HELiOS

HeliOS Framework

Newest version of the Social Media Framework

Version 2.1
March 2013

This document applies to HeliOS Framework v2.1 and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

TABLE OF CONTENTS

1 About This Guide

The purpose of this guide is to assist developers, testers, release engineers, managers and others aiming to use HeliOS Framework user interface for maintaining and controlling the critical phases in the project lifecycle.

1.1 Document Conventions

Table 1: Conventions

Convention	Description
Green	Identifies elements on a screen
	Note
Blue	HeliOS UI Navigation (Breadcrumbs)
Brown	Identifies tabs on a screen
Italic	Code structure
*	This is mandatory symbol

2 Introduction

HeliOS, a product of Photon InfoTech Pvt Ltd, is designed to work seamlessly with the powerful and more popular technologies; HeliOS assists users in creating projects that boast persistent uniformity in quality across platforms throughout the lifecycle of a project. Aiming to be the go-to tool for next generation multichannel development, HeliOS allows the user to develop projects simultaneously with faster cycles across multiple channels. HeliOS is endowed with latest tools and capabilities along with expanding libraries of pre-built templates, standardized codes and manifold archetypes which aid the development team to increase their capability by limiting the occurrences of errors and reducing the development time. Projects created using HeliOS Framework adheres to the best practices in the industry making them resourceful, effective and reusable.

2.1 What Is HeliOS Framework?

HeliOS is a next-generation development framework of frameworks. It is a platform for creating next generation web, mobile and Multichannel presences leveraging existing investments combined with accepted industry best practices. HeliOS publishes new artifacts (components) in the centralized maven repository or Customer specific repository under appropriate technologies. Once published in the centralized repository, subscribed users can view and deploy those artifacts in their projects.

HeliOS encapsulates the best practices of a project structure, re-usable components, standards, documentation, quality assurance and release process. This ensures the quality of a project deliverables, which in turn will increase the productivity of a project. Though HeliOS guarantees projects adhering to best practices, it permits every organization to follow their respective quality measures and standards.

3 HeliOS – Powered With Facets

3.1 Lifecycle Management

Maintaining end to end lifecycle of a project has never been easier.

From providing a stepwise method in creating a template project to integrating tools for code validation, build and deployment and continuous improvement, HeliOS makes sure that every aspect of a project lifecycle is taken care of. The integration of HeliOS with multiple open source projects allows the user community to reap the full benefits of the Framework.

3.2 Pre - Built Architecture

HeliOS provides a pre-built architecture model with certified template architectures such as Multichannel widget, SharePoint web parts and NodeJS Service gateway. It contains standard build structures with supported versions.

3.3 Quality Assurance

HeliOS contains powerful automated testing instruments designed to analyze and test the project at the code level. The testing methods deployed in HeliOS include:

- Unit Testing
- Functional Testing
- Performance Testing
- Load Testing

By using static code analysis tools to pick up and compare the metrics of different technologies with the source codes, HeliOS makes sure that the project is of quality standards. HeliOS Archetypes, created by following the best practices in the respective domains, help the developers in creating model projects by providing basic templates for any desired technology.

3.4 Web Development

HeliOS offers support for web development tools such as ASP.NET, SharePoint, SiteCore (an additional plugin for creating SharePoint projects), PHP (raw), PHP (Drupal), WordPress, HTML5 YUI Mobile Widget, HTML5 JQuery Mobile Widget, Html5 Multichannel YUI Widget, HTML5 Multichannel JQuery Widget, HTML5 and Java standalone. It also supports web applications using Responsive web design and Java / NodeJS based web services.

3.5 Mobile/Web Application Development

HeliOS patronage of the Responsive web design concept makes the mobile/web application development for iPhone, Android, Blackberry and Windows 8 unproblematic and resourceful.

3.5.1 iPhone Features

- **IOS Platforms**

  To test IOS applications, HeliOS framework allows testers to select the required devices and its versions. HeliOS finds out the installed versions of the devices available in the system and only those versions are shortlisted.

- **IOS Devices**

  While deploying and testing the IOS application, HeliOS Framework provides the option of selecting iPhone simulator or iPad simulator from the dropdown box listed under family. This is done by integrating waxsim plugin in the Framework

---

Note:

- Waxsim tool should be installed manually from the directory path phresco-framework/workspace/tools/waxsim.
- The command xcodebuild install DSTROOT= from the command prompt will install the tool to the local machine.

- **iPhone Workspace**

  The iPhone Workspace is an Xcode project for integrating multiple iPhone Native and iPhone library apps. HeliOS offers support throughout the development cycle of a Native application culminating with the generation of .app file.

- **Iphone Library**

  HeliOS allows for the creation of a static library and offers code validation and build generation support for the same. The newly created iPhone library can be used in any application and cannot be deployed separately.

3.5.2 Android Features

- **Proguard Enablement**

  The source code can be decompiled and viewed from the installed application. To keep it in encrypted format, Proguard enablement shuffles the java classes and gives an ID to each class. The Class and its ID will be unrelated; i.e., the class will be shuffled and will be renamed as a, b, c, d.

- **Application Signing**

  Android system requires that all installed applications be digitally signed with a certificate whose private key is held by the application's developer. In order to make application available in Android store (Market), the application has to be certified. HeliOS allows user to select certificate through the HeliOS UI while build the app for final release.

- **NDK Support**

  Native codes can also be used as an android application when the compiled native code is copied in the path <HeliOS_HOME>/workspace/projects /PHR_Android/source/libs>. HeliOS does not directly support NDK.

- **Android Library**

  HeliOS offers support to create an Android Library project. An Android Library is a development project that holds shared Android source code and resources. Any Android application project can reference the Library and, at build time, include its compiled sources in their .apk files.

Multiple application projects can reference the same Library and any single application project can reference multiple Libraries.

3.6 Standalone Application Development

Standalone application does not require any software other than the operating system to run. This support is provided in the latest version of HeliOS, where java standalone applications can be created.

3.7 Multichannel Presence

The Burgeoning user base of Internet has forced organizations to provide customers a seamless environment for buying or utilizing their services in a manner that the channels complement each other.

Having this in mind, HeliOS has enabled Widgetized Responsive web design to be accessed across various channels.

3.8 Responsive Web Design

When creating a web page, it's been a common tactic to serve up two different sets of pages - one for desktop browsers, another for mobile devices. With a wide variety of mobile devices, screen sizes and hardware features, designing a separate version for each quickly becomes impractical.

HeliOS deploys the Responsive Web Design (RWD) concept to adjust the layout and features of a website based on how it's being viewed. HeliOS with the aid of RWD helps developers in exposing the myriad functionalities developed using HeliOS across multiple channels i.e., web, mobile, tablet and kiosk with minimum resizing, panning and scrolling efforts.

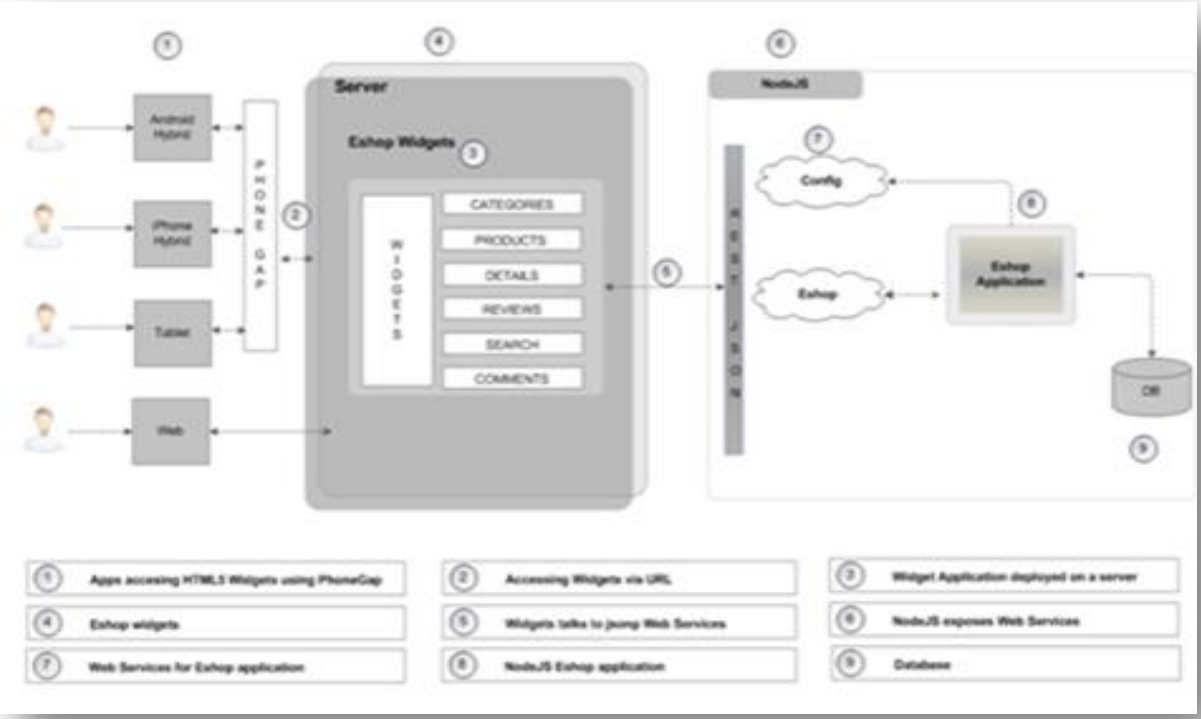

Figure 3-1: Multichannel Architecture

3.9 Reuse and Continuous Improvement

For any organization, promoting reusability of the components is important. The components (artifacts) such as libraries, features, web parts, archetypes, pilot projects and other artifacts can be reused in numerous projects with minimal changes.

3.10 Continuous Integration

One of the cherished aspects of HeliOS is the Continuous Integration feature made available as part of the Framework which generates the build automatically deploys it and tests the build in periodic manner. With this feature, it is trouble-free to keep track of the latest build updates and make sure that the builds adapt seamlessly to the existing project which is otherwise a tedious process. Added advantage in the HeliOS is multiple job creation. Multiple jobs can be created to a single project and it can be linked with each other to maintain the order of the automation.

4 Getting Started With HeliOS Framework

Once JAVA_HOME path is set and jdk 1.6.0_18 version is installed, it's time to extract, install and start creating projects using HeliOS.

4.1 Supported Platform

- Windows: xp, 2003, 2008, windows 7 (32 and 64 bits are supported), Windows 8 (32 and 64 bits are supported)
- Mac: Mac OS X Lion, OS X Mountain Lion
- Linux: RedHat, SUSE Linux, Ubuntu

4.2 Supported Browsers

- Firefox: Version 6 and above
- Google Chrome: Version 11 and above
- Internet Explorer: Version 9 and above
- Opera: Version 10 and above
- Safari: Version 5 and above

4.3 Steps To Extract and Execute HeliOS Framework

- Download and unzip the Phresco build <version>.zip file containing HeliOS Framework
- For Windows, Run <HeliOS-framework>/bin>start-framework-server.bat from the command prompt. This will automatically open HeliOS Framework in the browser. (User's System Default Browser)
- For Mac/Linux, Run <HeliOS-framework>/bin>sh start-framework-server.sh from the command prompt. This will automatically open HeliOS Framework in the browser. (User's System Default Browser)

4.4 Logging On To HeliOS Framework

- In the Web Browser, enter the URL where HeliOS Framework is hosted
- At the log in screen, enter organization's LDAP credentials
- Now click **Login**

4.5 Using the Navigation Controls in User Interface

HeliOS interactive navigation controls aid users in moving seamlessly through the Framework guiding them in achieving the optimum result.

After logging in, users will find a simple and easy way to navigate to the Home, Projects and Settings or Help menus.

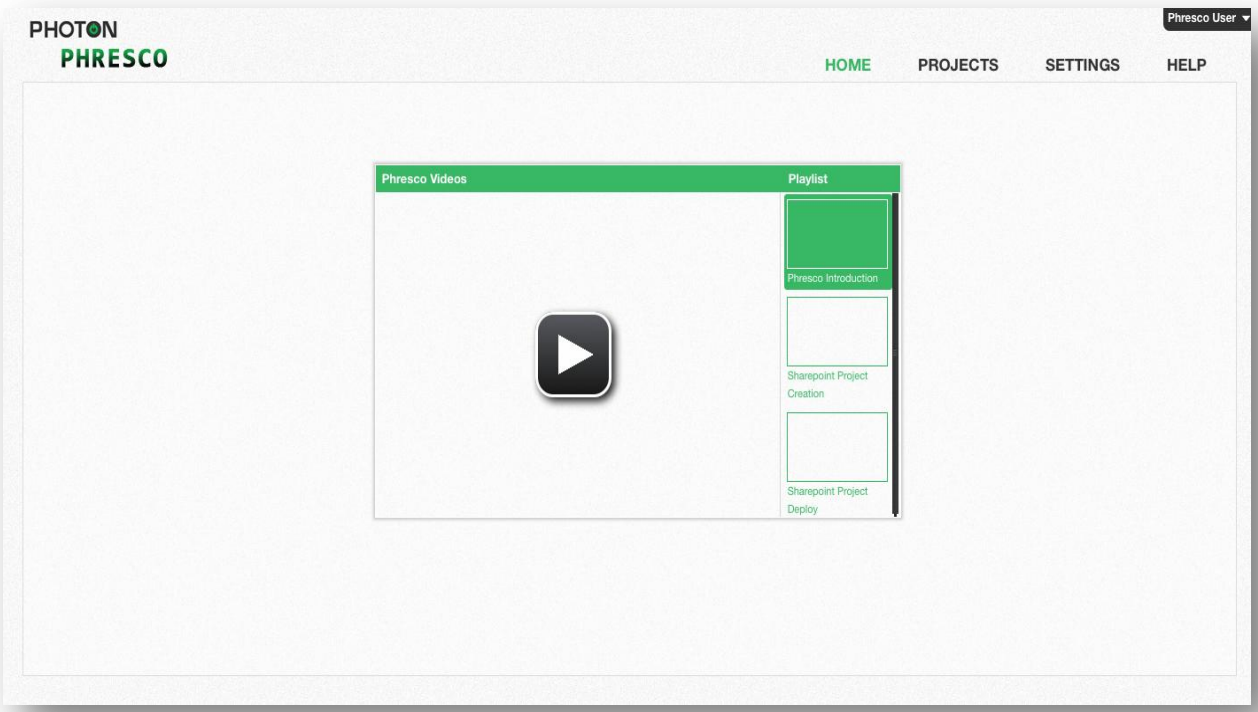

Figure 4-1: Home page

4.6 HeliOS Videos

The videos provide a gist of what HeliOS is all about and also gives a step by step audio-visual representation of how to browse through HeliOS. It helps the user with a clear picture as how to work on HeliOS framework. For Example: Creating an application, configurations of applications, testing a project etc.

HeliOS video page is readily obtainable when **Home** is clicked. The playlists mostly has the complex working experience of any project.

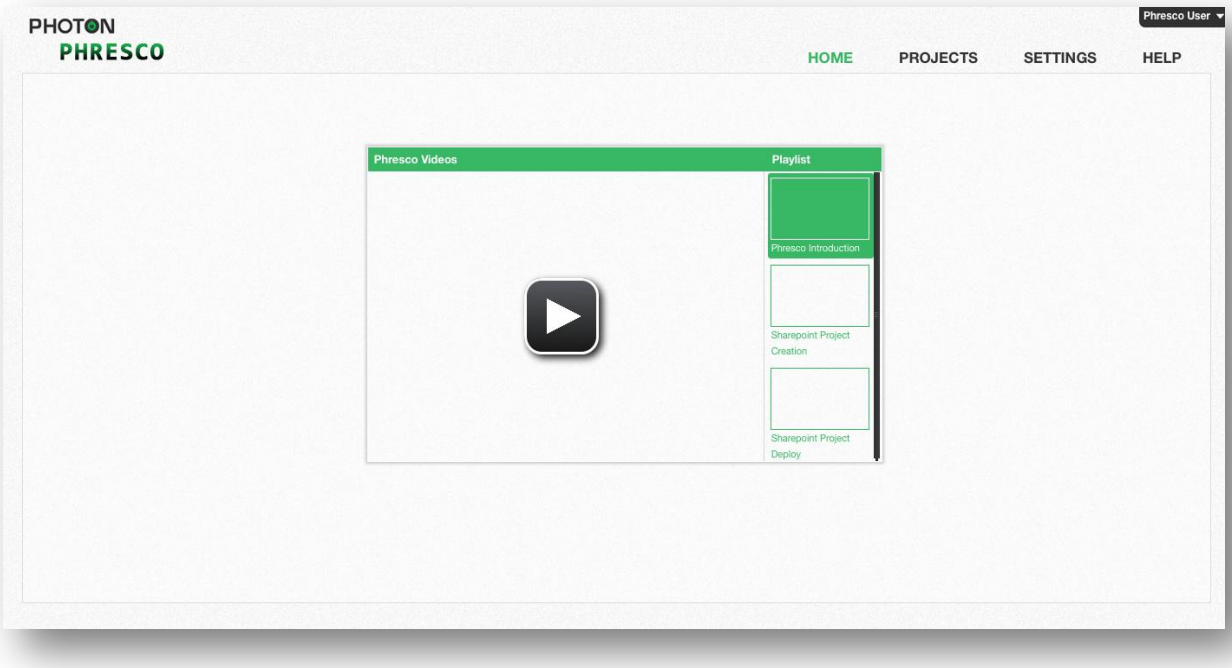

Figure 4-2: Video library in home page

4.7 Logging Out Of HeliOS Framework

To log out of HeliOS Framework, click **Sign Out** (located in the drop down list on the right top corner of the page).

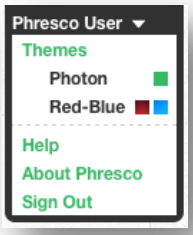

Figure 4-3: Change skin, Help, About HeliOS and Sign Out

4.8 To Change the Skin Color of Your Account

Photon theme is the default skin for the framework. Users can also choose from two more different skins (Red and Blue) available at the right top coner (mouse over the user account) of the page in the HeliOS Framework.

4.9 To Update the Available Version

HeliOS releases periodic updates for the Framework. Users can update to the latest release by clicking the **Update Available** button in the **About Phresco** link.

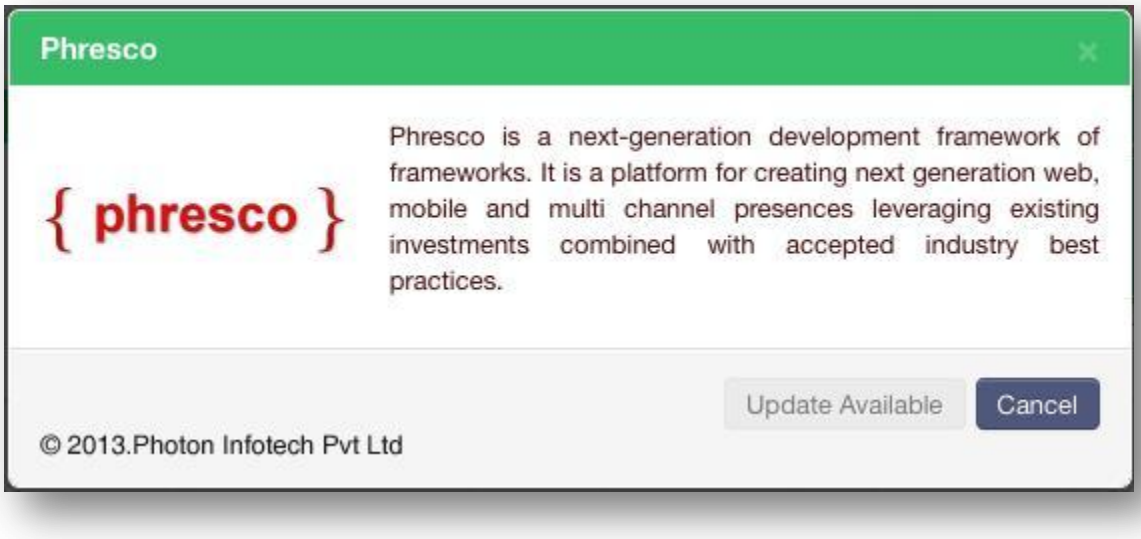

Figure 4-4: About HeliOS

Note:

- Only those using the earlier versions of the HeliOS Framework 2.0.X can update to the latest 2.0.X
- The **Update Available** button will be enabled only on the existence of a newer version of the Framework currently being used

---

5 HeliOS Project Life Cycle

5.1 Projects

HeliOS aids in developing web, mobile and web service applications based on any given project requirement. The HeliOS created project contains a well defined archetype (Project structure), features, a testing structure and help documents. Out of the box features can be adapted to the project based on the requirements. On specific requests, additional features can be added to the framework by Photon on validation of the license.

