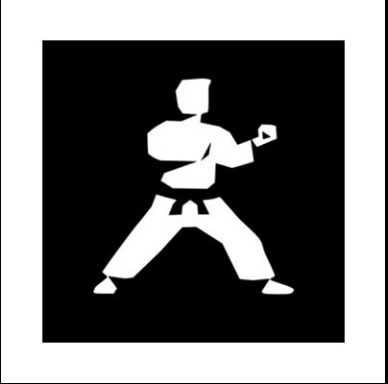
**USER MANUAL**



**Karate Framework**

**What is Karate**

[Karate](https://github.com/intuit/karate) is a relatively new open source framework for testing Web services. Even though Karate is written in Java, its main value proposition is that testers don’t need to do any Java programming in order to write fully automated tests. Instead, testers use a [Gherkin](https://automationpanda.com/2017/01/26/bdd-101-the-gherkin-language/)-like language with steps for making requests and validating responses. It’s like Cucumber with out-of-the-box Web API steps.

**Features:**

**System Configuration:**

Since Karate is an open-source Java project, it can run almost anywhere. We have used following system configuration.

* Windows OS
* Java 1.8
* Apache Maven 3.6.0
* Karate 0.9.5

**IDE:**

The Karate docs recommend using [Eclipse or IntelliJ IDEA](https://github.com/intuit/karate#running-in-eclipse-or-intellij) for developing Karate tests. Both IDEs offer support for JUnit and Cucumber, which Karate can leverage not only for editing but also for running tests. We have used IntelliJ IDEA Community Edition as it provide good support for karate.

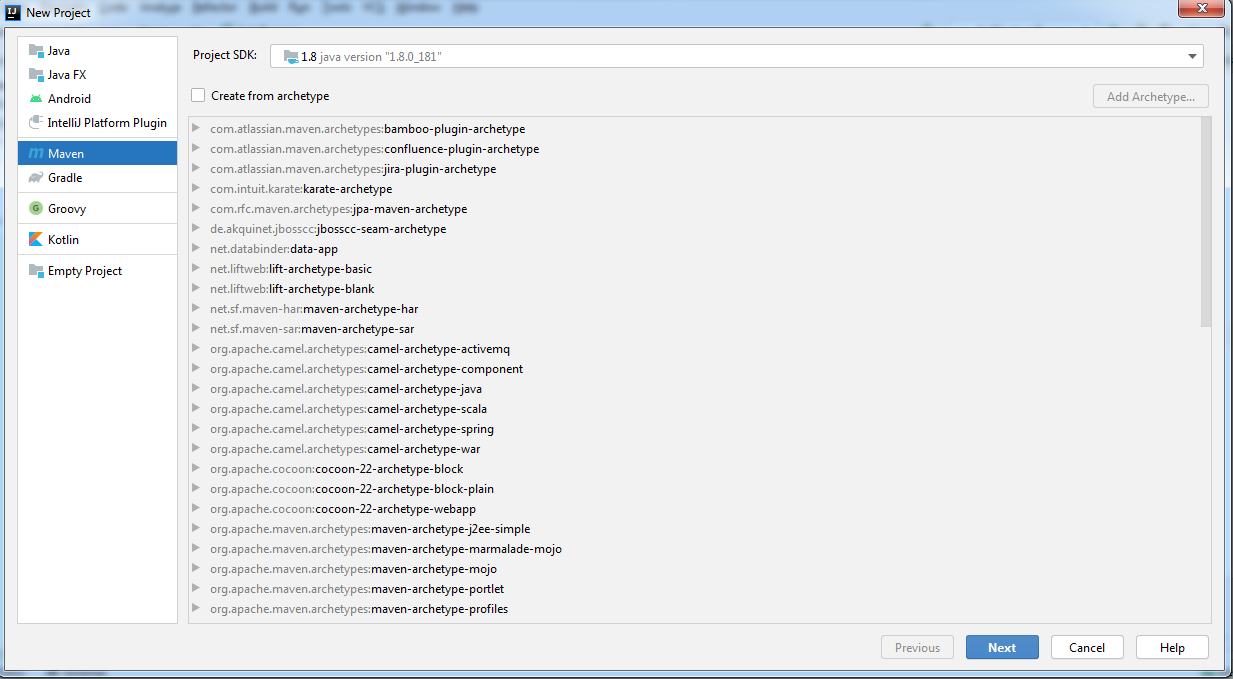
## Project Setup using IntelliJ IDEA Community Edition:

