BDD:

Behavior Driven Development is extension of Test Driven Development and it is used to test the system rather than testing the particular piece of code.

**Cucumber: Used in last 3 projects**

Cucumber is a tool based on Behavior Driven Development (BDD) .

automation of functional validation in easily readable and understandable format (like plain English)

Cucumber feature files can serve as a good document for all. There are many other tools like JBehave which also support BDD framework

**#1) Feature Files**

**Sample feature file:**

**Feature**: Login Functionality Feature

In order to ensure Login Functionality works,  
I want to run the cucumber test to verify it is working

**Scenario**: Login Functionality

**Given** user navigates to SOFTWARETETINGHELP.COM  
**When** user logs in using Username as “USER” and Password “PASSWORD”  
**Then** login should be successful

**Scenario**: Login Functionality

**Given** user navigates to SOFTWARETETINGHELP.COM  
**When** user logs in using Username as “USER1” and Password “PASSWORD1”  
**Then** error message should be thrown

Feature: purpose of Application under test

**Scenario:** particular functionality which is under test.

Format:

Given-When-Then==

Background(If user needs to clear database ) ==And(combine two or more same type of action)

**Scenario Outline:** Scenario outlines are used when same test has to be performed with different data set

**Junit Runner:**

To run the specific feature file cucumber uses standard Junit Runner and specify tags in @Cucumber. Options. Multiple tags can be given by using comma separate. Here you can specify the path of the report and type of report you want to generate.

**Example of Junit Runner:**

|  |  |  |
| --- | --- | --- |
| 1 | import cucumber.api.junit.Cucumber;</pre> | |
| 2 | import org.junit.runner.RunWith; |

|  |  |  |
| --- | --- | --- |
| 3 | | @RunWith(Cucumber.class) |
| 4 | @Cucumber.Options(format={"SimpleHtmlReport:report/smokeTest.html"},tags={"@smokeTest"}) | | |

|  |  |  |
| --- | --- | --- |
| 5 | Public class JUnitRunner { | |
| 6 | } |

Similarly you can give instruction to cucumber to run multiple tags. Below example illustrates how to use multiple tags in cucumber to run different scenarios.

|  |  |  |
| --- | --- | --- |
| 1 | import cucumber.api.junit.Cucumber; | |
| 2 | import org.junit.runner.RunWith; |

|  |  |  |
| --- | --- | --- |
| 3 | | @RunWith(Cucumber.class) |
| 4 | @Cucumber.Options(format={"SimpleHtmlReport:report/smokeTest.html"},tags={"@smokeTest",”@LoginTest”}) | | |

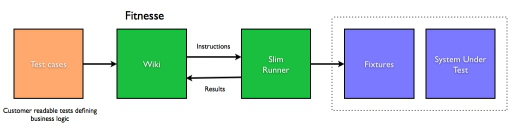
|  |  |  |
| --- | --- | --- |
| 5 | Public class JUnitRunner { | |
| 6 | } |

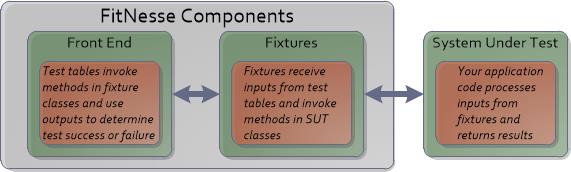
**Fitnesse: Used In 2 projects In another resume**

automated framework

software development by providing a WIKI powered test framework which enables customers, testers and programmers to easily create and edit tests in a platform independent way.

**FitNesse Overview**





**Working:**

FitNesse works by executing Wiki pages which call custom written Fixtures.

Whenever a Wiki test is executed, the Fixtures works by calling the System Under Test (SUT) with the appropriate parameters, execute a piece of business logic in the software system, and pass the results (if any) of the SUT back to the Wiki(**Wiki is a piece of server software that allows users to freely create and edit Web page content using any Web browser. Wiki supports hyperlinks and has a simple text syntax for creating new pages and crosslinks between internal pages on the fly.Wiki is unusual among group communication mechanisms in that it allows the organization of contributions to be edited in addition to the content itself.**

) front-end, which in turn will visually indicate if a test has passed or not.