5.1.1 Creating a Project

HeliOS offers support in creating projects for multiple technologies. A new project can be created from the **Projects→Add Project**. Depending upon the requirements, projects can be created simultaneously in the application, mobile and web layers and they will be grouped under one umbrella.

Select the customer from the dropdown box present at the top right corner to create a project for that customer. This way the projects can be grouped based on the customers.

Fields in the Add Project are as follows

Name

It denotes the Name of the project created. Maximum text input of the field is restricted to 30 alphanumeric characters and is a mandatory field.

Code

It denotes the code of the project created. Maximum text input of the field is restricted to 12 alphanumeric characters and is a mandatory field.

Description

It denotes the description of project created. Maximum text input of the field is restricted to 150 alphanumeric, special characters and is not a mandatory field.

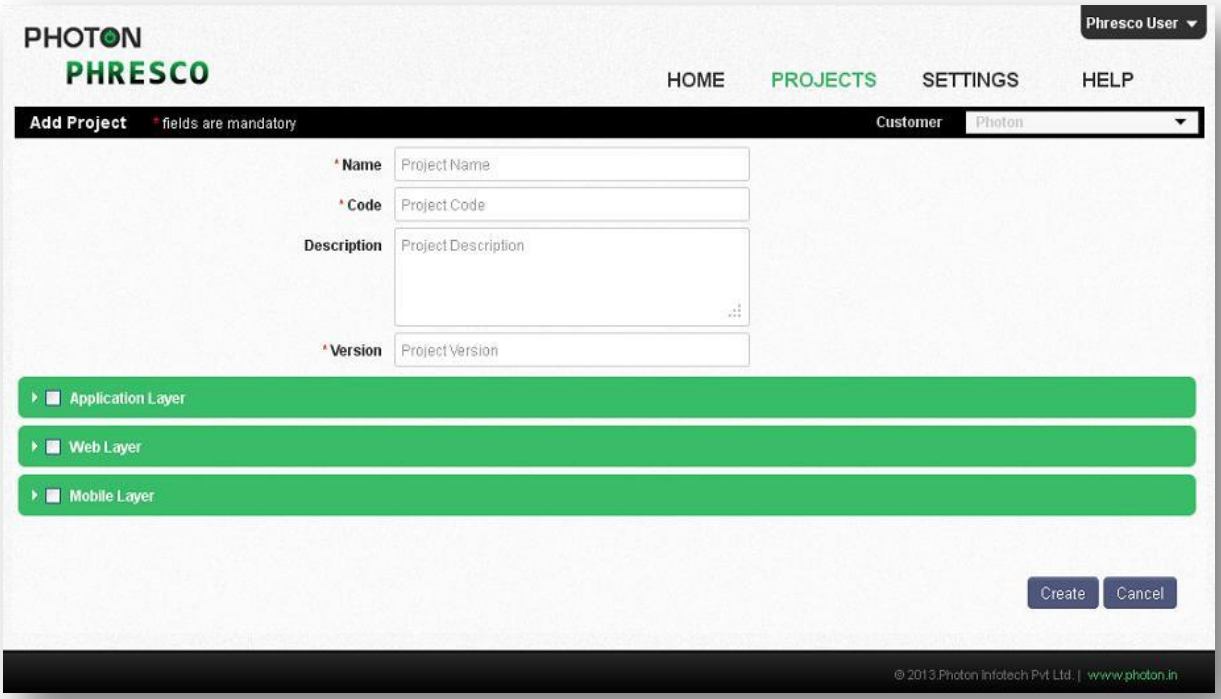

Figure 5-1: Creating a Project

Version

It denotes the version of the project created. Maximum text input of the field is restricted to 20 alphanumeric characters with a special character ".". It is a mandatory field.

Application Layer

This layer contains the list of HeliOS supported application technologies and their versions. The technologies supported include,

S.No	Technology	Version
1	ASP.NET	3.5, 3.0, 2.0
2	WordPress	3.4.2, 3.3.1
3	PHP	5.4X, 5.3X, 5.2X, 5.1X, 5.0X
4	Drupal 7	7.8
5	Drupal 6	6.3, 6.25, 6.19

6	Java WebService	1.6, 1.5
7	Java Standalone	1.6, 1.5
8	SharePoint	3.5, 3.0, 2.0
9	Site Core	3.5, 3.0, 2.0
10	NodeJs WebService	6.14, 6.11, 6.8, 6.7, 6.1

Web Layer

This layer contains the list of HeliOS supported Web technologies and their versions.
The Web technologies supported include,

S.No	Web Layer	Widget	Version
1.	HTML5	HTML5 JQuery mobile widget	1.6, 1.5
		HTML5 Multichannel YUI Widget	1.6, 1.5
		HTML5 Multichannel JQuery Widget	1.6,1.5
		HTML5 YUI Mobile Widget	1.6, 1.5

Mobile Layer

This layer contains the list of HeliOS supported Mobile technologies and their versions.
The Mobile technologies supported include,

S.No	Technology	Type	Version	Device
1	Android	Android Native	4.2, 4.1.2, 4.1, 4.0.3, 2.3.3, 2.2, 2.1_r1, 1.6	Phone, Tablet
		Android Hybrid	4.2, 4.1.2, 4.1, 4.0.3, 2.3.3, 2.2, 2.1_r1, 1.6	
		Android Library	4.2, 4.1.2, 4.1, 4.0.3, 2.3.3, 2.2, 2.1_r1, 1.6	
2	Blackberry	Hybrid	7.X	Phone, tablet

3	iPhone	iPhone native iPhone hybrid iPhone Library iPhone Workspace		Phone, Tablet
4	Windows	Windows Metro Windows Phone	 7.X	Phone, Tablet

5.1.1.1 Embed functionality

Select this option in the mobile layer to perform build generation for the web layer application along with the mobile layer. It will be automatically extracted into the www folder available in the mobile application folder structure.

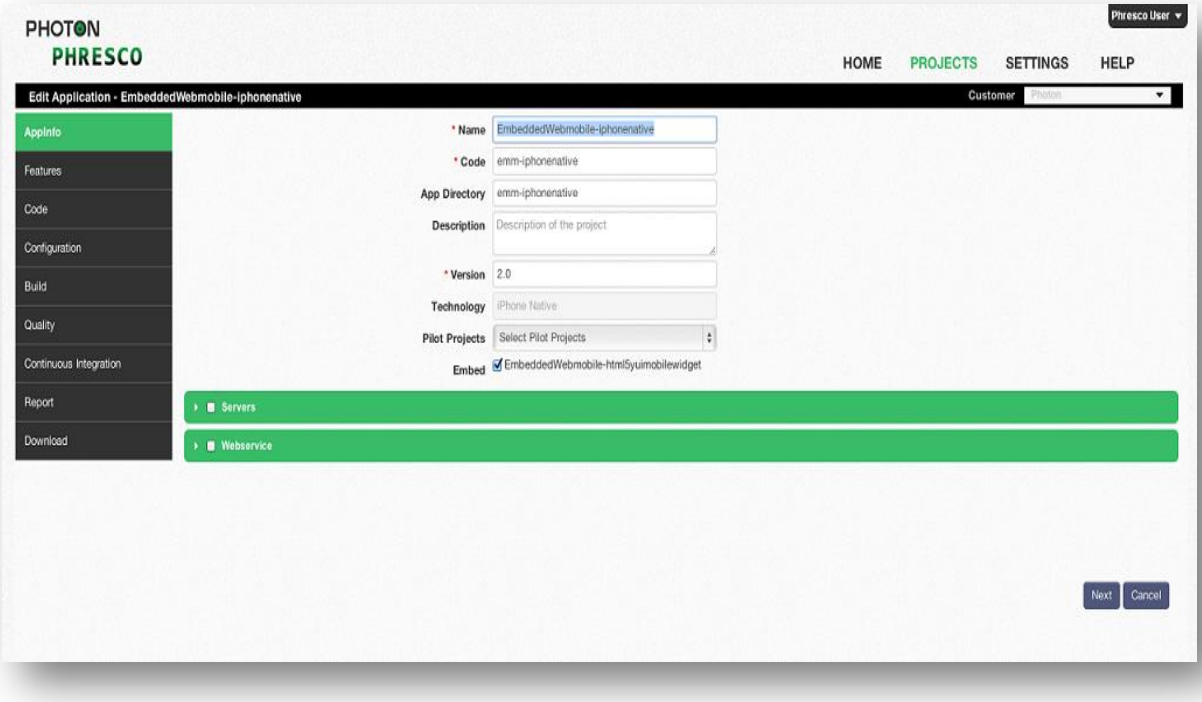

Figure 5-1 (a): Embed Functionality

5.1.2 Editing a Project

HeliOS permits editing certain aspects of existing project. This includes

- The general description of the project
- Adding new applications to the existing project

5.1.3 Import Application

Apart from creating projects using HeliOS, developers can also create their project with HeliOS standards and import them into HeliOS Framework.
Import Application will automatically check out a project in the desired path of the folder (*Phresco-framework-1.0\Phresco-framework\workspace\projects*)
SVN

Import from SVN is used for importing a replica of a project. Developers can check in and make changes to a project which will be reflected in the main project. Conflictions may occur if two developers check in at the same time. Only those projects developed using the HeliOS standards can be imported from the SVN.

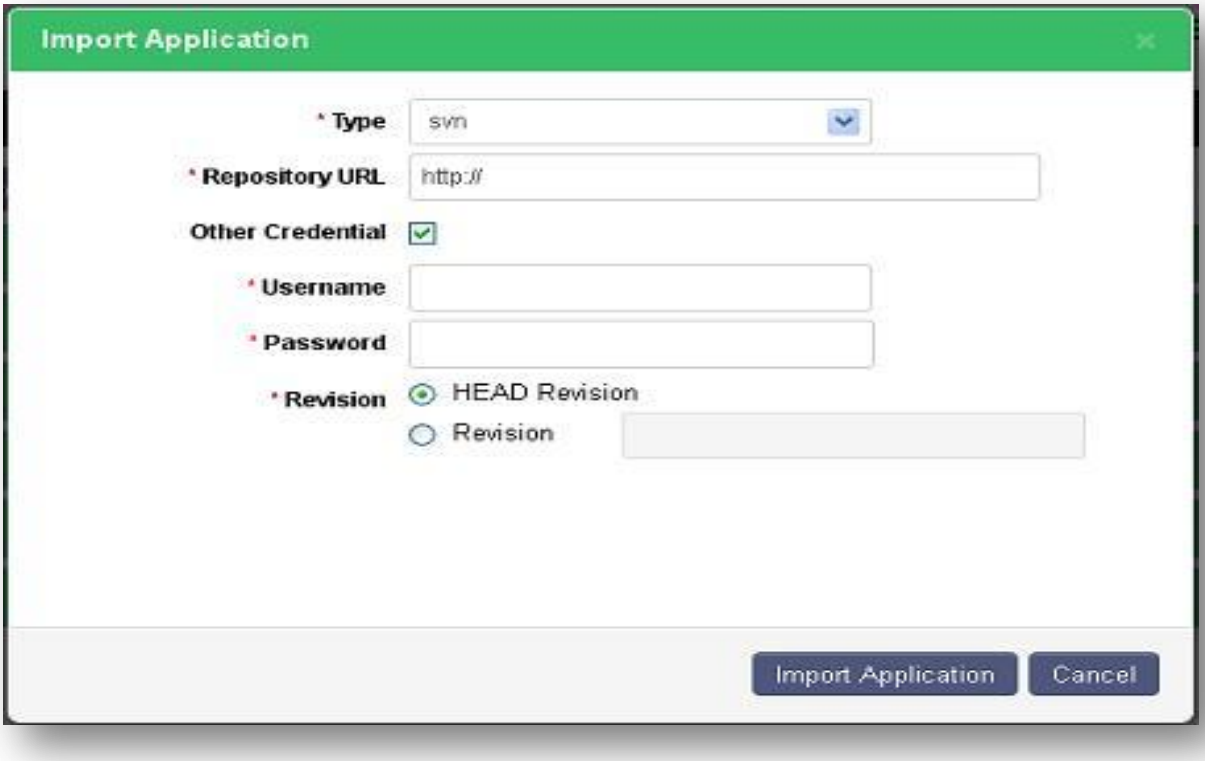

Figure 5-2: Import Project from SVN

Fields in import application are as follows

Repository URL

URL of the SVN project to be imported into HeliOS framework.

Other credential

It allows the user to enter the SVN credentials if it is different from the one used to sign in to HeliOS.

Username

The username used to login to the framework will be the default value in this field. This can be changed using the Other Credential option.

Password

The password used to login to the framework will be the default value in this field. This can be changed using the Other Credential option.

Head Revision

It imports the latest revision of a project from the version controlling system.

Revision

Revision is to import a project by providing the revision number. Previously checked in projects can be imported using revision number

GIT

Like SVN, GIT is a distributed revision control and source code management (SCM) system. Every GIT working directory is a full-fledged repository with complete history and full revision tracking capabilities. Import from GIT is used to import a replica of a project and make changes to it. Projects can be cloned from the repository by giving the repository URL.

Note

GIT_HOME environment variable should be set

E.g., GIT_HOME = C:\Program Files\Git;

path = %GIT_HOME %\bin;

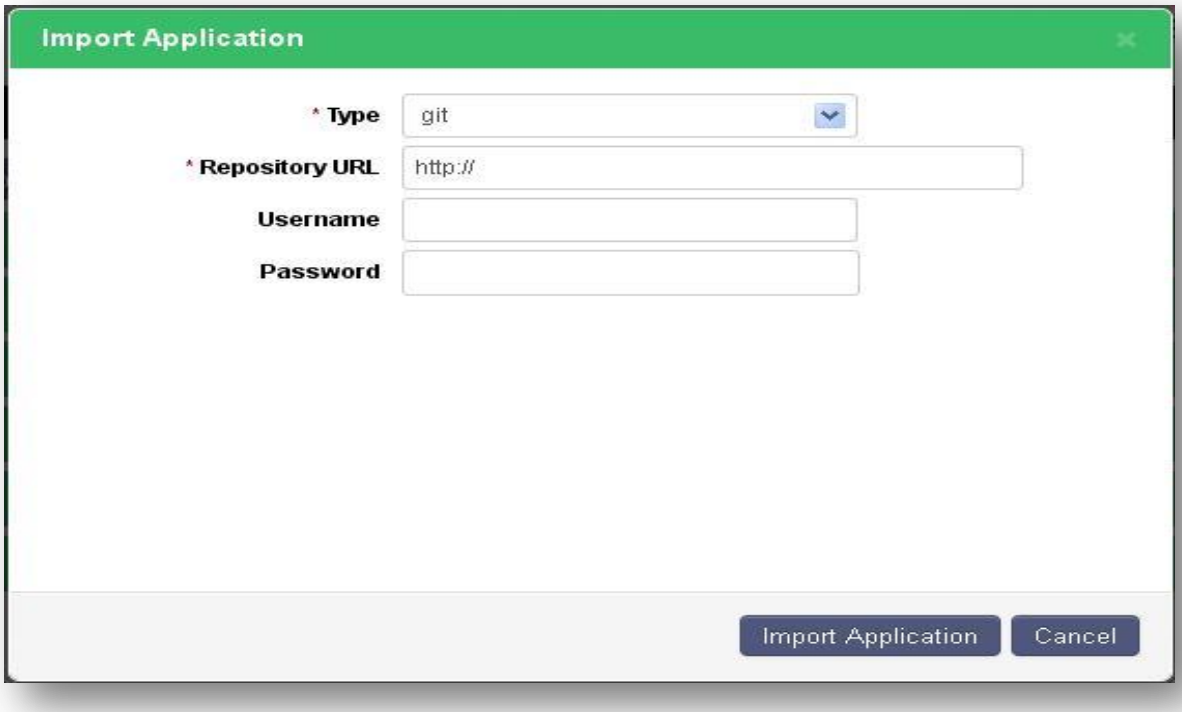

Figure 5-3: Import Project from GIT

Fields in import application are as follows

Repository URL

URL of the GIT repository to be imported into HeliOS framework.

Username

Username of the private repository for the private customers. This field is not mandatory and the public repository can be used by the customers.

Password

Password of the private repository for the private customers. This field is not mandatory and the public repository can be used by the customers.

5.1.4 Add to Repo & Commit to Repo

Using HeliOS, users can add their files to SVN repo and can also commit and update them. A detailed procedure for updating the applications is given in the next section. Unlike SVN, Git repo can only be updated.

Figure 5-4: Add, Update and Commit to Repo

5.1.5 Update Application in SVN and GIT

The applications imported using SVN and GIT can be updated by clicking the update icon available next to the application name in the Projects page. On clicking the update icon a pop up appears with two options.

SVN

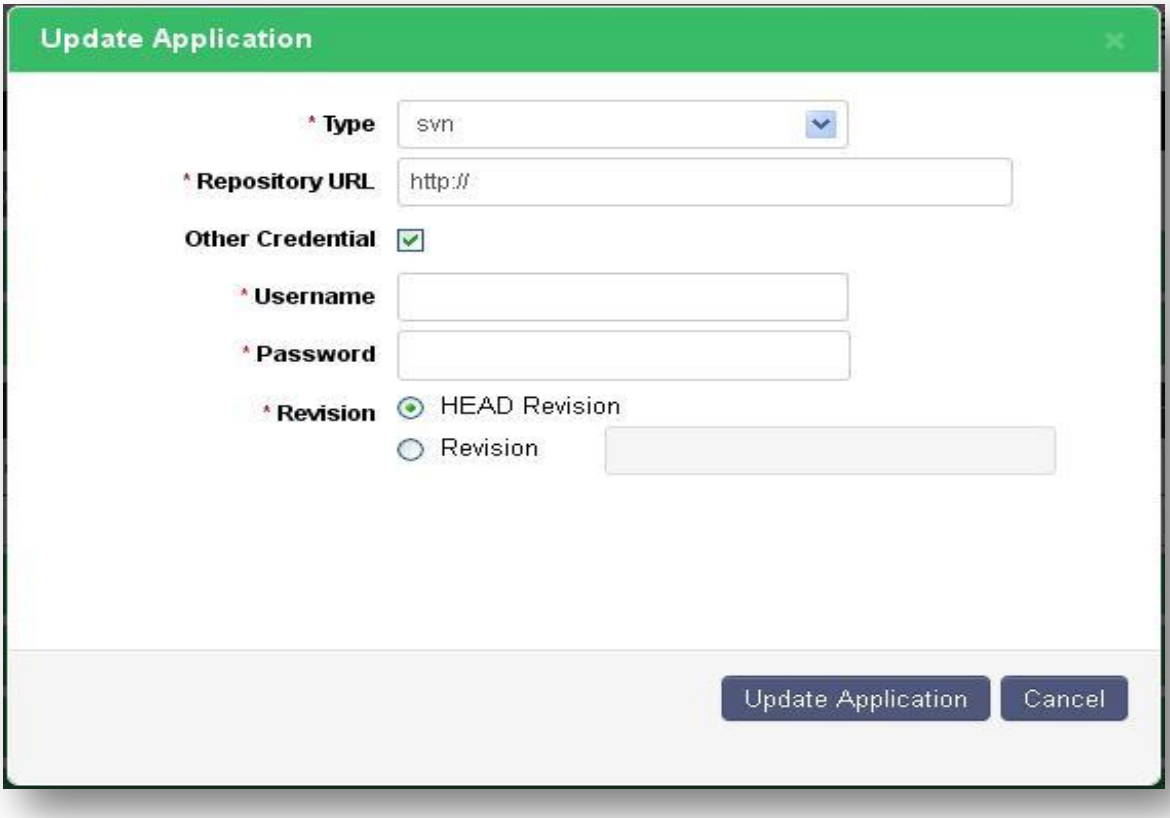

Figure 5-5: Update Application - SVN

Fields in the update application are as follows

Repository URL

URL of the SVN project to be imported into HeliOS framework.