**Install and set up**

1. Intellij IDEA Community Edition
2. Java 1.8 JDK and JRE.

**Steps:**

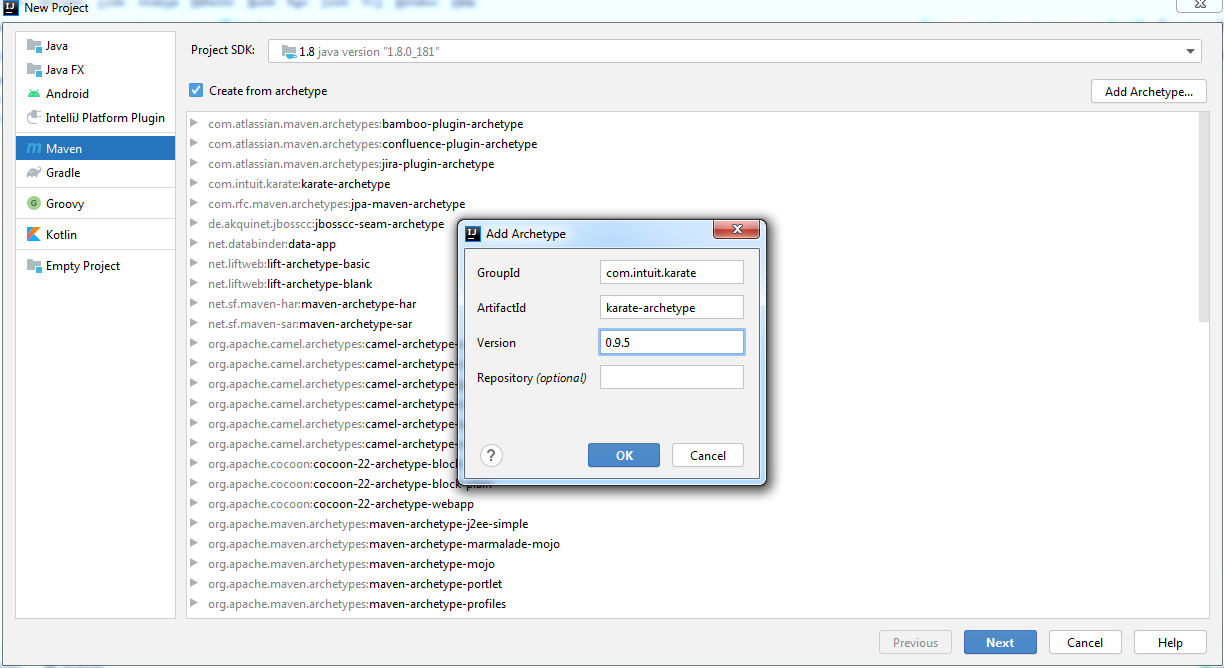
1. Open Intellij IDE and create a new project using Maven.