**Fixture:** Fixtures are a bridge between the Wiki pages and the System Under Test

FitNesse has two test systems, SLIM and FIT:

**FIT:** FIT is the older test system, and is no longer actively developed.

**SLIM:** lightweight version of the FIT protocol. One of the design goals of SLIM was to easily port implementations to different languages. Also, in contrast to FIT, SLIM doesn’t require any dependencies on the FitNesse framework in the Fixtures code, which makes writing fixtures more easy.

**FitNesse Suite:** Suites are sets of Tests, which is a way to organize the Tests. As an additional benefit, when executing a Suite, all Tests within the Suite are executed.

**Creating the Suite:**

To create a new FitNesse suite:

* Go to the FitNesse Front page at [http://localhost:9090](http://localhost:9090/)
* Click Edit in the left menu
* Type the name of the Suite after the existing text, e.g. JukeboxSuite

Important: FitNesse will only create links when the text is written in CamelCase. This means that every page needs to start with an uppercase character and at least one other letter in the word is written in uppercase.

* Click save
* Click on the question mark [?] next to the JukeboxSuite text
* Press save

An empty Suite has now been created, which is indicated by the ‘Suite’ text on top of the left menu.

**Note:** FitNesse marks a page as a Suite automatically when it starts or ends with Suite. A Wiki page can also manually be set as an Suite in the page properties by clicking ‘Properties’.

**Creating the test:**

The Decision Table is the default test table style used by FitNesse. When not specifying a table prefix, FitNesse decides it is a Decision Table. An example Decision Table is used below to assert the proper conversion of payments into credits.

Creating a Test is similar to creating a new Suite:

When in a Suite, click ‘Edit’ in the left menu

Clear the text area and type the name of the Test, e.g. PaymentTest

Click save

Click on the question mark [?] next to the PaymentTest

Clear the text area and replace the text by the following:

!4 Story: the amount you pay determines the received credits.

!|credits for payment|

|payment|credits?|

|.25 |1 |

|1 |4 |

|5 |20 |

(There are multiple types of test styles, but the above is a Decision Table)

Press ‘Save’

Your first test has now been created, including some markup. This markup is ignored when executing the Test; only tables are executed. When you execute this test by clicking on ‘Test’ in the left menu, your test will fail with an error. To get the test to work, we need to do two more things: write the Fixture and configure FitNesse correctly.

**Writing the Fixture**

The Fixture will be the layer between the production code (the Subject Under Test) and the FitNesse pages. There are multiple types of Fixtures, and to support the Test above, a Decision Table Fixture is needed. Consider the following code to test:

package jukebox.sut;

public class JukeBox {

public int calculateCredits(double payment) {

return payment \* 4;

}

}

The Fixture to test this class looks like this:

package jukebox.fixtures;

import jukebox.sut.JukeBox;

public class CreditsForPayment {

private double payment;

private int credits;

public void execute() { // executed after each table row

this.credits = new JukeBox().calculateCredits(payment);

}

public void setPayment(double payment) { // setter method

this.payment = payment;

}

public int credits() { // returning function because of question

mark in the test return this.credits;

}

}

The Fixture is created from the FitNesse page, and for each row:

* First the setters are called (in this case setPayment),
* Then the execute is called to do call the SUT
* Then the result is retrieved from the Fixture and compared to the FitNesse expectation.

**Slim Test Table Styles**

**WSDL stands for Web Services Description Language. It is the standard format for describing a web service. WSDL was developed jointly by Microsoft and IBM.**

**Features of WSDL**

* **WSDL is an XML-based protocol for information exchange in decentralized and distributed environments.**
* **WSDL definitions describe how to access a web service and what operations it will perform.**
* **WSDL is a language for describing how to interface with XML-based services.**
* **WSDL is an integral part of Universal Description, Discovery, and Integration (UDDI), an XML-based worldwide business registry.**
* **WSDL is the language that UDDI uses.**
* **WSDL is pronounced as 'wiz-dull' and spelled out as 'W-S-D-L'.**