Other credential

This field allows the user to enter the SVN credentials if it is different from the one used to sign in to HeliOS.

Username

The username used to login to the framework will be the default value in this field. This can be changed using the Other Credential option.

Password

The password used to login to the framework will be the default value in this field. This can be changed using the Other Credential option.

Head Revision

Head revision imports the latest revision of a project from the version controlling system.

Revision

Revision is to import a project by providing the revision number. Previously checked in projects can be imported using revision number

GIT

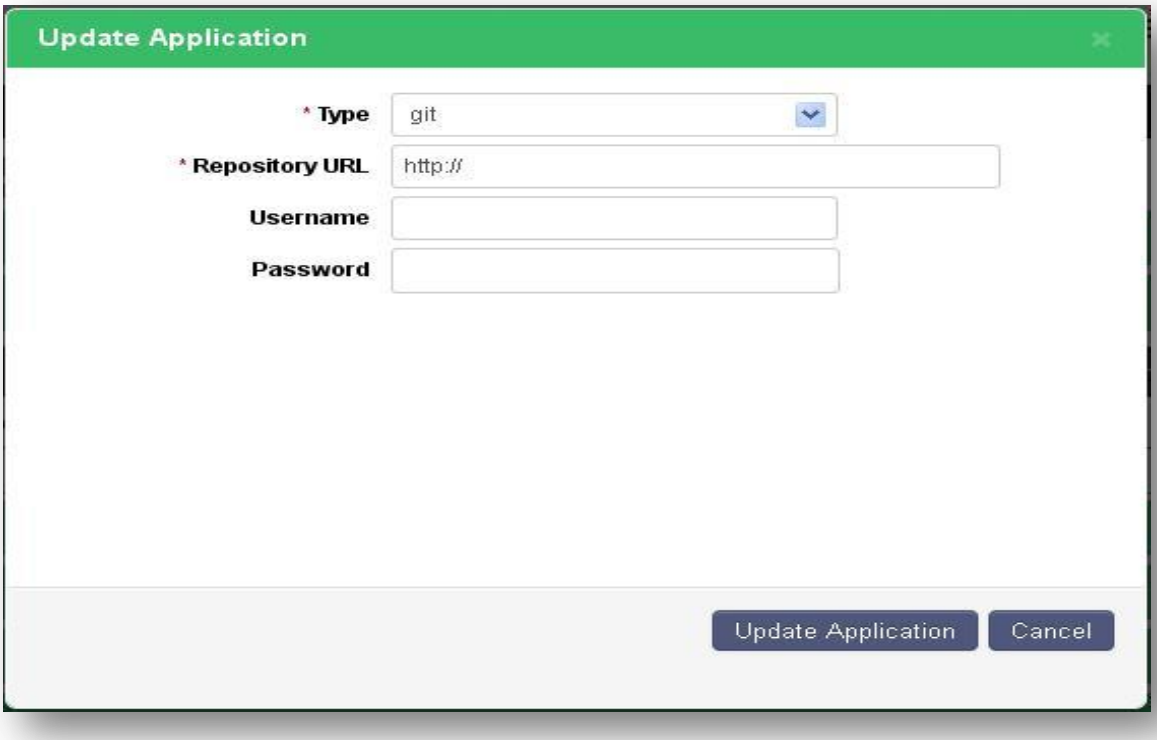

Figure 5-6: Update Application - GIT

Fields in the update application are as follows

Repository URL

URL of the GIT repository to be imported into HeliOS framework.

Username

Username of the private repository for the private customers. This field is not mandatory and the public repository can be used by the customers.

Password

Password of the private repository for the private customers. This field is not mandatory and the public repository can be used by the customers.

5.1.6 PDF Report

The PDF Reports for the Code Validation, Unit Test, Functional Test, Performance Test and Load Test can be viewed on clicking the PDF Report icon available next to the application name in the Projects page. On clicking the PDF Report icon a pop up appears.

The popup has an Report type dropdown in which the Over All report and Detailed Report values are available. The user can select either the Over All report and Detailed Report and click generate button, the PDF Report will be available.

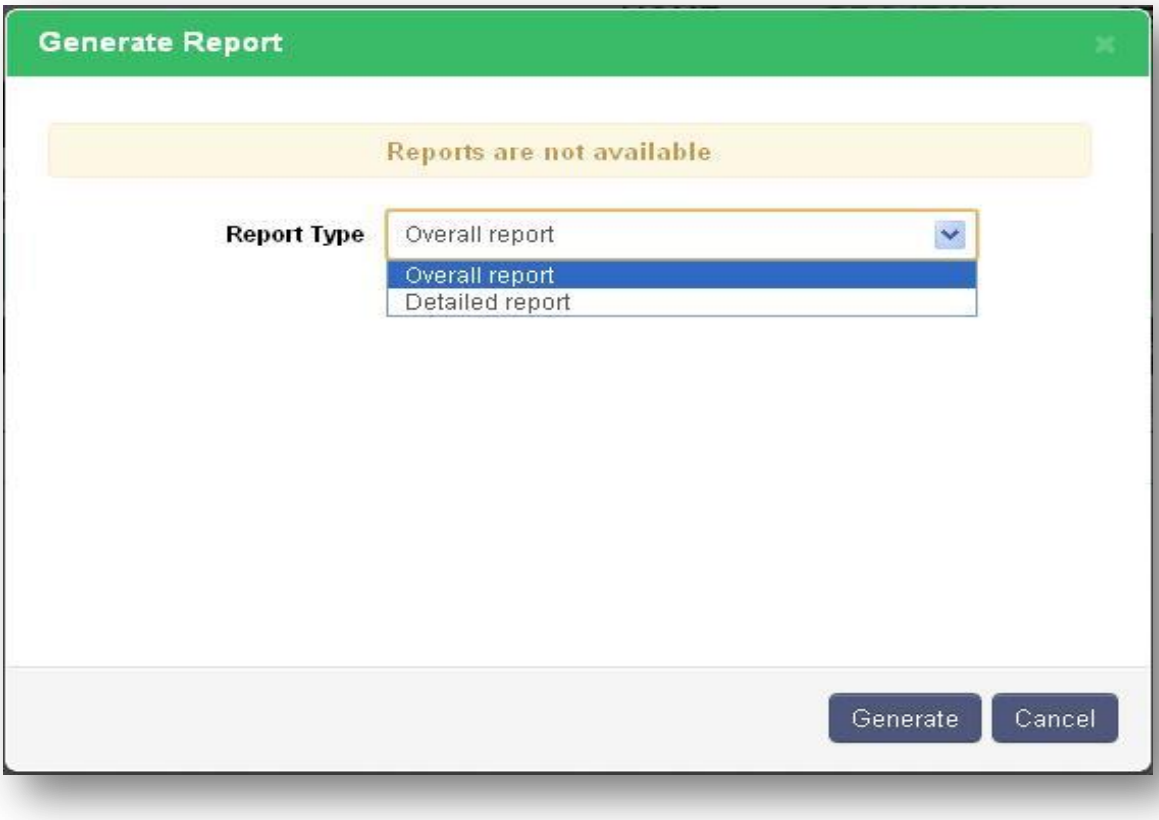

Figure 5.7: PDF Report Generation

Note

For viewing the PDF Reports any one of the below has to be executed

- Code Validation
- Unit Test
- Functional Test
- Performance Test
- Load Test

5.2 Edit Application

The project will be created after selecting the different layers and its components. It is not necessary to select all the three layers for a project. The layers are listed as different applications under one project. Each layer has its own configuration and lifecycle.

5.2.1 App Info

The App Info encompasses the information about an application. It holds the information about the Server, Database and the Web Service components utilized by a project.

Fields present in the App info are as follows

Name

It denotes the Name in which a project is created. Maximum text input of the field is restricted to 30 alphanumeric characters and is a mandatory field.

Code

It denotes the code in which a project is created. Maximum text input of the field is restricted to 12 alphanumeric characters and is not a mandatory field.

App Directory

This field is used to rename the application

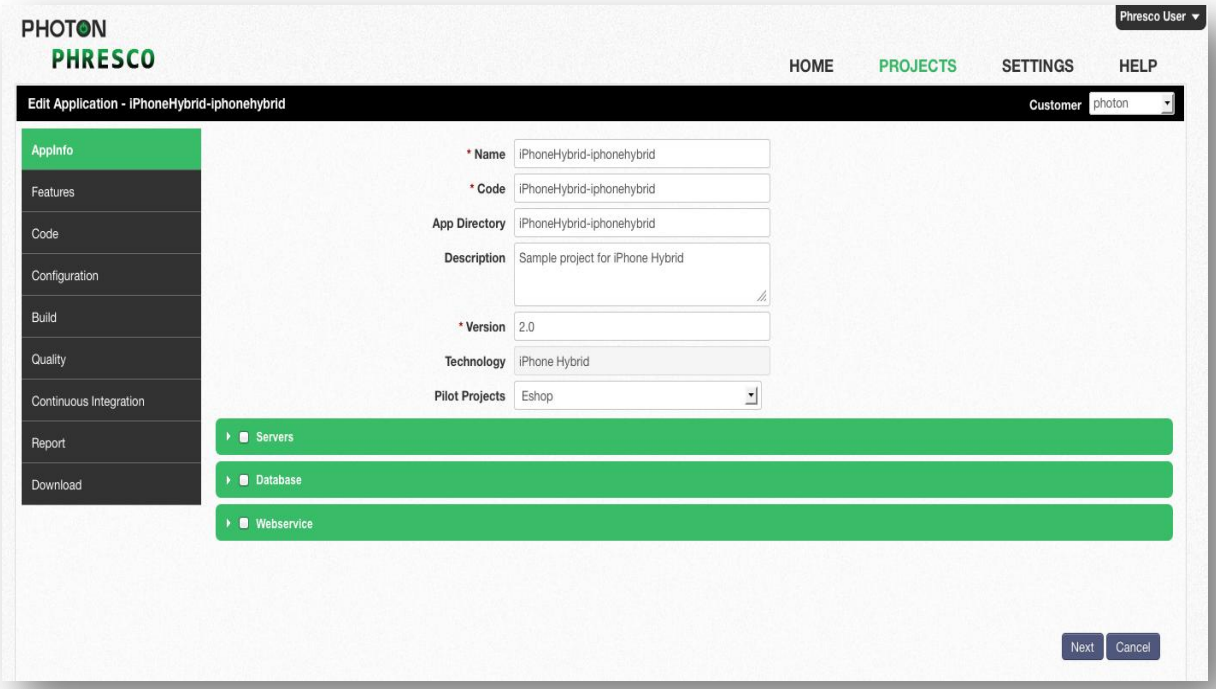

Figure 5-8: Edit Application

Description

It denotes the description of a project created. Up to 150 characters of the field can be entered and is not a mandatory field.

Version

It denotes the version of a project. 20 alphanumeric characters with a special character "." can be entered and is a mandatory field.

Technology (Archetype)

This field highlights the technology selected at the time of project creation. This field cannot be altered.

Pilot Project

HeliOS has included Pilot projects for most of the supported technologies. Pilot project is an actual project built with the best practices of project development, libraries, components and validated code structures. Any project can be made a pilot project provided it addresses the important criteria- being developer specific and tester specific.

When a new application is created, None project will be selected by default. This None project can be updated to E-Shop pilot project in the Edit Application. However, once the E-Shop pilot project is updated, it cannot be changed to None project once again.

The pilot project possesses libraries, components and validated code structures that can be modified based on the requirement. In the case of None project, all the components and libraries should be configured.

By publishing pilot projects in the repository server, HeliOS

- Allows components/libraries deployed to be re-used in multiple other projects.
- Authorizes re-branding of the pilot projects.

Advantages

- Less effort needed in creating new projects
- Ease in adopting the validated code structures and verified test cases ensure quality and saves time
- Improves application performance

Servers

Every organization has an affinity to adopt projects to their in-house servers, saving time and allaying adaptability concerns. Understanding this need, HeliOS has been configured in such a way that the project developed or adopted can run on a wide range of servers available in the market.

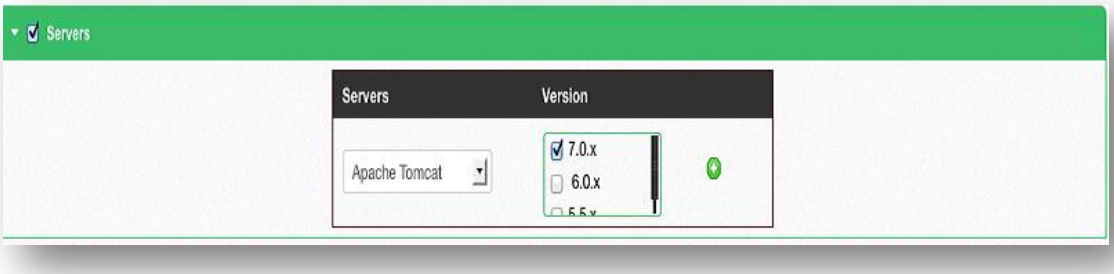

Figure 5.9: Server Selection in Edit Application Page

The desired server can be selected from the dropdown box in the accordion.

Supported Servers	Versions
Apache Tomcat	7.0.x, 6.0.x, 5.5.x
JBoss	7.1.x, 7.0.x, 6.1.x, 6.0.x, 5.1.x, 5.0.x, 4.2.x, 4.0.x
IIS	7.5, 7.0, 6.0, 5.1, 5.0
Web Logic	12c(12.1.1), 11gR(10.3.6), 11g(10.3.1), 10.3, 10.0, 9.2, 9.1
Apache	2.3, 2.2, 2.0, 1.3
NodeJS	0.6.x, 0.7.x, 0.8.x
Jetty	8.x, 7.x, 6.x, 5.x, 4.x
SharePoint Server	2007
MAMP	2.0.5
LAMP	0.0.9
WAMP	2.1e
XAMP	1.7.7
MicroApache	2.0.64
Apache Tomcat64	7.0.25

Databases

As with the servers, HeliOS incorporates popular databases that are widely preferred by organizations.

The desired database can be selected from the dropdown box in the accordion.

For the same project, HeliOS allows more than one database to be shortlisted. In the App info page, the developers can choose their preferred database from the list.

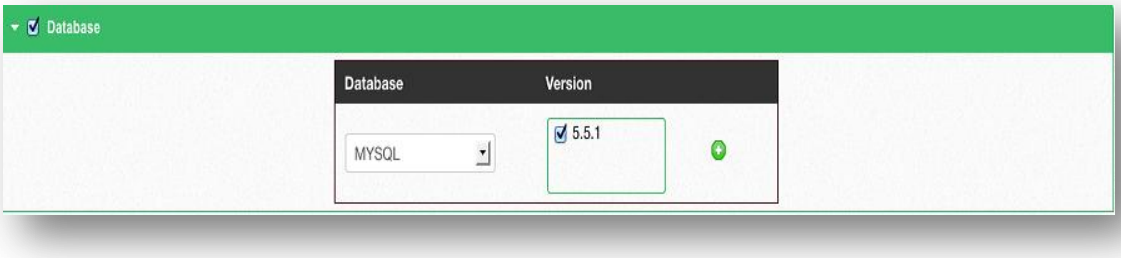

Figure 5-10: Database Selection in Edit Application Page

Supported Databases	Versions
MySQL	**5.5.1, 5.5, 5.1, 5.0, 4.1, 4.0**
Oracle	**11gR2, 11gR1, 10gR1, 10gR2 9iR2, 9iR1, 8iR3, 8iR2, 8iR1, 8i**
MongoDB	**2.0.4, 1.8.5**
DB2	**10, 9.7, 9.5, 9**
Microsoft SQL	**2012, 2008R2, 2008, 2005**

Note

Oracle database requires a dependency jar file. This jar should be used with a proper license and hence the users should manually add the dependency tag in the pom file which points the file path present in the local machine. Below is the example code snippet

```
<dependency>
   <groupId>com.oracle</groupId>
   <artifactId>ojdbc14</artifactId>
   <version>10.2.0.</version>
    <scope>system</scope>
    <systemPath>DRIVER NAME:/oracle/webdriver/ojdbc14--10.2.0.jar</systemPath>
</dependency>
```

Web Service

Some of the established Web Services are supported in HeliOS Framework. HeliOS allows more than one Web Service to be selected from the "App Info" page. From the shortlisted Web Services displayed in the App Info page, the desired option can be chosen.

Figure 5-11: Webservice Selection in Edit Application Page

The supported Web Services in HeliOS Framework includes,

- REST/JSON 1.0
- REST/XML 1.0
- SOAP 1.1
- SOAP 1.2

5.2.2 Features

HeliOS provides numerous Modules, JS libraries and Components to enhance a project. Every feature contains a feature icon and a brief description.

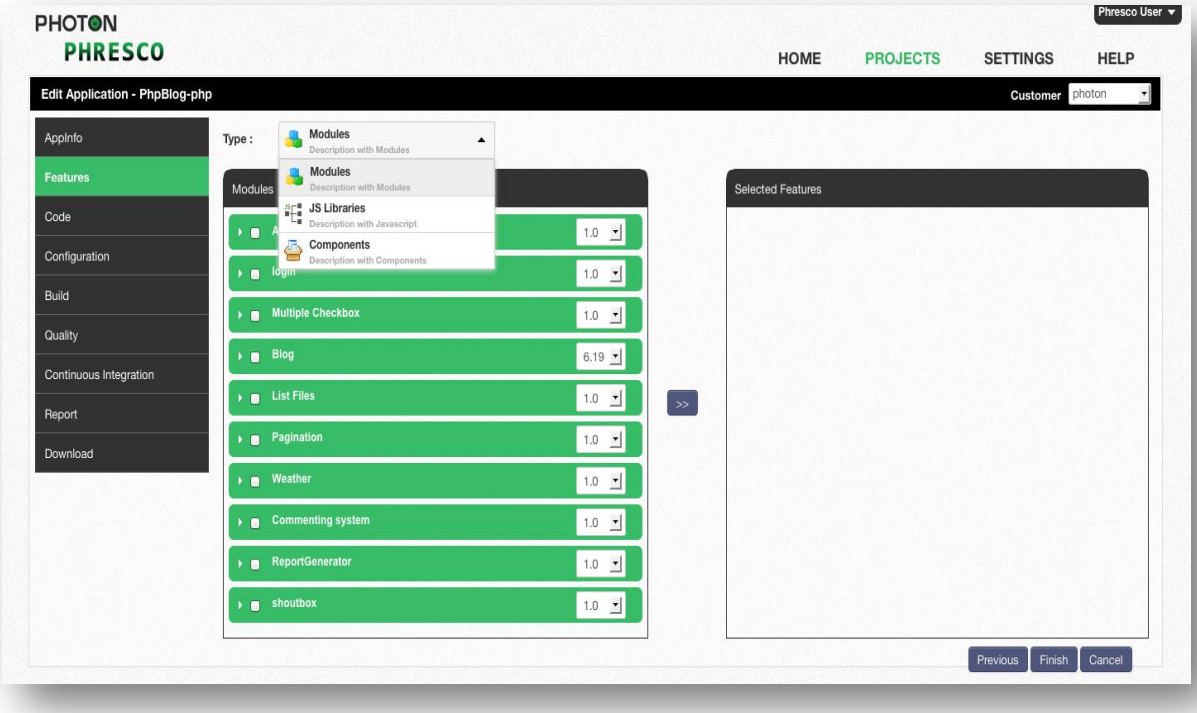

Figure 5-12: Feature Selection page

Modules

Modules include external features and custom features. External features are out of the box features offered by HeliOS to enhance a project. Custom features are customer specific features incorporated in HeliOS based on a request from the user.