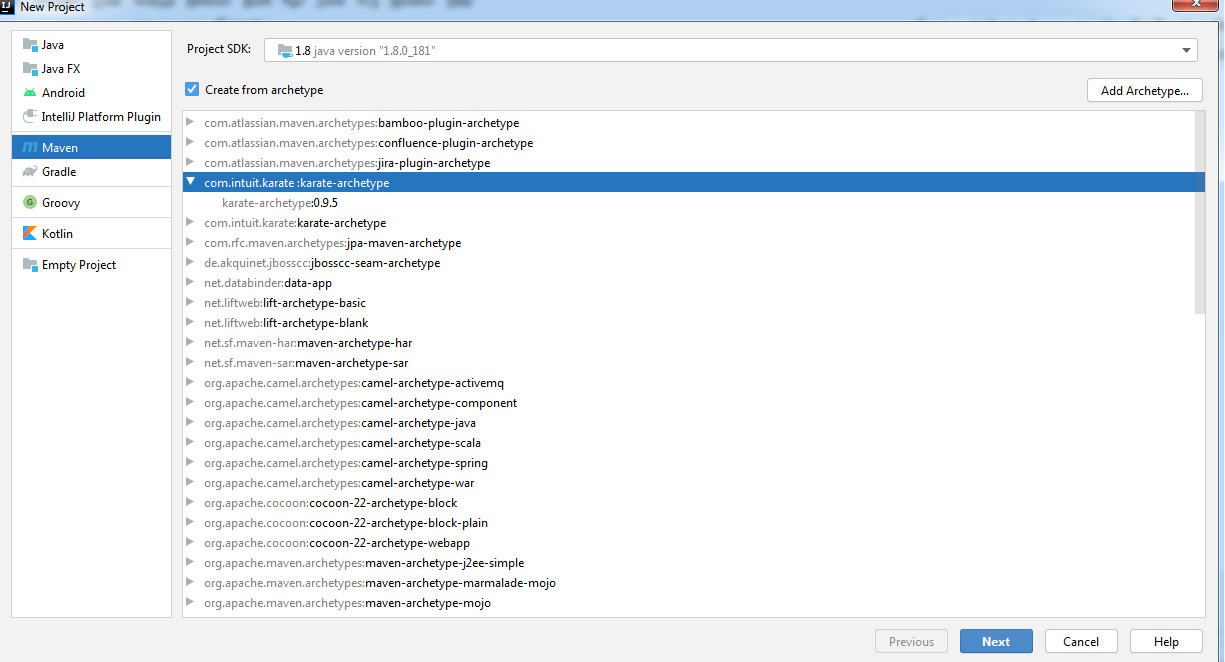
1. Check Create from archetype and click on Add Archetype.

Enter the following information:

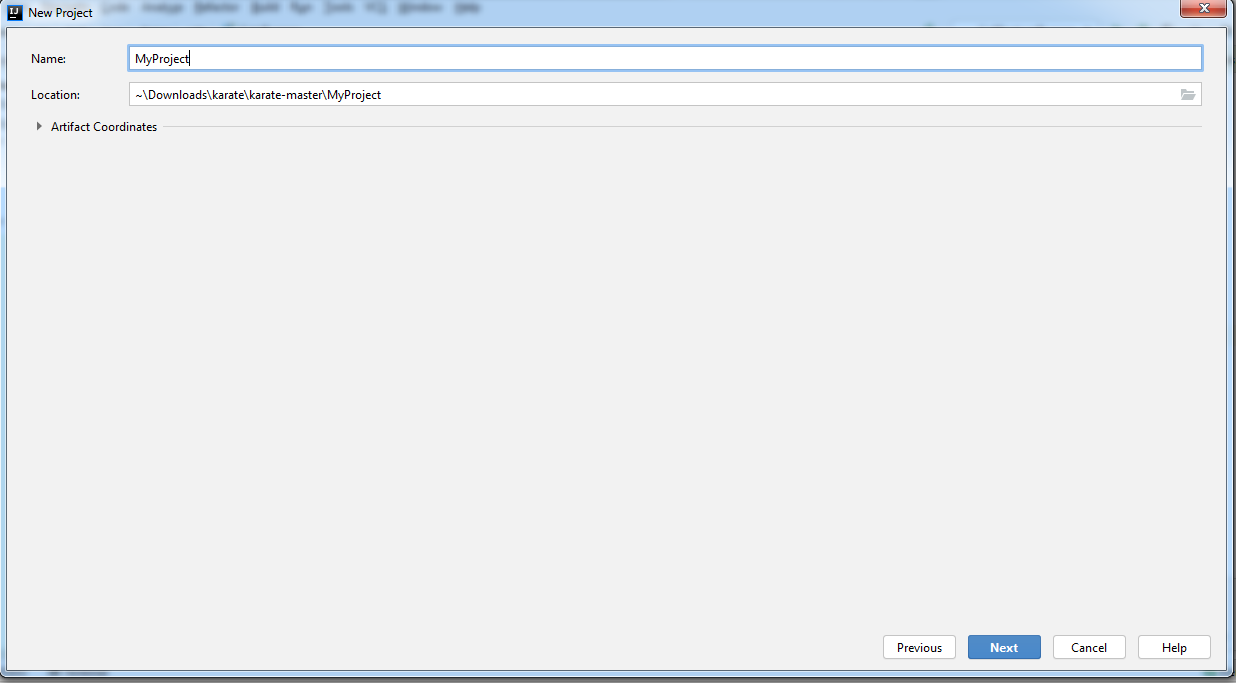
**Group** **Id** *= com.intuit.karate*  
**ArtifactId** *= karate-archetype*  
**Version** *= 0.9.5*



