ✓ Useful for sharing reports with non-technical stakeholders.

3. JSON Report (cucumber.json)

- Used by other tools (like Extent Reports, Allure, or Jenkins plugins) to generate advanced reports.
- Not human-friendly to read directly.
- Use when integrating with CI/CD.

Example snippet:

```
json
{
  "keyword": "Scenario",
  "name": "Successful login with valid credentials",
  "steps": [
    {
      "keyword": "Given ",
      "name": "User is on Login page",
      "result": { "status": "passed" }
    },
    ...
  ]
}
```

4. JUnit Report (cucumber.xml)

- Useful for CI tools like Jenkins, Bamboo.
- Can be parsed by test trend plugins.
- Shows which scenarios passed/failed/skipped.

✓ How to Analyze Reports

Step-by-Step Checklist

1. **Open the HTML Report** (cucumber.html)
2. **Locate the Failed Scenario** (usually shown in red or ✗)
3. **Click the Scenario to Expand**
4. **Check the Step that Failed**
5. **Read the Stack Trace/Message**
 - Example: `java.lang.NullPointerException`
 - Indicates a variable or object was `null`

6. Check Execution Time

- Helps in identifying performance issues

7. Look for Patterns

- Are multiple scenarios failing at the same step?
- Often points to broken shared setup (e.g., WebDriver not initialized)

✓ Example Analysis Case

Report Snippet:

```
pgsql
```

```
Scenario: Login with invalid credentials
```

```
✓ Given User is on Login page
```

```
✓ When User enters invalid username and password
```

```
x Then User should see an error message
```

```
java.lang.AssertionError: Expected "Invalid credentials" but found ""
```

🔍 Analysis:

- Step failed: "Then User should see an error message"
- Assertion failed → UI did not display expected error
- Possible causes:
 - Error element not present or not visible
 - Wrong locator
 - Logic skipped due to failed backend call

✓ Action: Reproduce manually, inspect error message element, check logs.

✓ Best Practices for Reading Reports

| Tip | Description |
|----------------------------|--|
| ✓ Expand Steps | Always expand failed steps to view errors |
| 📊 Review Trends | Watch for repeated failures over time |
| 🔄 Re-run Failing Scenarios | Sometimes flaky due to environment |
| 🔧 Use Logs | Add System.out.println() or logging in step definitions |
| 📁 Archive Reports | Save reports after each run (CI/CD or manually) |

✓ Optional: Advanced Visual Reports

If you want **interactive dashboards** with charts, filters, and historical trends:

- **Extent Reports**
- **Allure Reports**
- **Cucumber Reporting Plugin (Masterthought)**

Would you like help integrating any of those?

Let me know if you want:

- A failed test case simulation to analyze?
- Cucumber + Extent report integration?
- Or how to read JSON reports in Jenkins?

I'm here to help step-by-step.

Great! Let's continue step-by-step. Since you replied "yes," I'll guide you through **Cucumber + Extent Reports integration**, show a **failed test case simulation**, and teach you how to **read it visually** using the **Extent Report dashboard**.

✓ Part 1: Integrate Extent Reports in Cucumber Project

📁 Project Structure (Maven)

Same as before, but now we'll add Extent reporting.

✓ Step 1: Add Dependencies in pom.xml

```
xml

<dependencies>
  <!-- Cucumber Core -->
  <dependency>
    <groupId>io.cucumber</groupId>
    <artifactId>cucumber-java</artifactId>
    <version>7.14.0</version>
  </dependency>
  <dependency>
    <groupId>io.cucumber</groupId>
    <artifactId>cucumber-junit</artifactId>
    <version>7.14.0</version>
    <scope>test</scope>
  </dependency>

  <!-- JUnit -->
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.13.2</version>
  </dependency>

  <!-- Extent Report -->
  <dependency>
    <groupId>tech.grasshopper</groupId>
    <artifactId>cucumber-extentsreport</artifactId>
    <version>1.7.0</version>
  </dependency>
</dependencies>
```

✓ Step 2: Update TestRunner.java

```
java

import org.junit.runner.RunWith;
import io.cucumber.junit.Cucumber;
import io.cucumber.junit.CucumberOptions;

@RunWith(Cucumber.class)
@CucumberOptions(
  features = "src/test/resources/features",
  glue = "stepDefinitions",
  plugin = {
    "pretty",
    "html:target/cucumber-reports/cucumber.html",
    "json:target/cucumber-reports/cucumber.json",
    "com.aventstack.extentreports.cucumber.adapter.ExtentCucumberAdapter:"
  },
```

```
    monochrome = true
  )
  public class TestRunner {}
```

✅ This **ExtentCucumberAdapter**: plugin generates a professional HTML report.

✅ Step 3: Add extent.properties in src/test/resources

```
properties

extent.reporter.spark.start=true
extent.reporter.spark.out=target/extent-report/index.html
extent.reporter.spark.config=src/test/resources/spark-config.xml
```

✅ (Optional) Add spark-config.xml for customizing report look

You can skip this if you're okay with the default style.

✅ Part 2: Simulate a Failed Scenario

Login.feature

```
gherkin

Feature: Login Functionality

  Scenario: Login with invalid credentials
    Given User is on Login page
    When User enters invalid username and password
    Then User should see an error message
```

LoginSteps.java

```
java

package stepDefinitions;

import io.cucumber.java.en.*;

public class LoginSteps {

    @Given("User is on Login page")
    public void user_on_login_page() {
        System.out.println("Opened login page.");
    }

    @When("User enters invalid username and password")
    public void enter_invalid_credentials() {
        System.out.println("Entered invalid credentials.");
    }

    @Then("User should see an error message")
    public void user_should_see_error_message() {
        // Simulating a failed test
        String expected = "Invalid credentials";
        String actual = ""; // Simulating no error message shown
    }
}
```

```

        if (!expected.equals(actual)) {
            throw new AssertionError("Expected '" + expected + "' but found '" + actual + "'");
        }
    }
}

```

✓ Part 3: Run the Test and View Report

▶ Run:

- Right-click `TestRunner.java` → **Run as JUnit Test**

📁 Check Report:

- Navigate to:
`target/extent-report/index.html`
- Open in your browser.

✓ Part 4: Reading the Extent Report

🔍 What You'll See:

✓ Dashboard

- Total Features, Scenarios, and Steps
- Passed / Failed counts
- Timeline and pie charts

📄 Test Details

- Click a scenario → Shows step-by-step logs
- Failed step → Highlighted in **red**
- Exception details (like: `AssertionError: Expected 'Invalid credentials' but found ''`)
- Timestamps and execution duration

📷 Screenshot Support (Optional)

You can integrate screenshots in the `AfterStep` hook.

✓ Optional: Add Screenshot on Failure

Add this to your `Hooks.java`:




```
java

@AfterStep
public void addScreenshot(Scenario scenario) {
    if (scenario.isFailed()) {
        // Take screenshot using Selenium WebDriver
        byte[] screenshot = ((TakesScreenshot) driver).getScreenshotAs(OutputType.BYTES);
        scenario.attach(screenshot, "image/png", "Failed Step Screenshot");
    }
}
```

Summary

| Item | Description |
|----------------------|--|
| Extent Report | Professional, styled HTML report |
| Report Location | target/extent-report/index.html |
| Failure Analysis | Shows failed step, error message, execution time |
| Screenshot Support | Optional with Selenium |

Would you like:

-  To include screenshots automatically?
-  To convert this into a mini Cucumber project ZIP?
-  CI/CD integration for reports in Jenkins?

Let me know how you want to proceed!