JS Libraries

HeliOS offers JS Libraries as out of the box features to alleviate some of the difficulties faced in the development of Java Script based applications.

Components

These are prebuilt software packages that can be reused in multiple other applications.

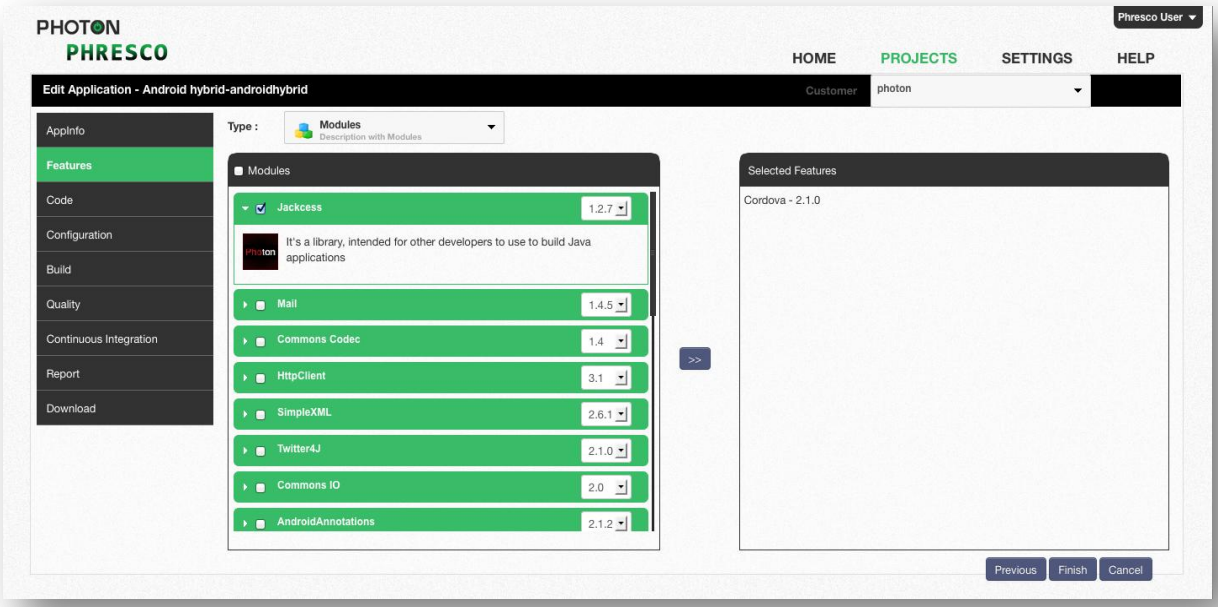

Figure 5-13: Sample Feature Explanation

5.2.3 Code Validation

Code validation is a method wherein the source code of a project is tested in compliance with the prescribed standards. HeliOS renders code validation by picking up standards for distinct technologies and examining the disparity with codes of a project. The errors are extricated depending on how the errors affect a project. Result is furnished in the form of graph.

HeliOS also supports code validation of test cases. Code validation can be done against source code and functional test cases for different technologies available in the validation pop up box. The technologies supported by HeliOS for code validation of source are Java, Java Script, HTML and JSP/JSF. This is available only for the widget projects.

For iPhone hybrid, blackberry hybrid and android hybrid technologies, code validation is executed against the targets or static HTML codes. Targets can be selected from the validation report drop down list available next to the Validate tab.

The code validation tool should have been started in order to perform Code validation. When HeliOS starts, the code validation tool also starts automatically.

The **Validate** button will be enabled once the code validation tool is started. Otherwise a warning message "Sonar not yet started" will be displayed.

Once the code is validated, validation reports for **Source** and **Functional test** are made available in the validation report dropdown box.

Use **Skip Unit test** to skip the unit test when performing code validation against source.

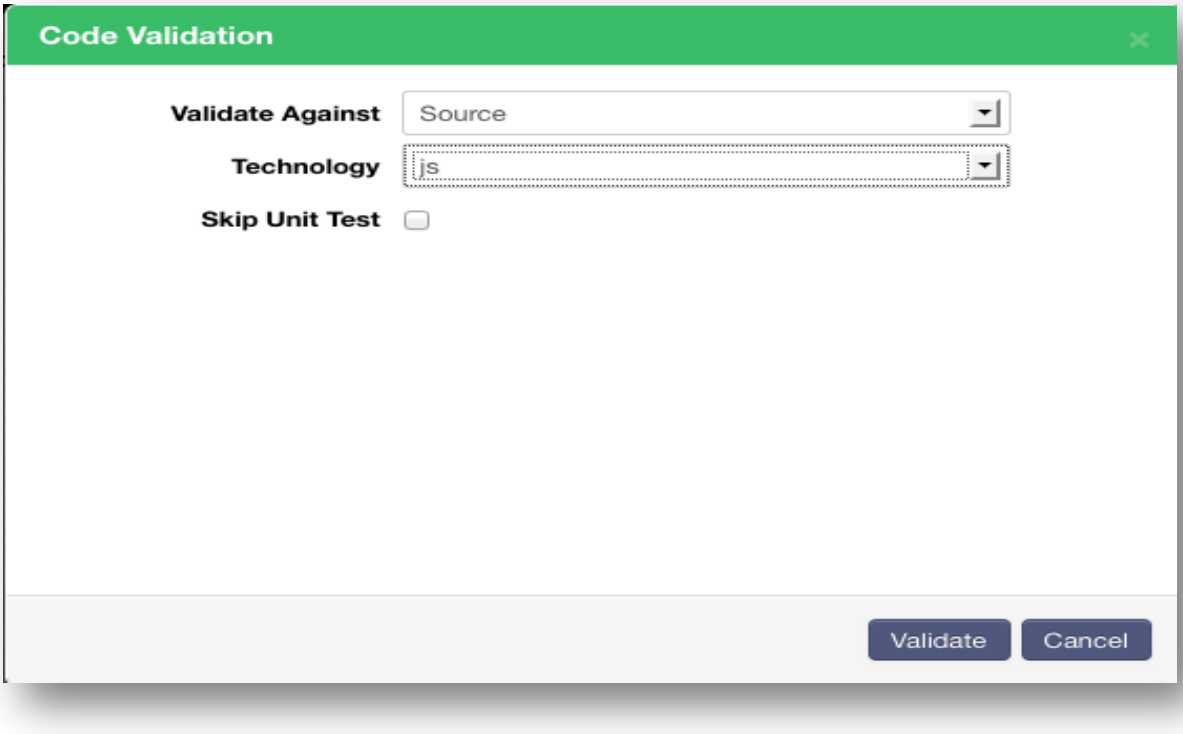

Figure 5-14: Code Validation Pop Up

Prerequisites for code validation

A prior installation of PEAR software is essential to run code validation for PHP, Drupal and WordPress projects.

- PHP Path should be set in environment variable
- Maven path should be set before the installation of Pear
- Pear has to be installed

To install PEAR in HeliOS

1. Set pear path- {pear installation path}/<APACHE>/bin/php in the environment variables for users
2. Set PHP path- {PHP installation path}/<APACHE>/bin/php/php 5.3.5 in the environment variables for system.
3. To set Maven path in the environment:

- Open a command prompt
- Navigate to HeliOS Framework-> bin
- Run the env.bat (windows)/env.sh (Mac and LINUX) for this will set Maven temporarily in the command prompt

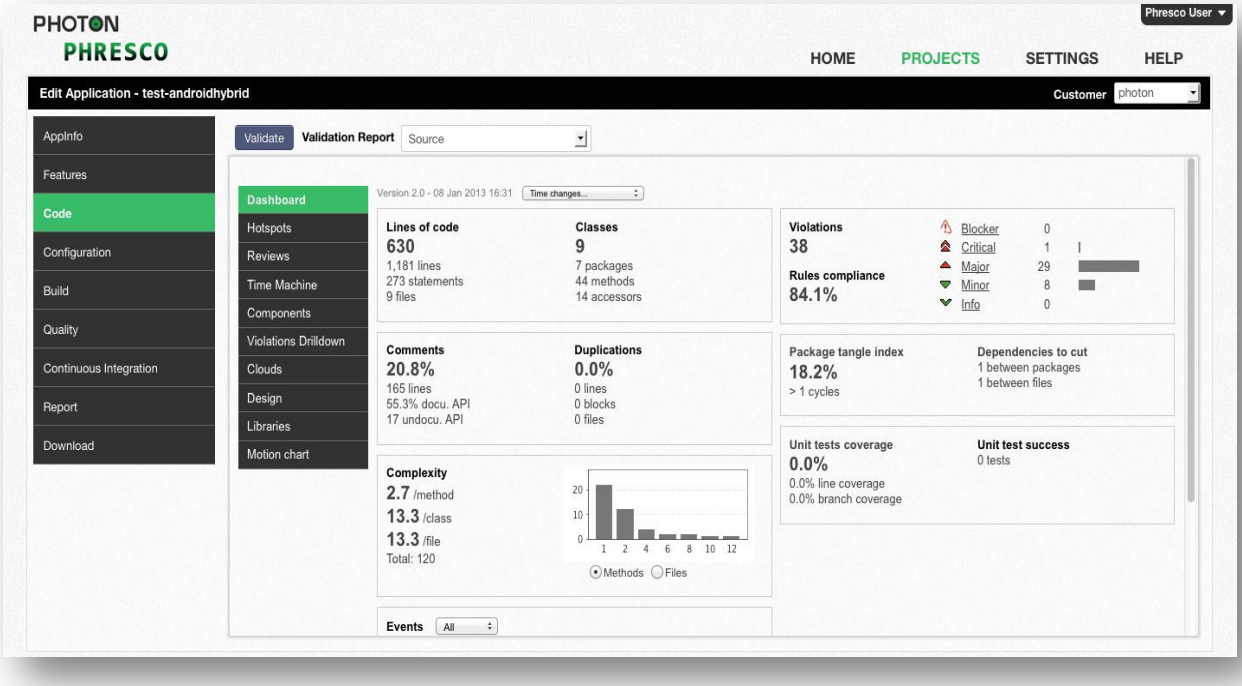

Figure 5-15: Code validation report

- Following this the steps for Pear installation should be carried over in the same command prompt

---

Note:

- This is carried over only when maven is not available.
- Setup for wamp and xamp is available in downloads tab of HeliOS.

---

4. Download <Php_Drupal_code_validation_setup> from HeliOS

5. Extract the zip file.
6. Edit Php_Drupal_code_validation_setup-> PearInstall-> setup.bat.
7. Configure
8. Php path should be set in setup.bat by opening it with edit tool. Below mentioned is the syntax to set the php path.
9. Call pear.bat <php path>
10. E.g.: call pear.bat <driver_name>:\<APACHE>\bin\php\php5.3.5
11. Execute setup.bat in Command Prompt / Terminal
12. On the execution of the command there are few steps which require User's input

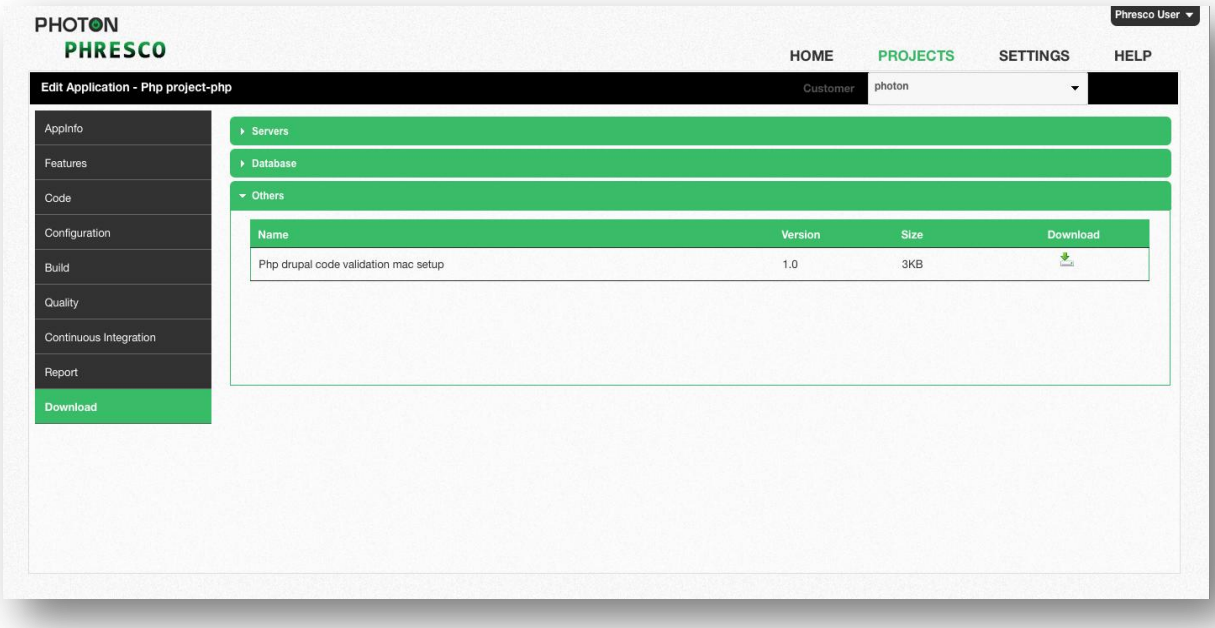

Figure 5-16: Downloading Php_Drupal_code_validation_setup for pear installation

Steps which requires user input

1. *Are you installing a system-wide PEAR or local copy?*
   Select an option or if it takes time to respond, a default value will be chosen by the system itself.
   The default value would be *<system:local> [system] :*
   Press Enter.

2. *1-12, 'all' or Enter to continue: _*
   Press enter.

3. *Would you like to alter php.ini <driver_name>:\<APACHE>\bin\php\php5.3.5\php.ini>? [Y/n] :*
   Type y and press enter.

4. *Press Enter to continue:*
   Press enter.

5. *Press any key to continue*
   Press enter.

6. *<driver_name>:\<APACHE>\bin\php\php 5.3.5>call pear upgrade pear*
   Wait till the system downloads the file.

7. *After downloading system pops up with a message box.*
   Are you sure you want to add the information in <driver_name>:\<APACHE>\bin\php\php5.3.5\PEAR_ENV.reg to the registry?
   Click yes.

8. *Click ok in the message box stating "Information in <driver_name>:\<APACHE>\bin\php\php5.3.5\PEAR_ENV.reg has been successfully entered into the registry"*

9. *After the installation of PEAR and when the build is success*
   Press any key to continue
   Press any key.

Note:

- When Pear is installed for the first time, the standards need not be downloaded.
- When Pear is already installed in the machine, Drupal standards should be downloaded for Drupal projects from download link and WordPress standards should be downloaded for WordPress projects (refer fig: 5.6)
- The standards should be pasted in the path,

For Windows,
"<DRIVER_NAME>\Apache \bin\php\php5.3.5\PEAR\PHP\CodeSniffer\Standards"

For Mac
"/usr/local/pear/share/pear/PHP/CodeSniffer/Standards"

5.3 Environment Configuration

Environment Configuration is the powerful feature of HeliOS for managing multiple environments in a project, where a single build can run on multiple environments without any configuration changes in build.

Environment set up in HeliOS user interface is made very simple and the developers can create any number of environments to deploy a single build.

The environment configurations with user credentials and other significant data are stored in Phresco-env-config.xml. Hence HeliOS encrypts this xml file for the technologies like PHP, Drupal and WordPress to hold intact data.

When the environment to be deployed is not selected, the application selects the default environment for the applications. The default environment can be switched over between any environments that are created using HeliOS Framework.

Note

- For the successful installation of Drupal the following changes has to be enabled.
- Extension of php_mcrypt in php.ini should be added with php_mcrypt.dll.

Steps for creating an environment

Step1 : Multiple environments can be created from the Edit Application page by clicking the **Application created -> Configuration -> Environments -> Add**.

Step2: When the add button is clicked after entering the fields, an environment is created.

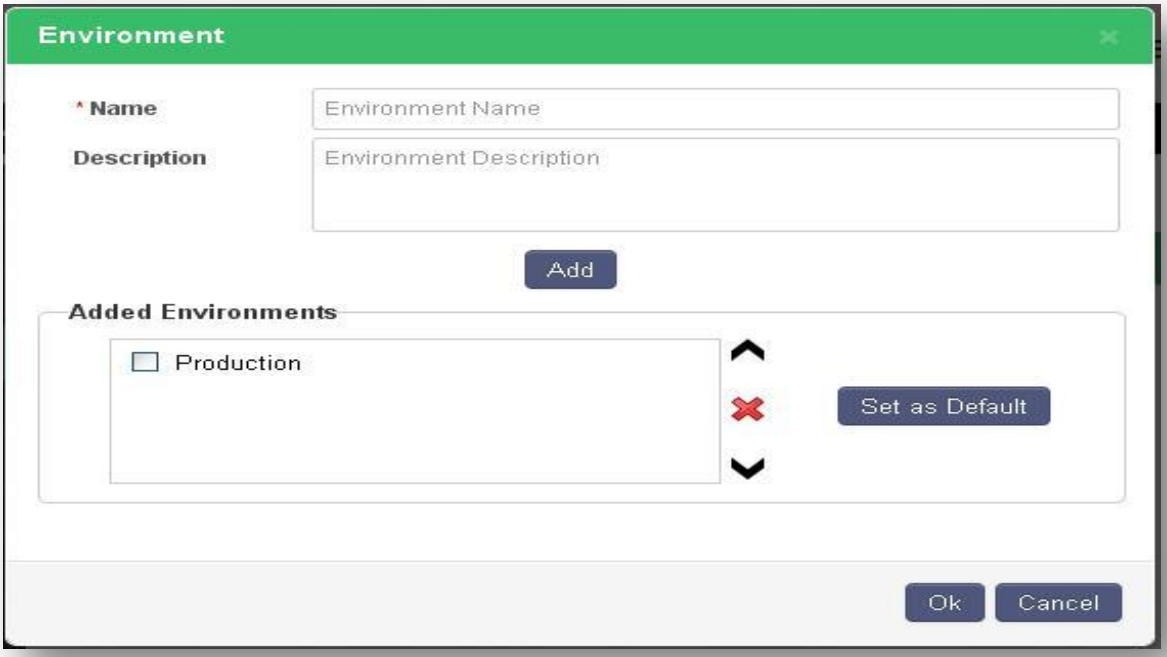

Figure 5-17: Environment Creation

Name

It denotes the Name in which a project is created. Maximum text input of the field is restricted to 30 alphanumeric characters and is a mandatory field.

Description

It denotes the Description in which a project is created. This field supports 150 alphanumeric, special characters and is not mandatory.

Added Environment

The added environment lists the environments created. Ordering of environment can be done by using the up and down arrow icons. Deletion of the environment can be done by using the delete icon.