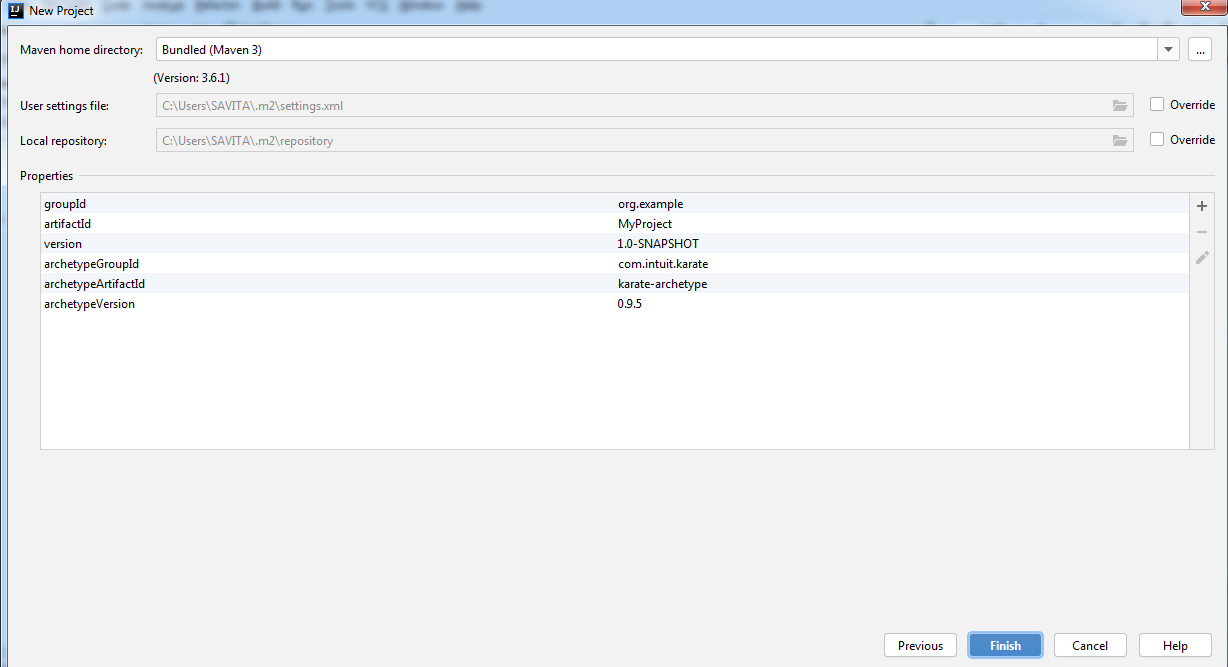
1. Click on OK.
2. It should find the Karate-archetype. Click on Next**.**



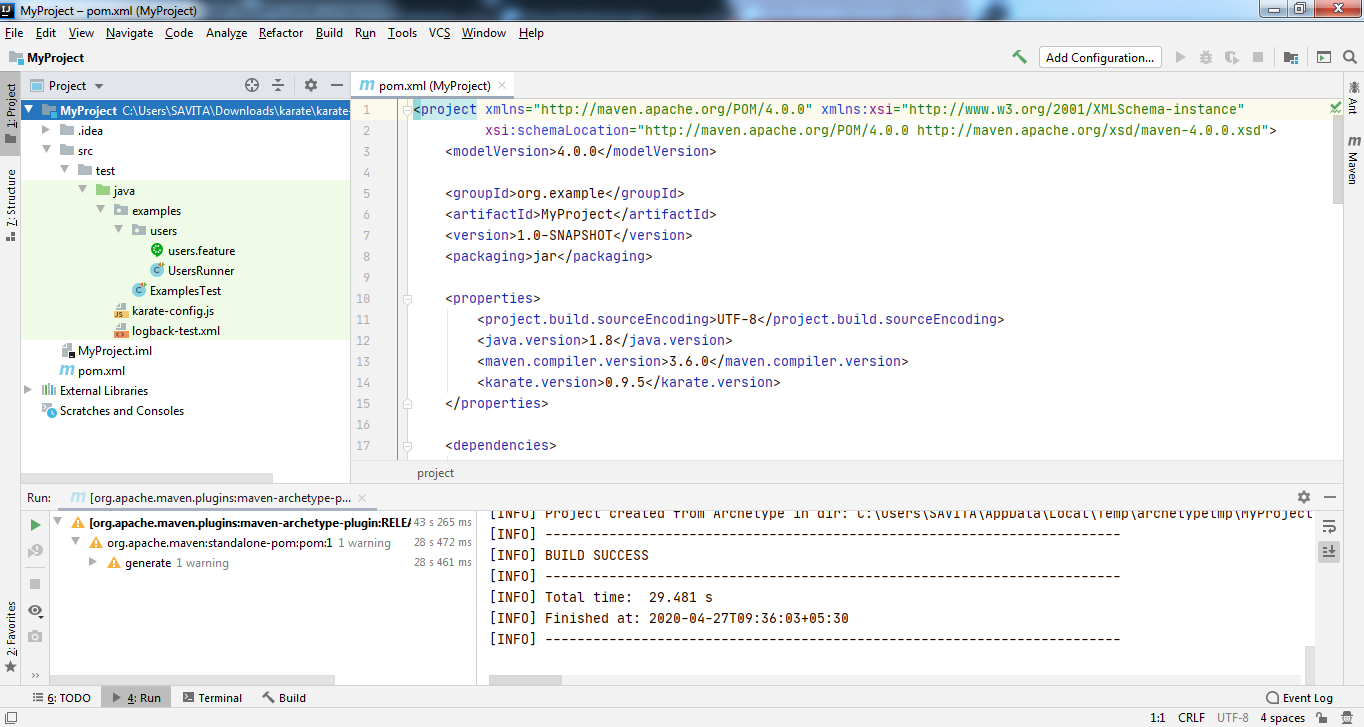
1. Name your project and click on Next



1. Select archetypeVesion 0.9.5 and Click on Finish



1. Project gets created and dependencies get added in pom.xml file.



### Add the dependencies (pom.xml)

<dependencies>  
 <dependency>  
 <groupId>com.intuit.karate</groupId>  
 <artifactId>karate-apache</artifactId>  
 <version>${karate.version}</version>  
 <scope>test</scope>  
 </dependency>

<dependency>  
 <groupId>com.intuit.karate</groupId>  
 <artifactId>karate-junit5</artifactId>  
 <version>${karate.version}</version>  
 <scope>test</scope>  
 </dependency>

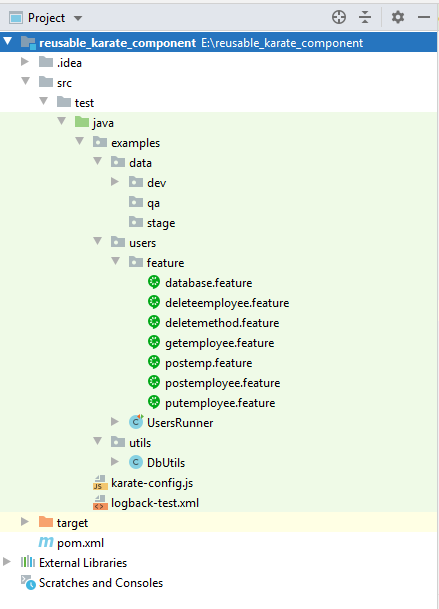
<dependency>  
 <groupId>com.github.javafaker</groupId>  
 <artifactId>javafaker</artifactId>  
 <version>1.0.2</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-jdbc</artifactId>  
 <version>4.3.8.RELEASE</version>  
 <scope>test</scope>  
 </dependency>  
  
 <dependency>  
 <groupId>net.masterthought</groupId>  
 <artifactId>cucumber-reporting</artifactId>  
 <version>4.9.0</version>  
 <scope>test</scope>  
 </dependency>

<dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 <version>8.0.20</version>  
 </dependency>

<dependency>  
 <groupId>org.testng</groupId>  
 <artifactId>testng</artifactId>  
 <version>6.14.3</version>  
 <scope>test</scope>  
 </dependency>  
</dependencies>

### Project Structure:

We have organized and structured our project like this:



data : here we kept test data file as per environment such as dev, qa, stage etc. if you have any other environment to test then create respective folder in data folder and keep the test data

user: here we have maintained feature files and test runner class.

util: here we have maintained database utilities also we can add other utility class.

target: generated reports are saved here.