Set as Default

Use this option to set a default environment

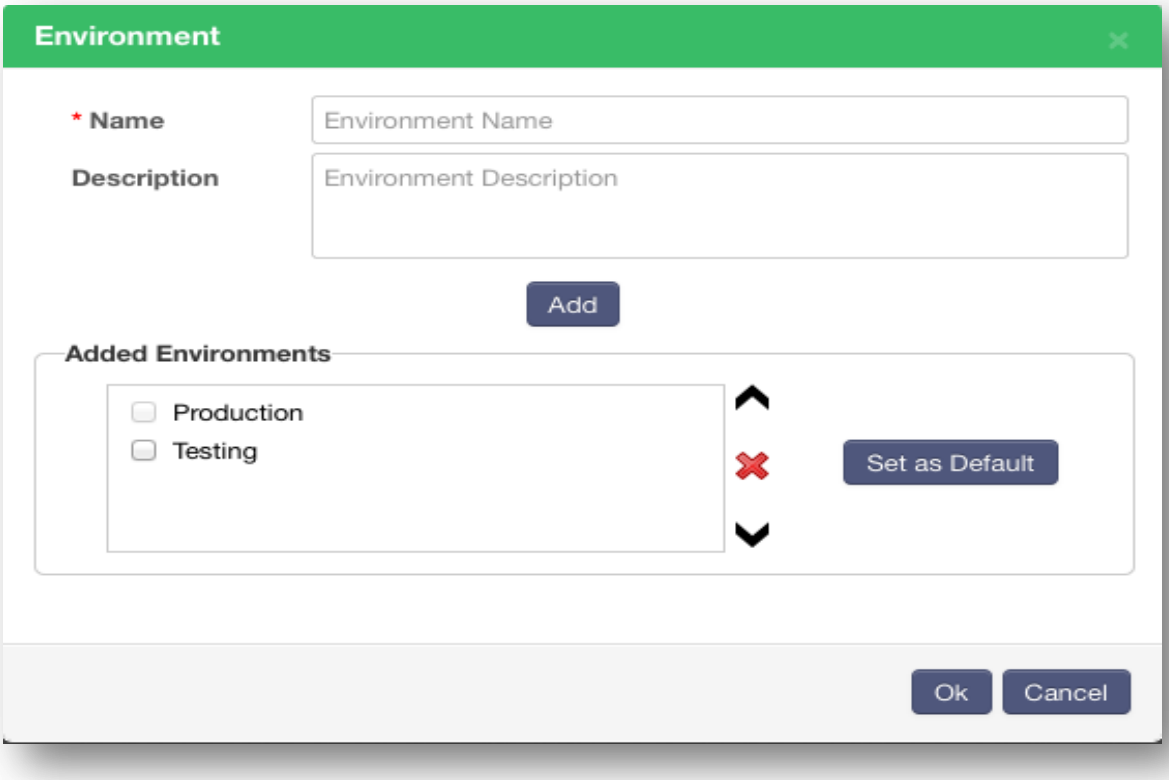

Figure 5-18: Environment Creation and Ordering

- Server, Database, Web Service, Email, SAP and LDAP can be configured in **Application created->Configuration->Add**. The created environment reflects in the environment field and the configurations can be associated under an environment by selecting the environment name from the drop down box

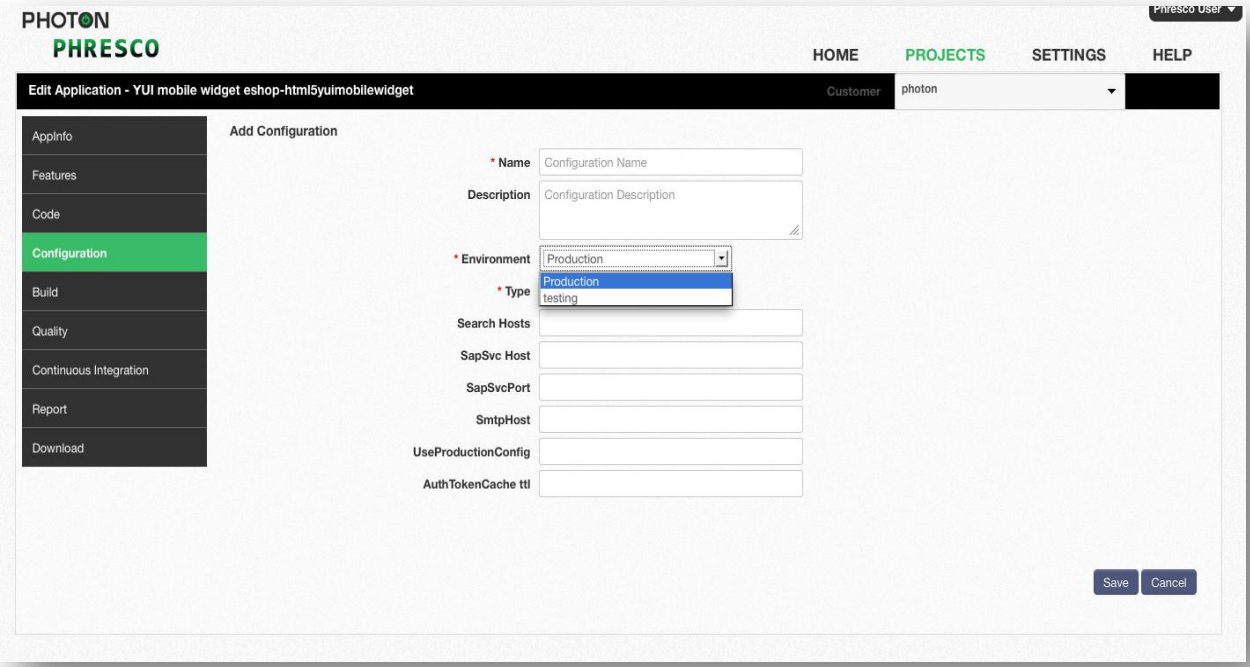

Figure 5-19: Environment Selection for a configuration

Note

- Multiple environments can be associated when building a project.
- Projects can be deployed only in one environment at a time.
- In web application only one environment can be selected at a time for functional, performance and load testing whereas for mobile application environments cannot be selected.

Advantages

HeliOS also allows user defined configuration template. New configuration templates can be published in HeliOS server which would be immediately seen in the drop down list of configuration page. For example log4j template can be published in HeliOS server which can be used across a project.

When the environment is created in the global settings, it can be used globally across projects using the show settings option.

5.3.1 Configuration Types

5.3.1.1 Server Configuration

The server configuration can be added from **Application created->Configuration-> Add**. The configuration type varies based on the configurations selected in the App Info page.

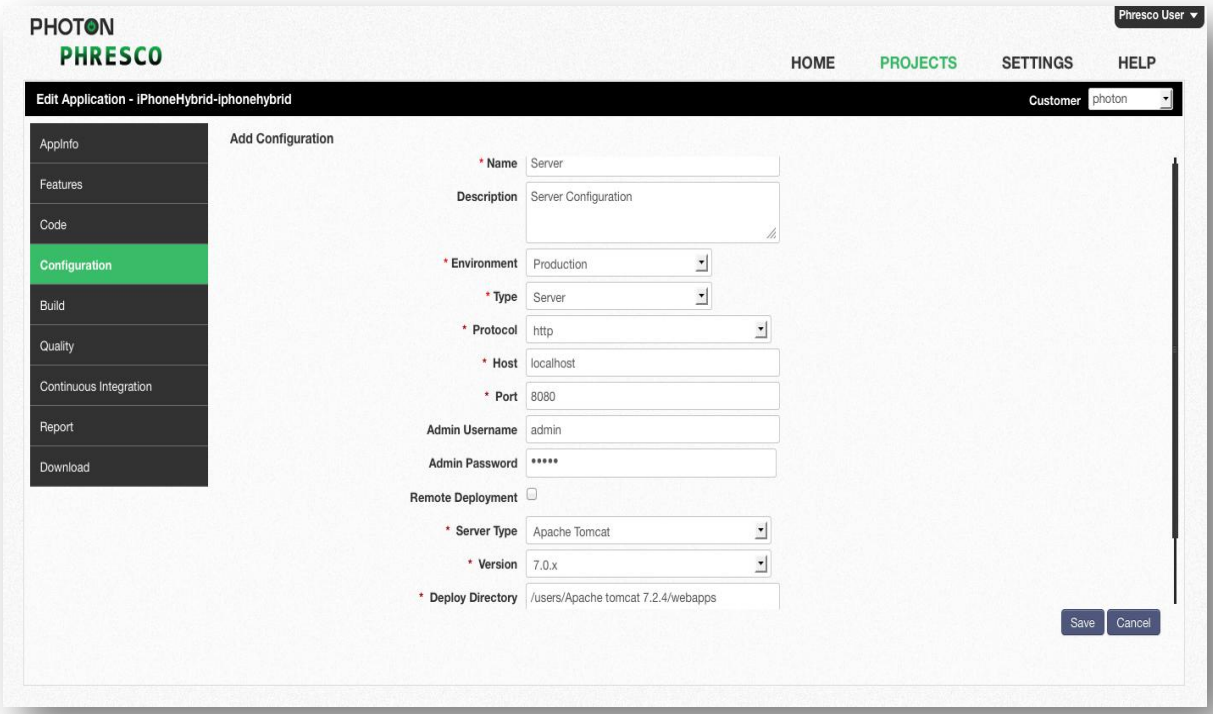

Figure 5-20: Server Configuration for an environment

Mentioned below are the details to be filled for server configuration:

Name

It denotes the Name in which a project is created. Maximum text input of the field is restricted to 30 alphanumeric characters and is a mandatory field.

Description

It denotes the Description of a project which is created. This field supports 150 alphanumeric, special characters and is not mandatory.

Environment

Created environments can be selected from the drop down box. If the environment is not created Production environment is selected as default.

Type

Configuration type can be selected from the drop down box. Server options should be selected for configuration.

Protocol

Protocol should be selected to access the application from dropdown box. http and https are the two types of protocols available. This field is mandatory.

Host

It denotes the Host in which a project is created and is mandatory.

Port

Port number of the server and it must be between 1 and 65535. This field is mandatory and only numeric characters are supported.
Port number should be selected such that the port does not run any other application which will not be known by the configuration.

Admin Username

It denotes the Admin Username of the server in which a project is created. This field supports alphanumeric, special characters and is not mandatory.

Admin Password

It denotes the Admin Password of the server in which a project is created. This field supports alphanumeric, special characters and is not mandatory.

Remote Deployment

The checkbox can be enabled to deploy a project in remote machine. Admin Username and Admin Password fields are mandatory for remote deployment.

Server Type

This contains the list of servers that was already short listed in the app info page. Only one server can be selected from the list.

Version

Version of the selected server will be displayed here.

Deploy Directory

Deploy directory of the server on the instance should be filled in this field. It is not a mandatory field and supports alphanumeric and special characters.

Context

Root context of the server can be specified here. It does not allow special characters. This is a mandatory field.

Additional Context Path

A project's additional context path can be specified here which can also include special characters.

For example http//localhost:2468/Phresco/login#. Here Phresco is the root context and login# is the additional context path.

Note

- By providing a port number for the server configuration and clicking on RunAgainstSource button for java projects, the inbuilt tomcat will reserve the next immediate port number for tomcat shutdown service.
- For example: If "7070" is provided as port number in server configuration and RunAgainstSource is clicked, "7071" port will be reserved. So configuring the port number "7071" in any other server configuration in a different project and clicking on RunAgainstSource will result in bind exception.

5.3.1.2 Database Configuration

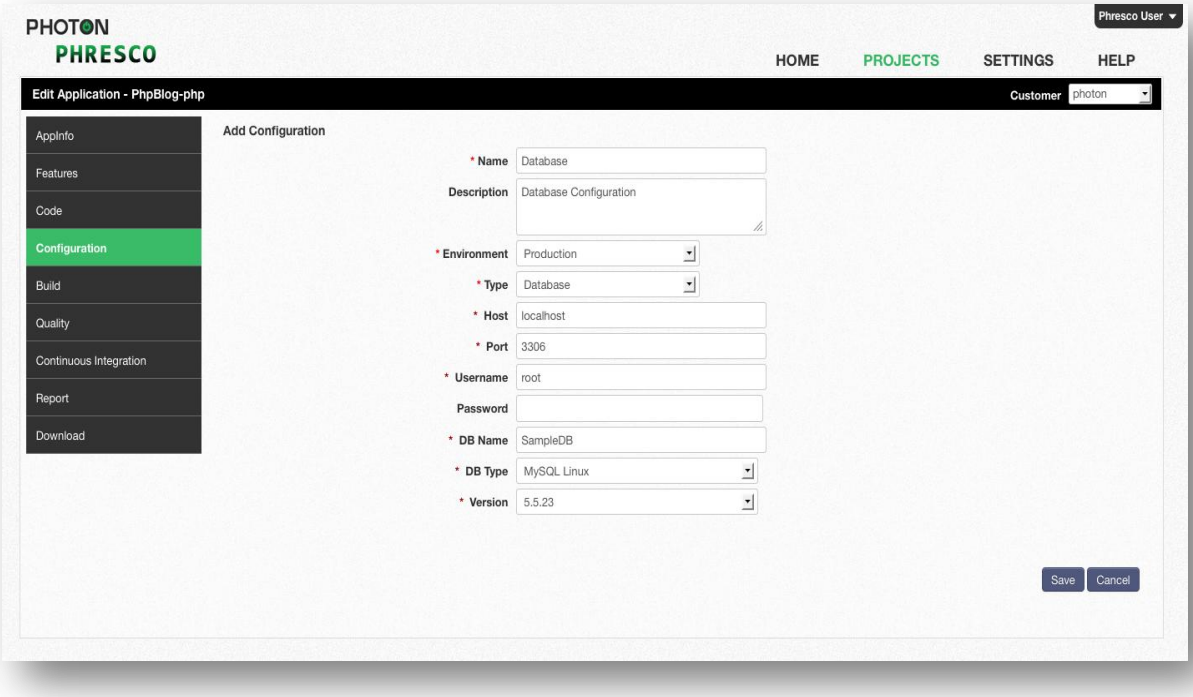

Figure 5-21: Database Configuration for an environment

Name

It denotes the Name in which a project is created. Maximum text input of the field is restricted to 30 alphanumeric characters and is a mandatory field.

Description

It denotes the Description in which a project is created. This field supports 150 alphanumeric, special characters and is not mandatory.

Environment

Environment created can be selected from the drop down box. If the environment is not created, production environment is selected as default and this field is mandatory.

Type

Configuration type can be selected from the drop down box. Database option should be selected for database configuration.

Host

It denotes the Host in which a project is created and is mandatory.

Port

It denotes port number of the database. Port number must be between 1 and 65535. This field is mandatory and only numeric characters are supported. Port number should be selected such that the port does not run any other application which will not be known by the configuration.

Username & Password

It denotes Username and Password to access the database. Username is “root”.

DB Name

Database name should be entered and this field supports alphanumeric and special characters.

DB Type

This contains the list of database that is already short listed in the app info page.

Version

Version of the selected database will be displayed here.

5.3.1.3 Web Service Configuration

Name

It denotes the Name in which a project is created. Maximum text input of the field is restricted to 30 alphanumeric characters and is a mandatory field.

Description

It denotes the Description in which a project is created. This field supports 150 alphanumeric, special characters and is not mandatory.

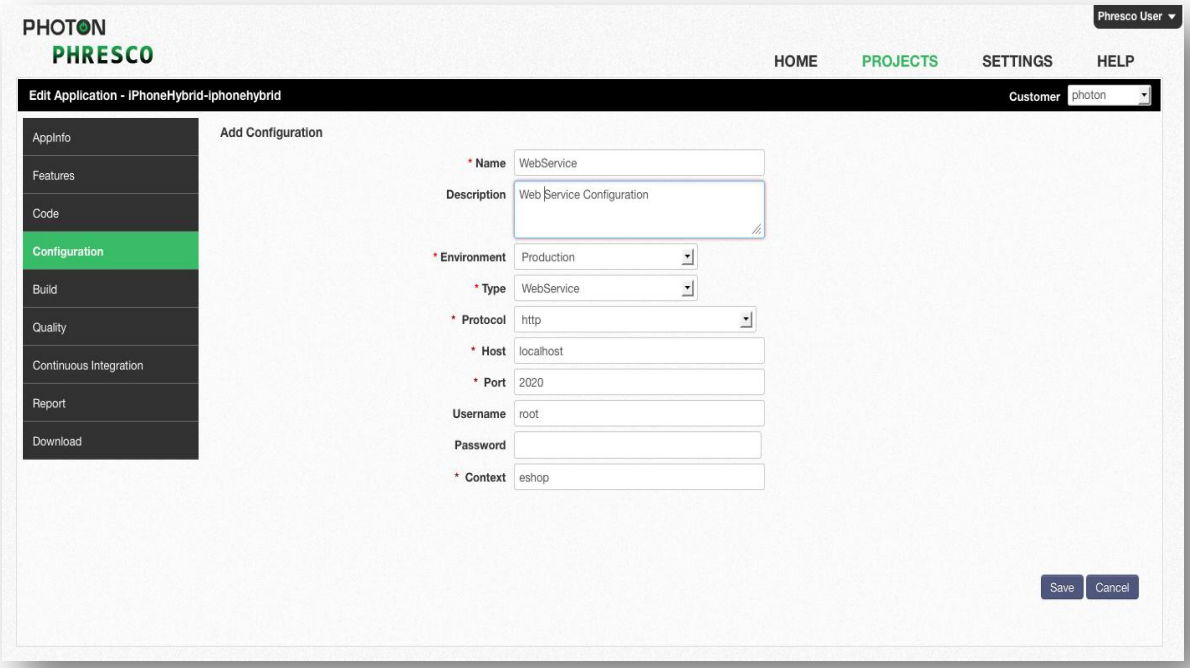

Figure 5-22: Webservice Configuration for an environment

Environment

Environment created can be selected from the drop down box. If the environment is not created Production environment is selected as default and this field is mandatory.

Type

Configuration type can be selected from the drop down box. Web Service option should be selected for Web Service configuration.

Protocol

Protocol should be selected to access the application from dropdown box. http and https are the two types of protocols available. This field is mandatory.

Host

It denotes the Host name of the web service in which a project is created and is mandatory.

Port

This field is the port number of Web Service. Port number must be between 1 and 65535. This field is mandatory and only numeric characters are supported.

Port number should be selected such that the port does not run any other application which will not be known by the configuration.

Username and Password

It denotes Username and Password to access the webservice.

Context

Context of the web service can be entered in this field and is mandatory.

5.3.1.4 Email Configuration

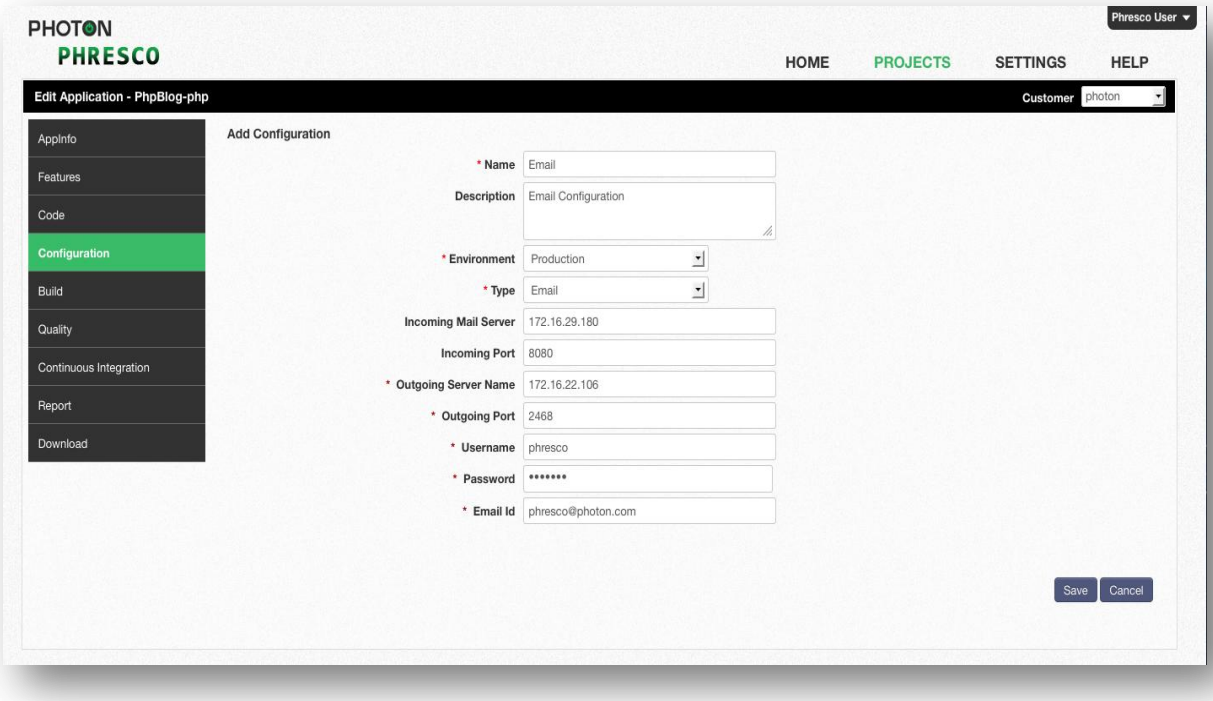

Figure 5-23: Email Configuration for an environment

Name

It denotes the Name in which a project is created. Maximum text input of the field is restricted to 30 alphanumeric characters and is a mandatory field.

Description

It denotes the Description in which a project is created. This field supports 150 alphanumeric, special characters and is not mandatory.

Environment

Environment created can be selected from the drop down box. If the environment is not created Production environment is selected as default and this field is mandatory.

Type

Configuration type can be selected from the drop down box. Email option should be selected for Email configuration.

Incoming Mail Server

The incoming mail server configuration can be entered and this field is not mandatory.

Incoming Port

Incoming mail server's port can be entered and this field is not mandatory.

Outgoing Server name

The outgoing mail server configuration can be entered and this field is mandatory.

Outgoing port

The outgoing mail server's port can be entered and this field is mandatory.

Username

It denotes username credential to access the mail server.

Password

It denotes the password credential to access the mail server.

Email Id

It denotes the email id in which the recipient receives the email notifications.

5.3.1.5 SAP Configurations

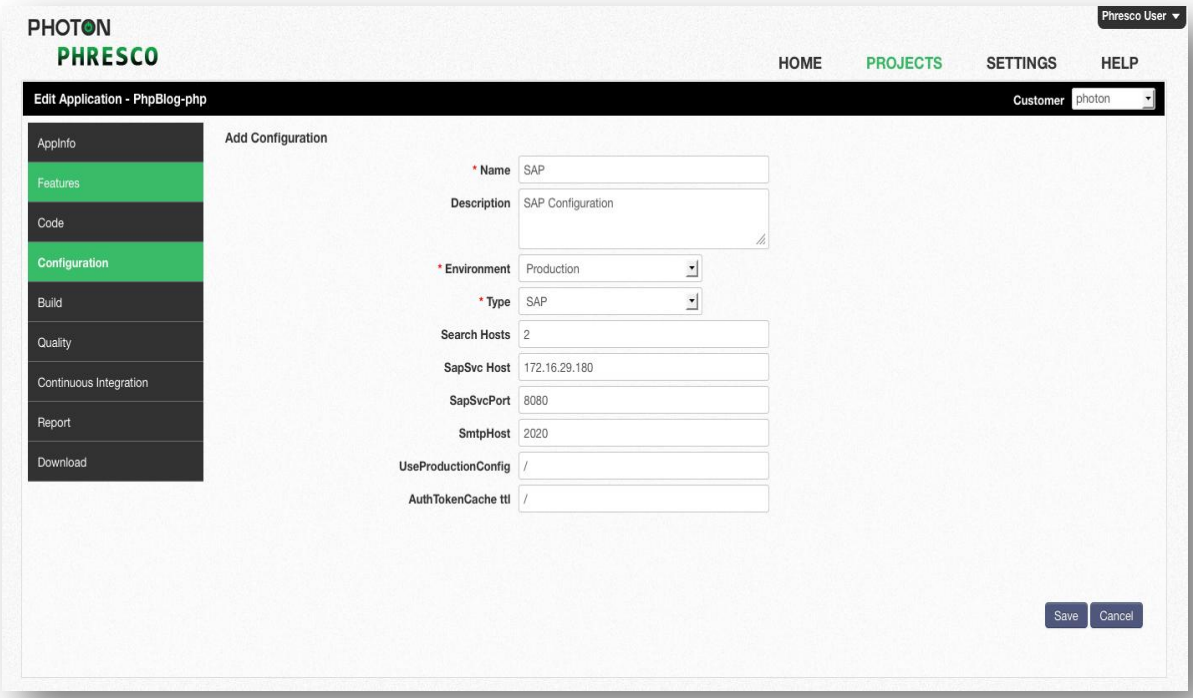

Figure 5-24: SAP Configuration for an environment

Name

It denotes the Name in which a project is created. Maximum text input of the field is restricted to 30 alphanumeric characters and is a mandatory field.

Description

It denotes the Description in which a project is created. This field supports 150 alphanumeric, special characters and is not mandatory.

Environment

Environment created can be selected from the drop down box. If the environment is not created Production environment is selected as default and this field is mandatory.

Type

Configuration type can be selected from the drop down box. SAP option should be selected for SAP configuration.

Search Hosts

This field is to search the host of the SAP Server. It supports alphanumeric, special characters and is not a mandatory field.

SapSvcHost

It denotes the Host name of the SAP Server in which a project is created and is not a mandatory field.

SapSvcPort

This field is the port number of SAP Server. This field supports alphanumeric, special characters and is not mandatory. Port number should be selected such that the port does not run any other application which will not be known by the configuration.

5.3.1.6 Dynamic configurations

Name

It denotes the Name in which a project is created. Maximum text input of the field is restricted to 30 alphanumeric characters and is a mandatory field.

Description

It denotes the Description in which a project is created. This field supports 150 alphanumeric, special characters and is not mandatory.

Environment

Environment created can be selected from the drop down box. If the environment is not created Production environment is selected as default and this field is mandatory.

Type

Configuration type can be selected from the drop down box. When the other option is clicked property key and property value field can be entered.

+ & -(Dynamic Key Values)

By clicking "+" users can enter both property key and property value of the field. Multiple entries can be added by clicking this and can be deleted by clicking '-'button

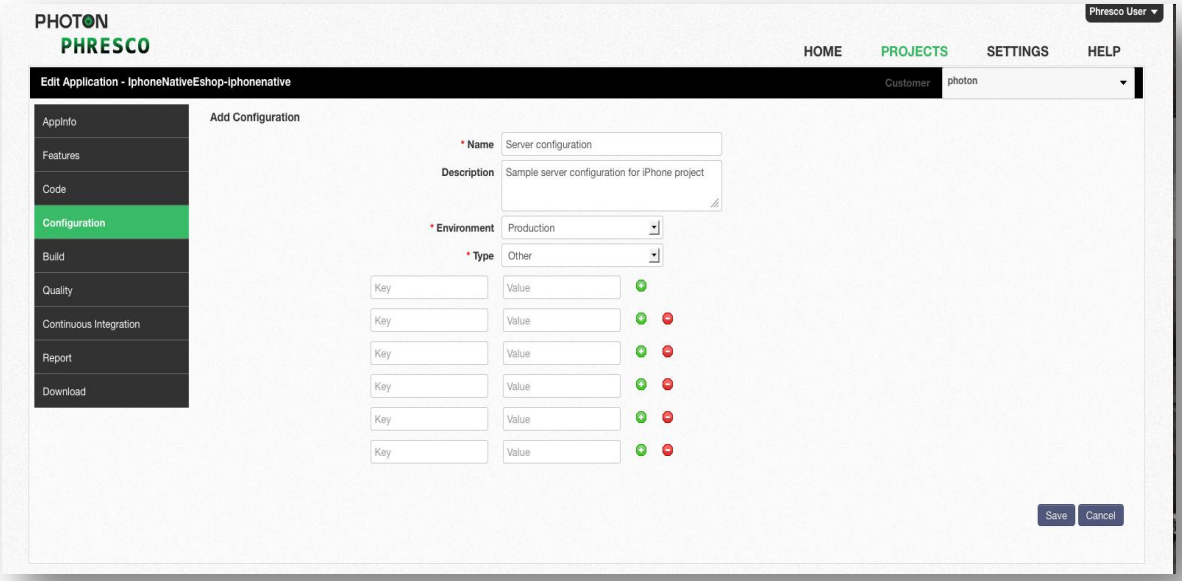

Figure 5-25: Dynamic Configurations for an environment

5.3.1.7 Cloning the Configurations

Apart from the advantages and various configuration types in HeliOS, users are also allowed to clone the configurations for different environments created. Instead of creating many server or database configurations for different environments, single server or database configuration can be cloned. This cloning of configurations is available for all the types of configuration including the dynamic configuration.

A configuration can be cloned from one environment to other by clicking the clone icon present next to the configuration that is created. A pop up appears to choose the environment for which the configuration is to be cloned.

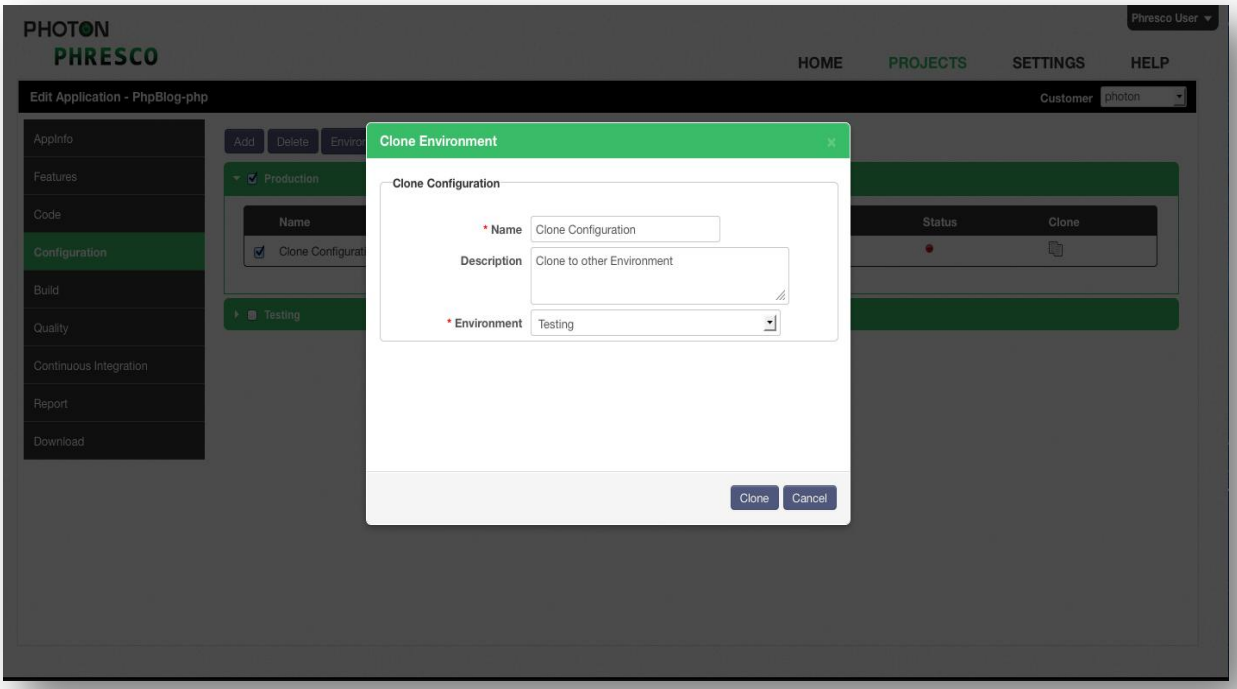

Figure 5-26: Cloning configuration

Fields in cloning configuration are as follows

Name

Name of the configuration to be cloned

Description

Description of the configuration to be cloned

Environment

The configurations are cloned to the environment that is selected from the dropdown box. The environments will be displayed in the dropdown box only after creating.

5.4 Global Settings

Global settings is the method of creating configurations for server, database, email and web service globally and use it for different projects which requires the same configurations. HeliOS also allows creating an environment and its configurations globally and reuse it for the other projects to make the work easier. Global settings can be configured by clicking **Settings->Add** for Server, Web Service, email and Database while for environment click **Settings->Environment**.

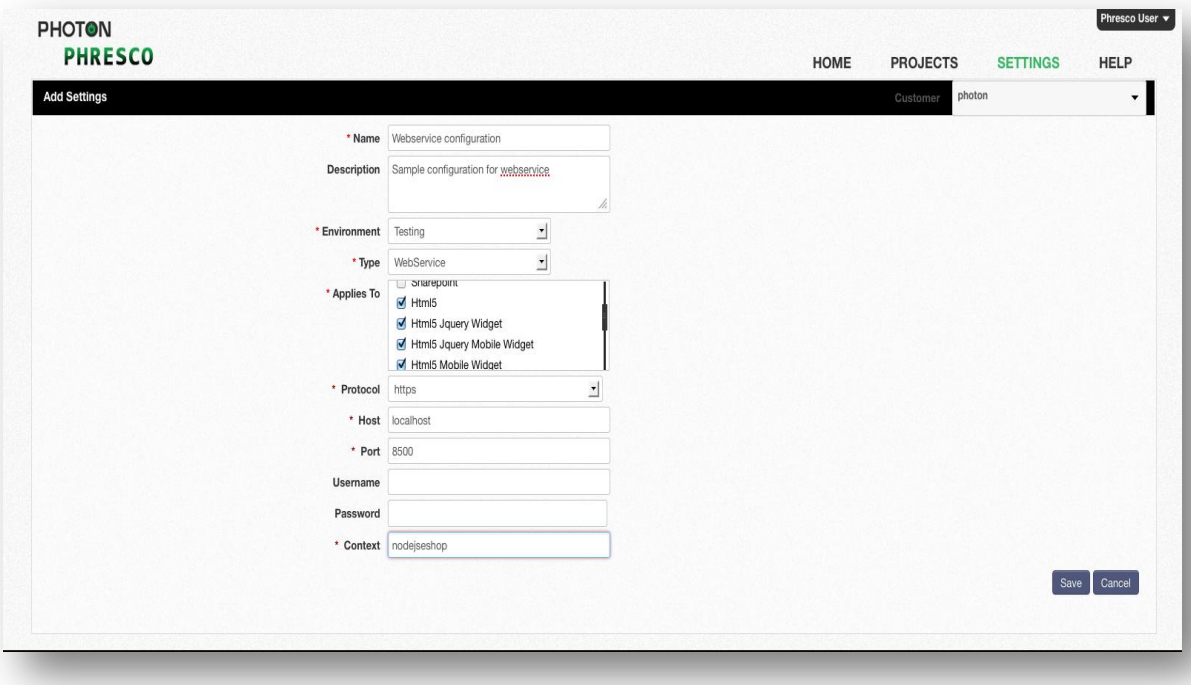

Figure 5-27: Configuration in Global Settings

5.5 Build Generation

The process of converting source code files into standalone software artifacts to run on a computer. The important process of the build generation is converting a source code into executable codes.

Project build process can be selected from **Applications->project created->Build->Generate Build**. A pop up box appears with environment selection, show settings, show error and Hide log. HeliOS allows user to name the build and also number the build that should be generated. The names or numbers can be provided by the users which may contain the version numbers or any other name related to the build.

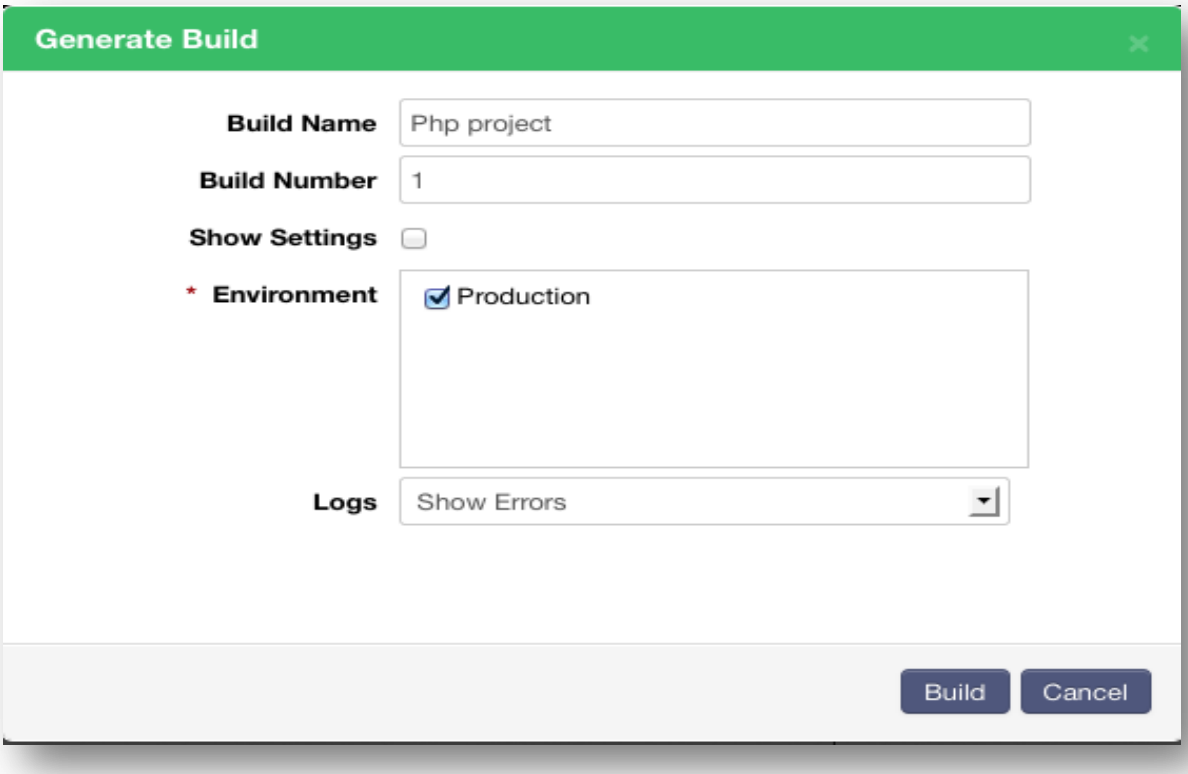

Figure 5-28: Generating Build

Fields in the Generate Build are as follows

Build Name

Name of the build to be generated

Build Number

Number of the build to be generated

Show Settings

The global configurations for server, database, web service and email can be selected using the show settings option.

Environment Selection

Multiple environments can be selected to build a project in the environment field and production environment will be already selected as default.

Logs

Selecting Show error in the logs will provide the error information along with the log for build generation.

Hide log hides the log console and provides only the error message while generating the build.

Skip Unit Test

HeliOS enables skip unit tests where the unit tests will be skipped on building a project. This consumes less time on building a project.

Note:

Generate build pop-up varies according for different technologies. HeliOS supports application level Build structure .i.e user can specify the target location for the build generation alongwith the required files that needs to be generated as a build.

5.6 Project Deployment

Project deployment is interpreted as a general process that should be customized according to the requirement and characteristic of a particular project which makes a project available for use. After the build generation process, a deploy icon appears on the screen. On clicking the icon, deploy pop up box appears which has environment selection, show error and hide log check boxes which has the same functionality as in build generation pop up box.

Deployment: Execute SQL

HeliOS provides an option to execute the SQL scripts to the configured database while deploying the application. To enable this feature, the Execute SQL checkbox should be selected in the deploy pop up.

- When the environment is selected in the environment field, the database chosen for that corresponding environment will be displayed in the database field.

- The site.sql files depending on the features selected in the features page will be displayed in the box on the left side as a list.
- Users can select the desired site.sql using the arrows present in the user interface and deploy the application.

Note

- The prerequisite for executing the SQL script is that, the configured database schema should be created in the database.
- In MongoDB, execute SQL will export BSON files instead of SQL files.

5.6.1 Remote Deployment

The application package can be deployed to a local machine or remote machine. HeliOS enables the remote deployment concept whenever there is a requirement to deploy to a remote staging or production machine.

Enabling Remote Deployment

Remote deployment in HeliOS can be done by selecting the remote deployment in the server configuration screen and by entering the host (IP address of the remote system), port (server port of the remote system), admin username (admin username of the server) and password (admin password of the server).

Application Servers supported in HeliOS for Remote Deployment

- Apache Tomcat – Above 6.x
- JBoss (Restricted to 4.x & 7.x)
- WebLogic (Restricted to 10.3.6, 12c)

Web Logic Server

For remote deployment using the Web Logic server, make the following changes in the Web Logic server configuration

Go to Domain Configurations in the Home page and select Domain -> Web Applications. Select the Archived Real Path Enabled check box and restart the server. This change is needed to launch pilot project in that server.