**karate-config.js**

This is where you can create variables which have a global scope. Karate reads this file before executing any scenario. This comes in very handy when switching environments which specific variables are used for different environments.

Karate-config.js file shows how the UrlBase property is injected into all the test scripts [on startup](https://github.com/intuit/karate#configuration). Also provided the connectTimeout and readTimeout so that the tests 'fail fast' if servers don't respond.

function fn() {

var env = karate.env; // get system property 'karate.env'

karate.log('karate.env system property was:', env);

if (!env) {

env = 'dev';

}

var config = {

env: env,

myVarName: 'someValue',

UrlBase: 'http://localhost:8080/'

}

if (env == 'e2e') {

env = config.env,

config.UrlBase = 'http://localhost:8080/'

} else if (env == 'stage') {

env = config.env,

config.UrlBase = 'http://localhost:8080/'

}

karate.configure('connectTimeout',5000);

karate.configure('readTimeout', 5000);

return config;

}

## Script Structure

Karate scripts are technically in ‘[Gherkin](https://docs.cucumber.io/gherkin/reference/)’ format - but all you need to grok as someone who needs to test web-services are the three sections: Feature, Background and Scenario. There can be multiple Scenario-s in a \*.feature file, and at least one should be present. The Background is optional.

**Feature**: brief description of what is being tested

**Background**:

# this section is optional !

# steps here are executed before each Scenario in this file

# variables defined here will be 'global' to all scenarios

# and will be re-initialized before every scenario

**Scenario**: brief description of this scenario

# steps for this scenario

**Scenario**: a different scenario

# steps for this other scenario.

**Examples:**

**How to send an HTTP Request (Get, Post, Put, Delete )**

This feature file is designed to check returned employees by API.



**Example 1: getemployee.feature**

* Given steps build requests
* When steps make request calls
* Then steps validate responses

**How to read request templates and call other feature files**



**Example 2: postemployee.feature**

In above example we can read input.json file

\* def input = read('examples/data/dev/input.json')

Also we can call scenario from other feature file.

\* def result = call read('getemployee.feature@getEmployeeById')

### Create a Test Runner class

We can execute the scenarios in the feature file using maven (which is useful to run the tests in a CI environment)

package examples.users;  
  
import static org.junit.jupiter.api.Assertions.\*;  
import com.intuit.karate.Results;  
import com.intuit.karate.Runner;  
import java.io.File;  
import java.util.ArrayList;  
import java.util.Collection;  
import java.util.List;  
import com.intuit.karate.junit5.Karate;  
import net.masterthought.cucumber.Configuration;  
import net.masterthought.cucumber.ReportBuilder;  
import org.apache.commons.io.FileUtils;  
import org.junit.jupiter.api.Test;  
  
class UsersRunner {  
  
 @Karate.Test  
 Karate testAll() {  
 return Karate.*run*().relativeTo(getClass());  
 }  
  
 @Test  
 void testParallel() {  
 String karateOutputPath = "target/surefire-reports";  
 Results results = Runner.*path*("classpath:examples/users/feature").tags("~@ignore").parallel(5);  
 *generateReport*(results.getReportDir());  
 *assertEquals*(0, results.getFailCount(), results.getErrorMessages());  
 }  
 public static void generateReport(String karateOutputPath) {  
 Collection<File> jsonFiles = FileUtils.*listFiles*(new File(karateOutputPath), new String[] {"json"}, true);  
 List<String> jsonPaths = new ArrayList(jsonFiles.size());  
 jsonFiles.forEach(file -> jsonPaths.add(file.getAbsolutePath()));  
 Configuration config = new Configuration(new File("target"), "Karate\_Reusable\_component");  
 ReportBuilder reportBuilder = new ReportBuilder(jsonPaths, config);  
 reportBuilder.generateReports();  
 }  
  
}

It also creates a cucumber report for visualizing the results of the test runs.