Apache Tomcat
Edit tomcat-users.xml in the path apache tomcat /conf/tomcat-users.xml and add the following

```
        <role rolename="manager"/>
        <role rolename="admin"/>
        <role rolename="manager-gui"/>
        <role rolename="manager-script"/>
        <user username="admin" password="admin" roles="manager-gui, manager-
        script, admin, manager"/>

<plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>tomcat-maven-plugin</artifactId>
        <version>1.1</version>
        <configuration>
        <username>admin</username>
        <password>admin</password>
        <url> URL </url>
        <path>/${project.build.finalName}</path>
        </configuration>
</plugin>
```

Note:

Server should be up and running during remote deployment.

5.7 Enabling https in HeliOS

http

HeliOS supports http (Hypertext Transfer Protocol) which is used for accessing resources normally. HeliOS runs in the port number "2468".

https

HeliOS additionally supports https ((Hypertext Transfer Protocol over Secure Socket Layer). This can be configured for the projects by using the following steps.

1. Edit server.xml in server/conf folder of tomcat installation.
2. Add the below details with preferred port, and keystore location

```
<Connector port="<PORT NUMBER>" SSLEnabled="true"
        maxThreads="150" scheme="https" secure="true"
        keystoreFile="<KEYSTORE FILE" keystorePass="<PASSWORD>"
        clientAuth="false" sslProtocol="TLS" />
```

3. Rerun the tomcat server and this will connect HeliOS in secure connection

Remote deployment through https is also possible in HeliOS only if valid certificate is selected from the User Interface. This is possible by clicking the Add certificate option after selecting the Remote deployment checkbox. The certificate from the local system can be browsed and attached.

5.8 Project Testing

Testing can be stated as the process of validating and verifying that a project

- Meets the requirement that guides its design and development;
- Works as expected; and
- Can be implemented with the same characteristics.

Many type of testing process can be done for a single project and it consumes more time. Using HeliOS framework circumvents the troubles associated with testing.

Report generated in PDF

Reports can be generated in the form of PDF which includes total report of all the test cases and also the code validation report when the report icon is clicked. The generated PDF can be sent to the stakeholders when there is a need.

Figure 5-29: PDF Report Generation

There are four types of testing process available and they are categorized into the following:

5.8.1 Unit Testing

Unit testing ensures that the developers write proper project implementation. For Unit testing, coverage and reporting HeliOS framework uses many tools like JUnit, NUnit, WSUnit, PHPUnit, NodeUnit, and OCUnit for different technologies. The results are provided both in a tabular and graphical output format indicating the ratio of success, failure or error among the test cases.

The unit test can be performed by clicking the **Test** button available in the **Unit** tab.

5.8.1.1 Unit testing for iPhone

In HeliOS, execution of Unit Test differs for iPhone applications.On clicking **Test** button in **Unit** Tab, the below pop-up appears.

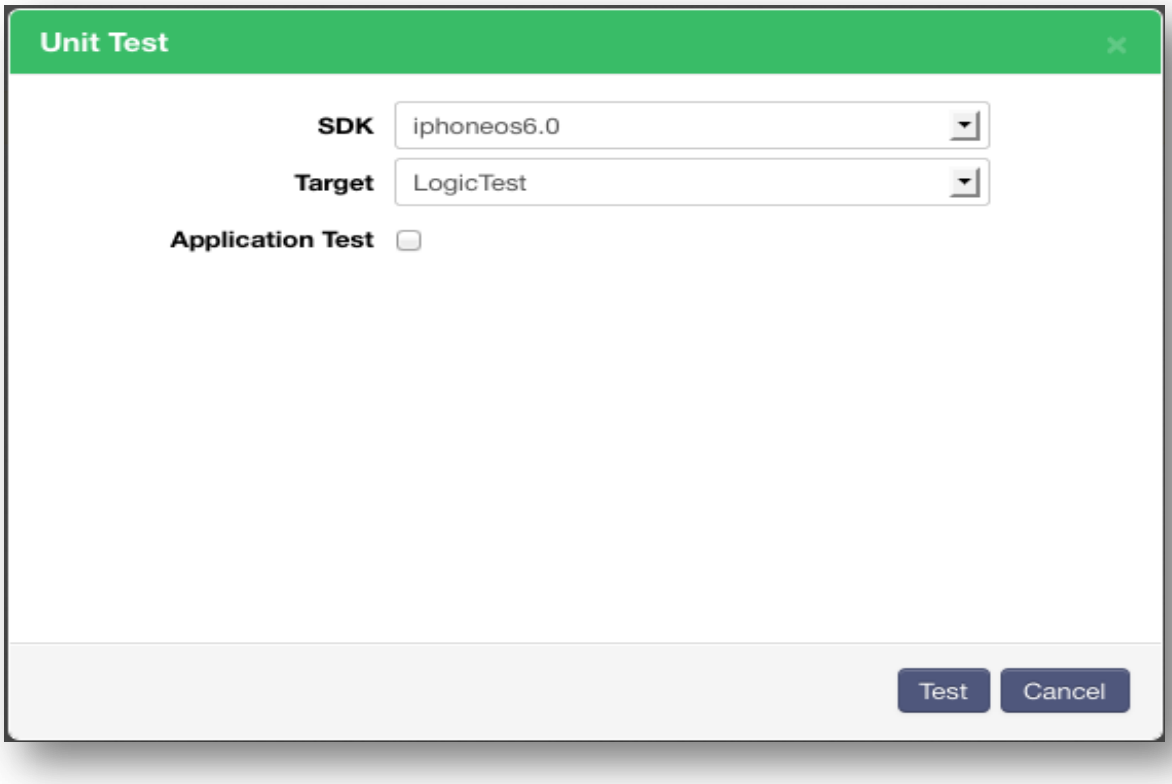

Figure 5-30:Unit Test for iPhone Application

SDK

SDK versions installed in the system will be displayed in this field and the SDK in which tests needs to be executed can be selected from this drop down box.

Target

Targets will be listed in this drop down user can select the target to be executed from this drop down box.

Application Test

Application Test can be performed by selecting the Application Test in the Target drop down and by checking the Application Test check box.

Note:

- Refer section 8.8.1 for detailed information on Logic and Application Tests for iPhone applications
- A PDF report of the executed tests can also be generated using "Report" icon as shown 

5.8.2 Functional Testing

Functional testing involves testing on the specifications of a project under test. Functions are tested by feeding them input and examining the output.

Functional testing involves

- Identifying functions the software is expected to perform.
- Creating input data based on the function's specifications.
- Determining output based on the function's specifications.
- Executing test cases.
- Comparing actual outputs and expected outputs.

Functional testing uses tools like selenium, Robotium, Monkey Talk and Xcode. The result is given through static analysis and test cases.

5.8.2.1 Functional Test execution for Widgets, PHP, Nodejs & Java Webservices

The functional test cases can run in different browsers parallelly in different machines that are supported by HeliOS using the Selenium Grid.

Though HeliOS supports Selenium Grid by default, Selenium Webdriver testcases can also be executed for the above mentioned technologies.
Changes that has to be made for Selenium Webdriver Support:

- Go to directory D:\Phresco-Build<version>\phresco-framework\workspace\projects\Your_Project
- Open the pom.xml and replace the value grid to webdriver as shown:

<phresco.functionalTest.selenium.tool>grid</phresco.functionalTest.selenium.tool>

Note:

For Widgets, HeliOS supports integration and execution of automation testscripts using Cucumber. Refer Section 5.8.4 for detailed explanation on the Functional test execution pop-up for Selenium Webdriver.

5.8.2.2 Functional Test execution in Selenium Grid

Click on the created application in "**Project**" page and navigate to Quality tab by clicking the **Application created →Quality→Functional**

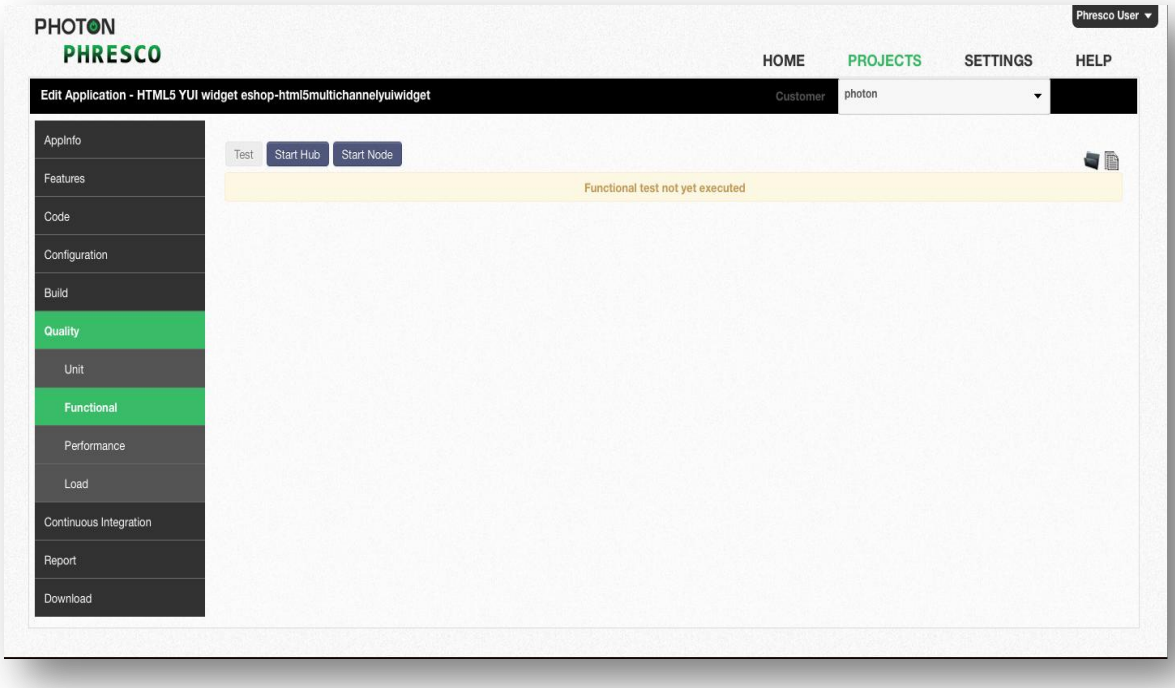

Figure 5-31: Executing Functional Test

Start Hub

On clicking the **Start Hub** button, the following popup appears:

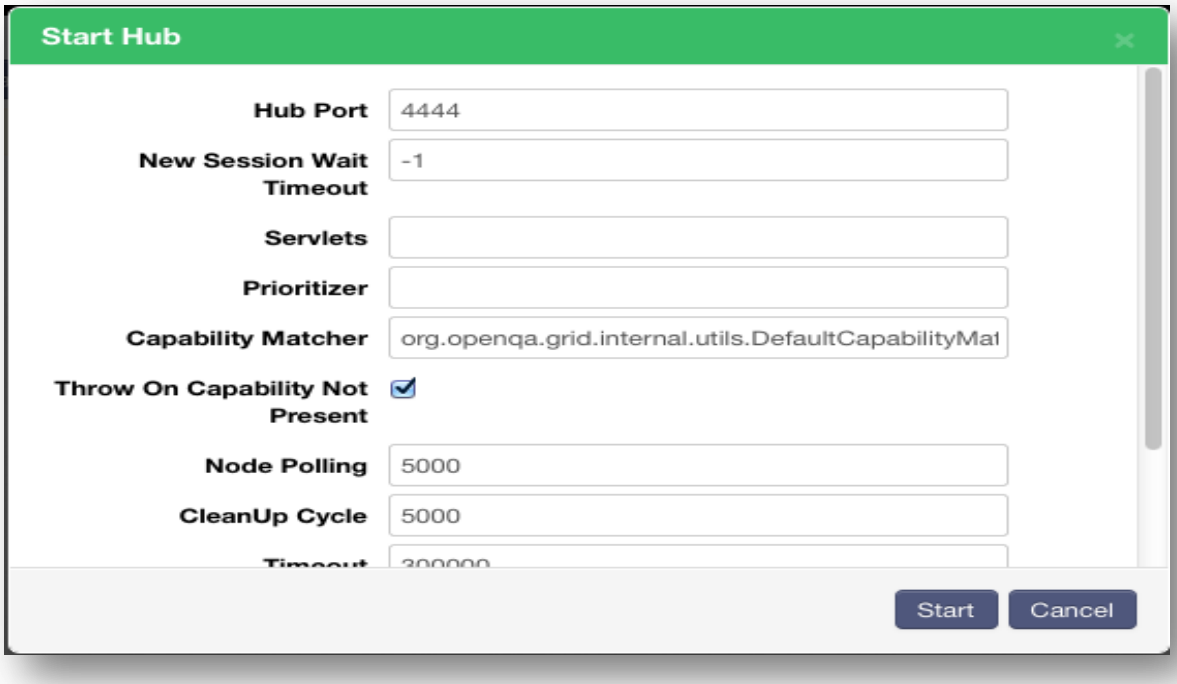

Figure 5-32: Start Hub

Hub Port

This field is to configure the listening port of the hub. Default port is 4444.

New Session Wait Timeout

Timeout value required for the test to be executed in node has to be mentioned here in milliseconds.This is not mandatory, however (-1 ms) is set as default in HeliOS. When the timeout value for a specific test case exceeds the value as mentioned in "**New Session Wait Timeout**", the test will throw an exception before starting a browser.

Servlets

This is to register a new servlet on the hub/node.

Prioritizer

This is a class implementing the Prioritizer interface. It is set to null by default (no priority = FIFO).For example, User needs to specify a custom prioritizer if they need the grid to process the tests from CI, or IE tests first.

Capability Matcher

This is a class implementing the CapabilityMatcher interface. Logic that the hub follows to assign requests to the node has to be specified here. This class can be changed if the user wants to have the matching process.For e.g Regular expression can be used instead of exact match for the version of the browser. By default, it is set to org.openqa.grid.internal.utils.DefaultCapabilityMatcher. All the nodes of a grid instance will use the same matcher, defined by the registry.

Throw On Capability Not Present

Can be set to either <true | false> and by default it is set to true. If true, the hub will reject the test requests right away when no proxy that can host that capability is currently registered. User can set it to false to have the request queued until a node supporting the capability is added to the grid.

Node Polling

It denotes how often the hub checks if the node is still alive.

CleanUp Cycle

It denotes how often a proxy will check for timed out thread in milliseconds.

Timeout

This refers to the timeout in seconds before the hub automatically ends a test that hasn't had any activity in the last X seconds.

After which the browser will be released for another test to use. This typically takes care of the client crashes.

Browser Timeout

Timeout value in seconds for which a browser will hang has to be mentioned here.

Max Session

This refers to the max number of tests that can run at the same time on the node, independently of the browser used. The Hub starts on clicking the "**Start**" button after filling all the required details in the Start Hub popup.

Start Node

On clicking the "**Start Node**" button the following popup appears

Hub Host

This denotes the <IP | hostname>, usually not needed since it is determined automatically. For exotic network configuration, network with VPN, specifying the host might be necessary.

Hub Port

This field is to configure the listening port of the hub. Default port is 4444.

Node Port

This field is to configure the listening port of the Node. Default port is 5555.

Max Session

This refers to the max number of tests that can run at the same time on the node, independently of the browser used.

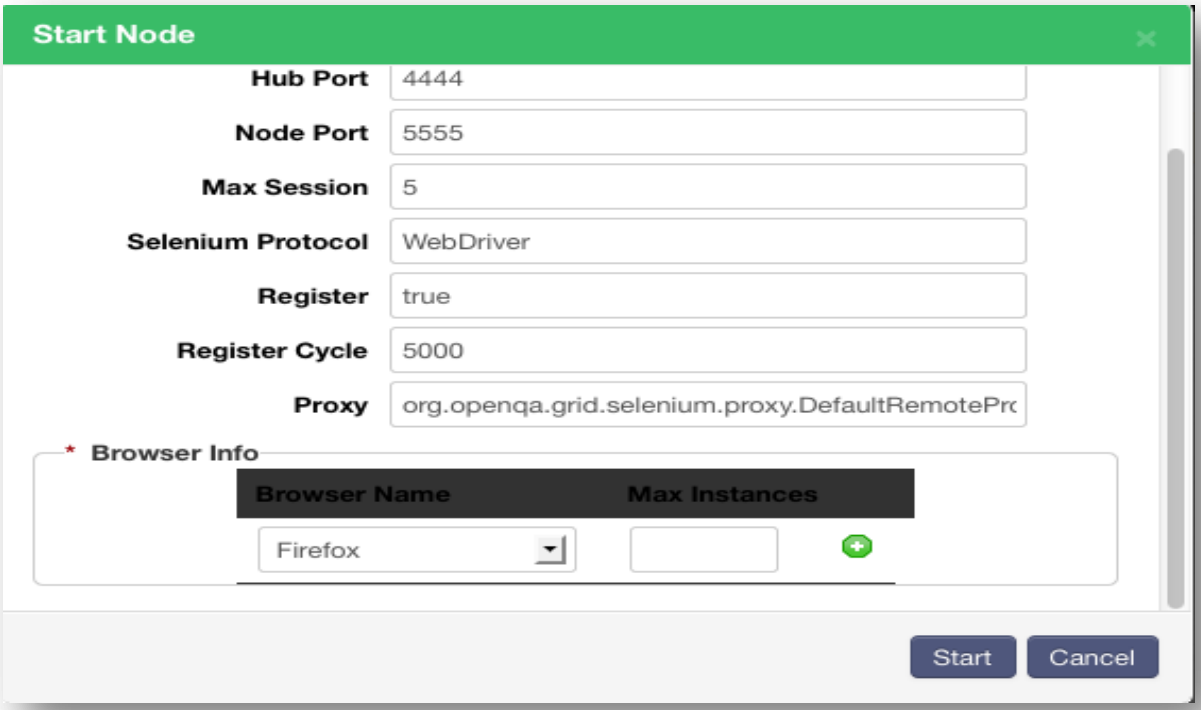

Figure 5.33: Start Node

Selenium Protocol

It refers to the selenium protocol that is used to run the Tests.

Register

Can be set to either <true | false> and by default it is set to "true" in HeliOS. If configured to true, then the node will try to register itself again according to the register cycle set in milliseconds. It can also be set to false if not required.

Register Cycle

It refers to how often the node will try to register itself again (in milliseconds). It allows restarting the hub without restarting the nodes.

Proxy

It's a class that will be used to represent the node. By Default, "org.openqa.grid.selenium.proxy.DefaultRemoteProxy" value is set.

Browser Info

This section is used to select the combination of browsers and their maximum instances. The Node starts on clicking the "**Start**" button after filling all the required details in the Start Node popup.

Test

On clicking the "**Test**" button, the below popup appears.