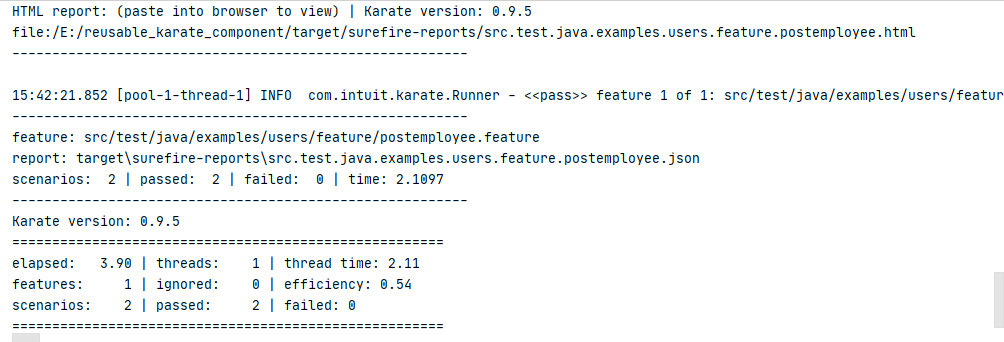
### Run the tests from a command line or CI

mvn clean test -DargLine="-Dkarate.env=staging" -Dtest=UserRunner

Here, we are running the UserRunner class and setting the environment as staging.

**Running Tests**

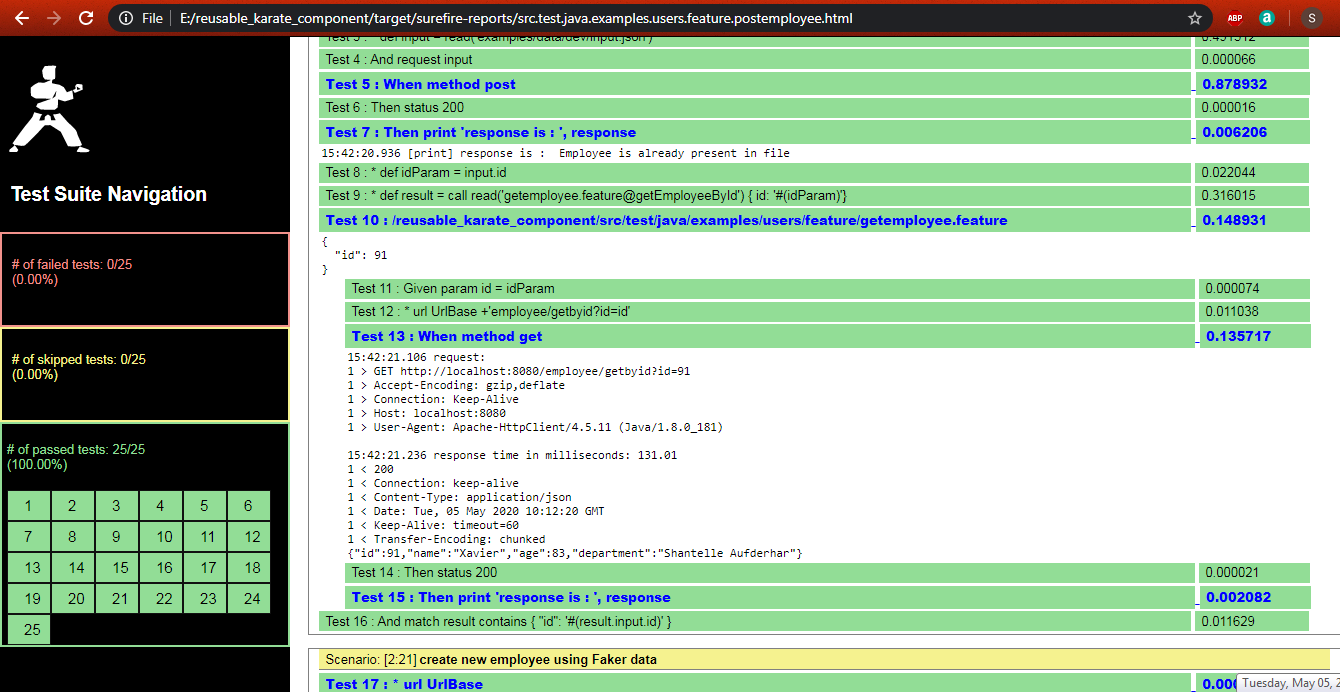
Running the tests will print many lines to the console. Every request and response will be printed. Below is the tail end of a successful run for postemployee.feature:



Karate can generate helpful test reports, too. The project generates JUnit reports by default, but [other report formats like Cucumber](https://github.com/intuit/karate#test-reports) reports are also possible.



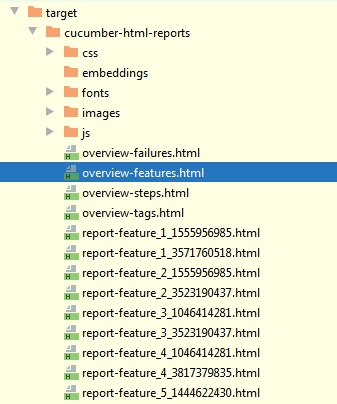
**HTML Report**



**HTML Report Step by step log**

**Test Reports**

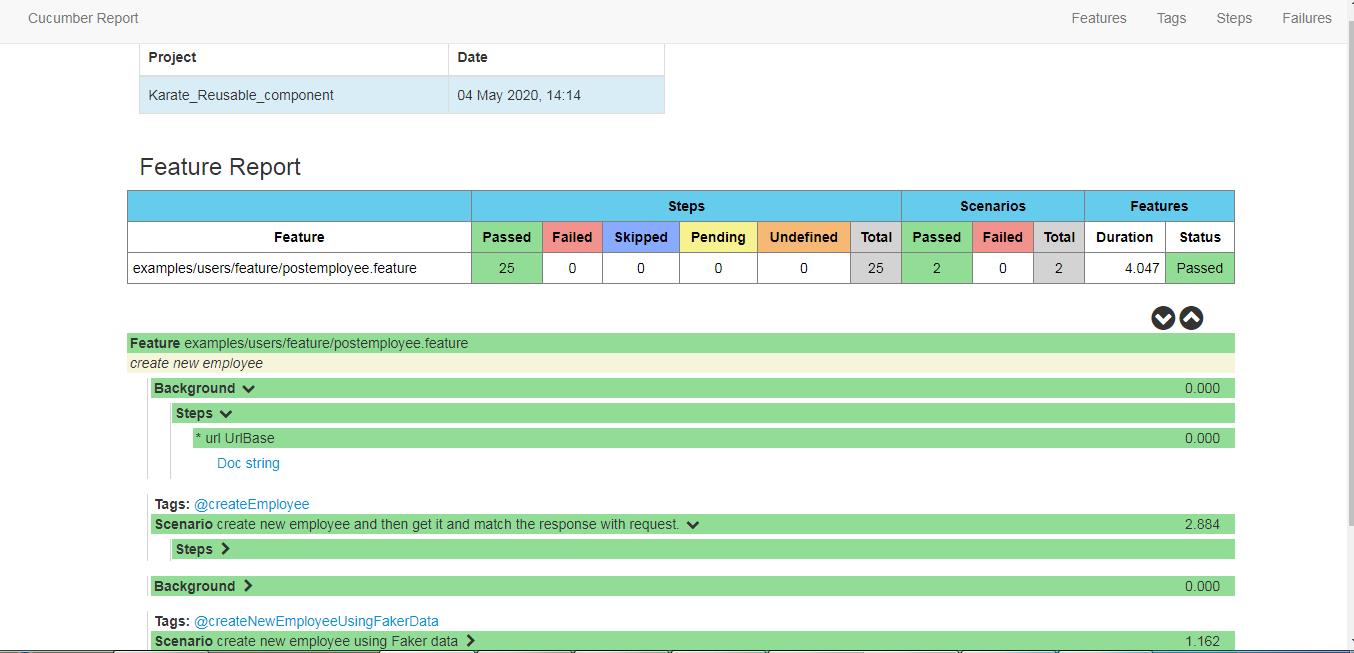
Generated reports are saved in target/cucumber-html-reports



For example, here below is an actual report generated by the cucumber-reporting open-source library.



This report is recommended especially because Karate’s integration includes the HTTP request and response logs in-line with the test report, which is extremely useful for troubleshooting test failures.



**References**

* <https://github.com/intuit/karate>
* <https://devqa.io/karate-api-testing-tool-cheat-sheet/>
* <https://automationpanda.com/tag/karate/>