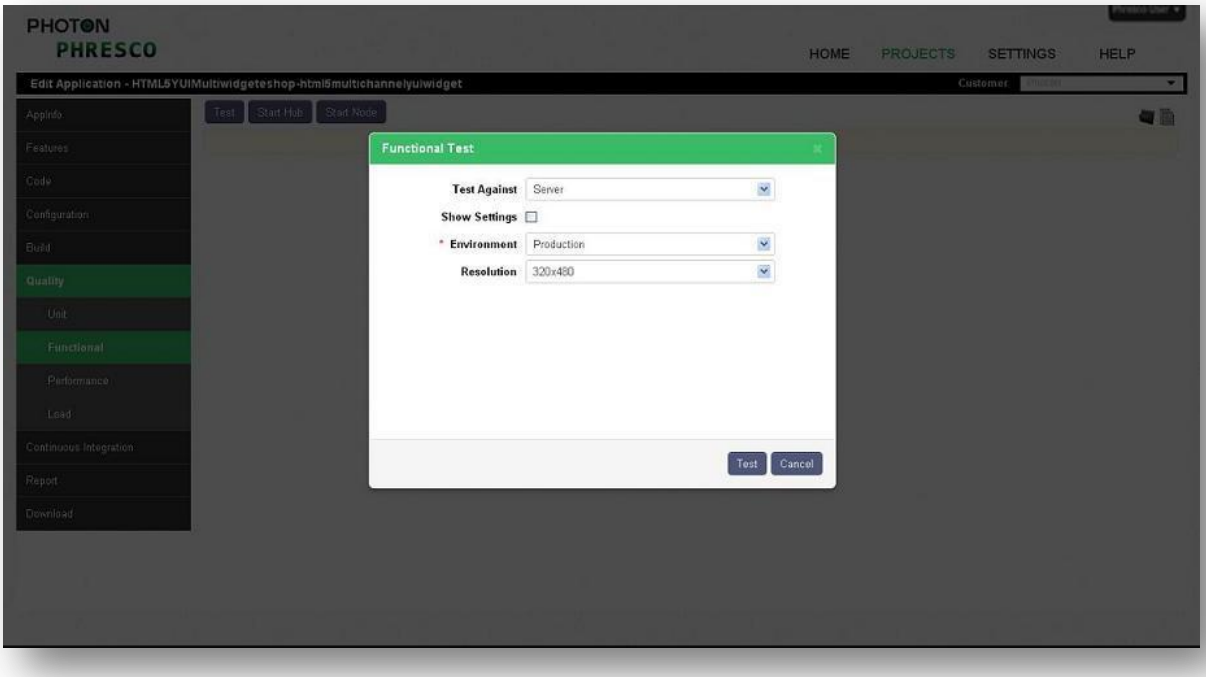

Figure 5-34: Functional test pop-up

Fields in the functional test pop-up are as follows

Test Against

This option can be selected to run functional test against the project that is deployed in the server.Makesure that the project deployed in the server before performing this test.

Environment

Created environments can be selected from the drop down box. If the environment is not created default environment will be selected.

Resolution

The resolution size of the browser can be selected from the dropdown available for the functional test to execute. The resolution size is the view port of the browser.

5.8.2.3 Functional Test execution for Java Standalone

The functional test cases can be run in different browsers that are supported by HeliOS and the resolution for the browser window can also be selected by the users when the functional test tab is clicked.

Fields in the functional test execution are as follows

Test Against

This option can be selected to run functional test against the project that is deployed in the server.Makesure that the project deployed in the server before performing this test.

JAR Location

Use this option to upload the JAR file needed to perform the test.

Figure 5.35: Functional Test execution for Java Standalone

5.8.2.4 Functional Test execution for Share Point

The functional test cases can be run in different browsers that are supported by HeliOS and the resolution for the browser window can also be selected by the users when the functional test tab is clicked.

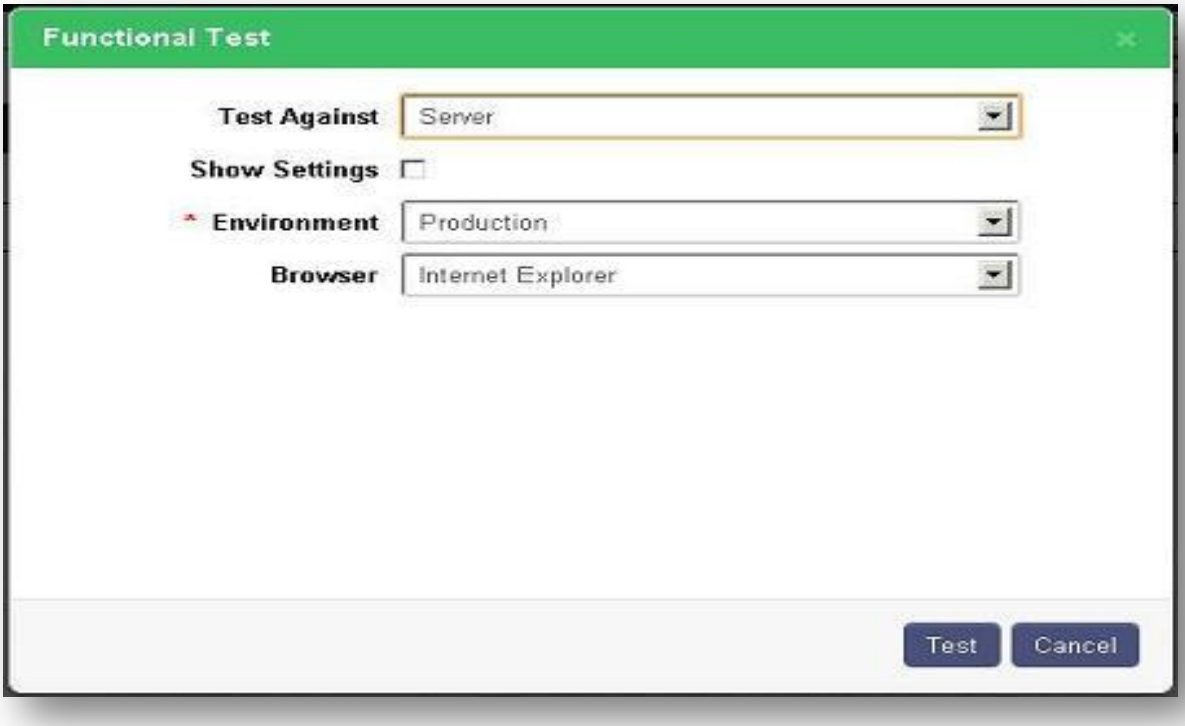

Figure 5.36: Functional Test execution for Sharepoint

Test Against

This option can be selected to run functional test against the project that is deployed in the server.Makesure that the project deployed in the server before performing this test.

Show settings

The global configurations for server, database, web service and email can be selected using the show settings option.

Environment

Created environments can be selected from the drop down box. If the environment is not created default environment will be selected.

Browser

The browser in which the functional test case has to be executed can be selected from the dropdown box. The supported browsers in HeliOS will be listed in the dropdown list.

5.8.2.5 Functional Test execution for Android

Click on the application created in **Project** page.Execute the functional test as shown by navigating to Quality tab: **Application created -> Quality -> Functional -> Test**.

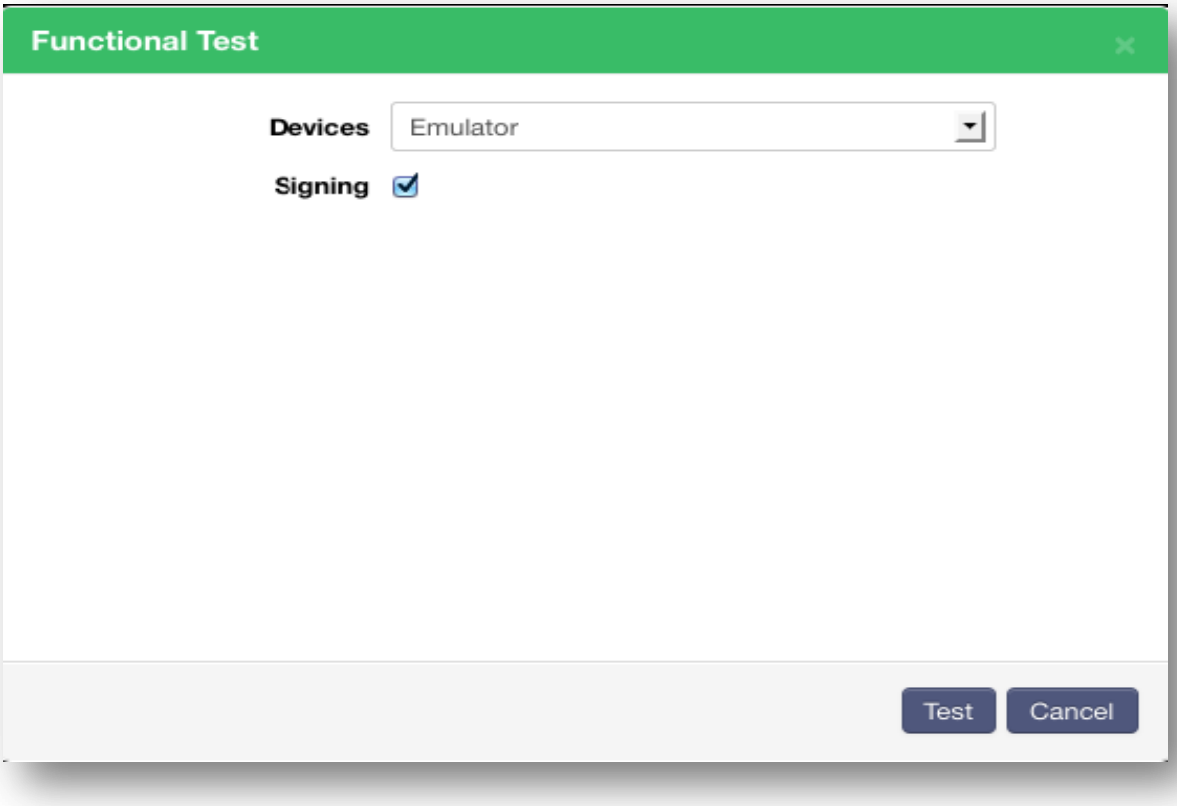

Figure 5-37: Functional Test pop-up for Android application

Devices

Devices in which tests needs to be executed can be selected from this drop down box.

Signing

Android system requires that all installed applications be digitally signed with a certificate whose private key is held by the application's developer. In order to make application available in Android store (Market), the application has to be certified. This can be selected if signing functionality is required.

5.8.2.6 Functional Test execution for iPhone

Click on the application created in **Project** page.Execute the functional test as shown by navigating to Quality tab: **Application created -> Quality -> Functional -> Test**.

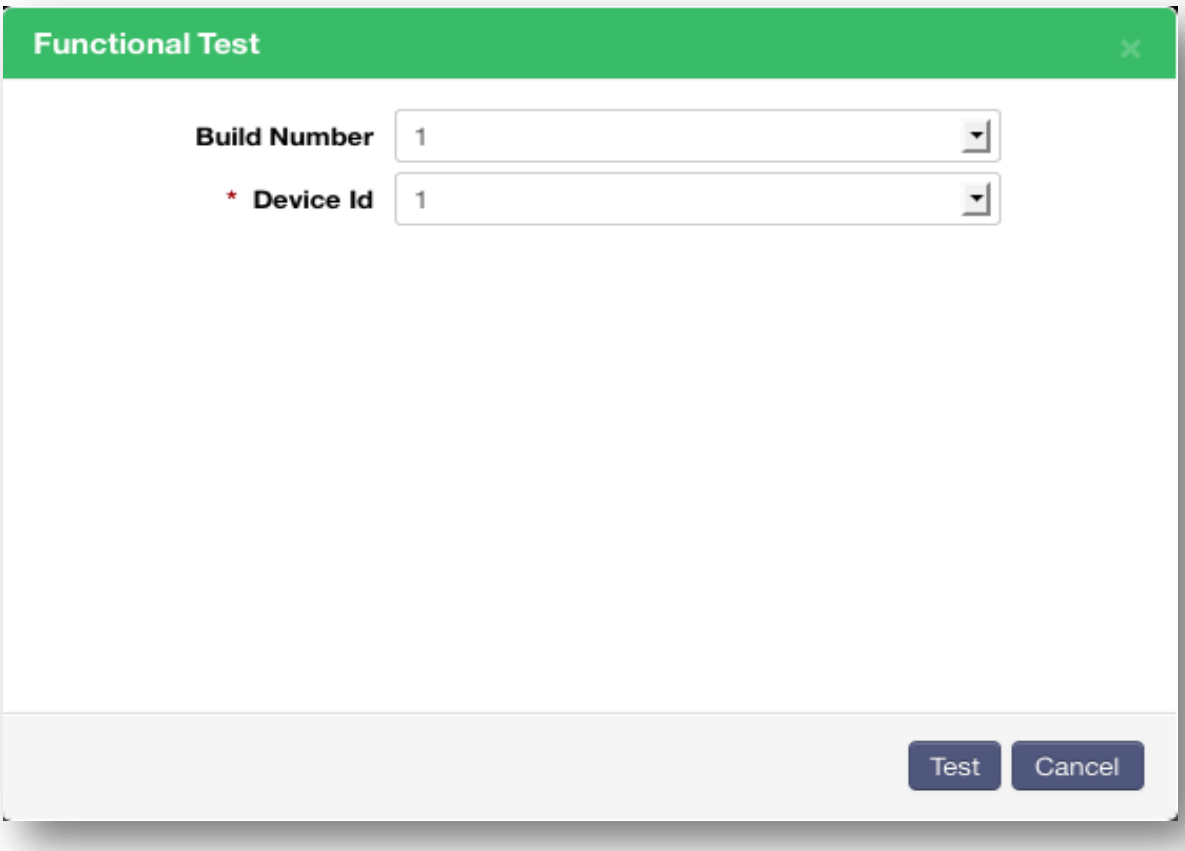

Figure 5-38: Functional Test pop-up for iPhone application

Build Number

Build Number against which the Test to be executed can be selected from this drop down box.

Device Id

If the Test is to be executed in device then the device Id should be mentioned in this field.

Note:

A PDF report of the executed tests can also be generated using "Report" icon as shown 

5.8.3 Performance Testing

Performance testing determines the performance of a system in terms of receptiveness and solidity under a particular workload. This testing also determines the speed or effectiveness and the response time or throughput of a project. In HeliOS the result of the testing is given through static analysis and test cases.

5.8.3.1 Performance Test execution for Android

Device List

Devices in which tests needs to be executed can be selected from the drop down box.

Signing

Android system requires that all installed applications be digitally signed with a certificate whose private key is held by the application's developer. In order to make application available in Android store (Market), the application has to be certified. This can be selected if need signing functionality.

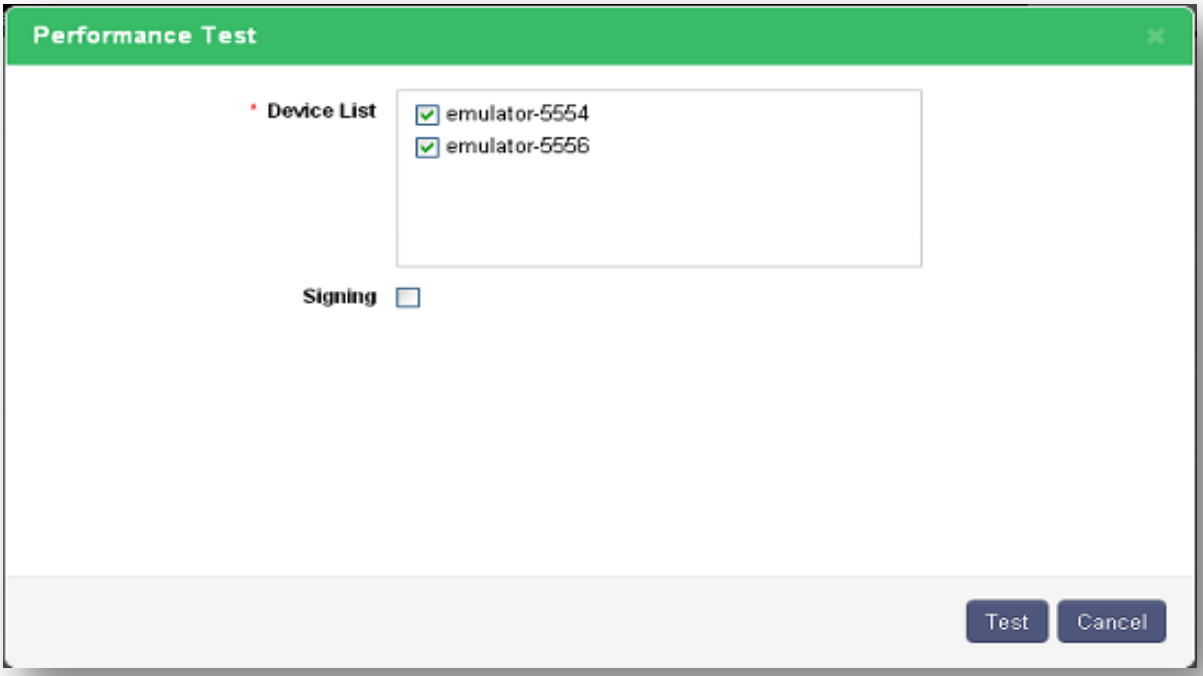

Figure 5-39: Performance test execution for Android

5.8.3.2 Performance Test execution for other Technologies

On clicking the **Test** button in the Performance tab, the following popup appears

Test Against

This option is used to run Performance test against the project that is deployed in the server.It can also be used to run the test againt the database or the web services used in the project. These values are available in the drop down and the user can select the required one.

Show Settings

The global configurations for server, database, web service and email can be selected using the show settings option.

Environment

Created environments can be selected from the drop down box. If the environment is not created, default environment will be selected.

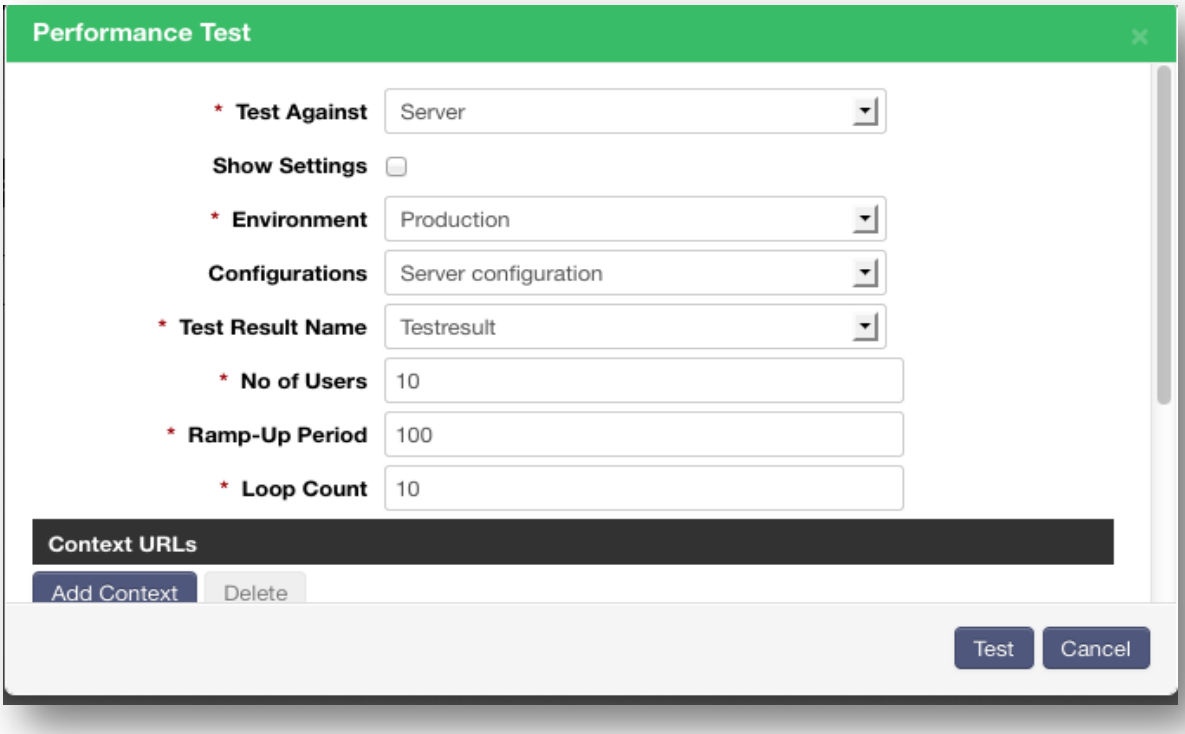

Figure 5.40: Performance Testing for other technologies

Configurations

The name of the configuration created in configuration page will be dispalyed in this field.

Test Result Name

The name of the Test can be entered in this field.

No of Users

This field defines the number of users to access the application at the same time.

Ramp-Up Period

This field defines the time period in milliseconds within which the performance test has to be carried out.

Loop Count

Loop count determines the number of times each user needs to access the Application.

Context URLs

Context URLs specify the name and context of the Application to be tested together with its type & encoding values. Headers contain the key and the value details. An instance is given below.

Key- Content-Type
Value - application/json

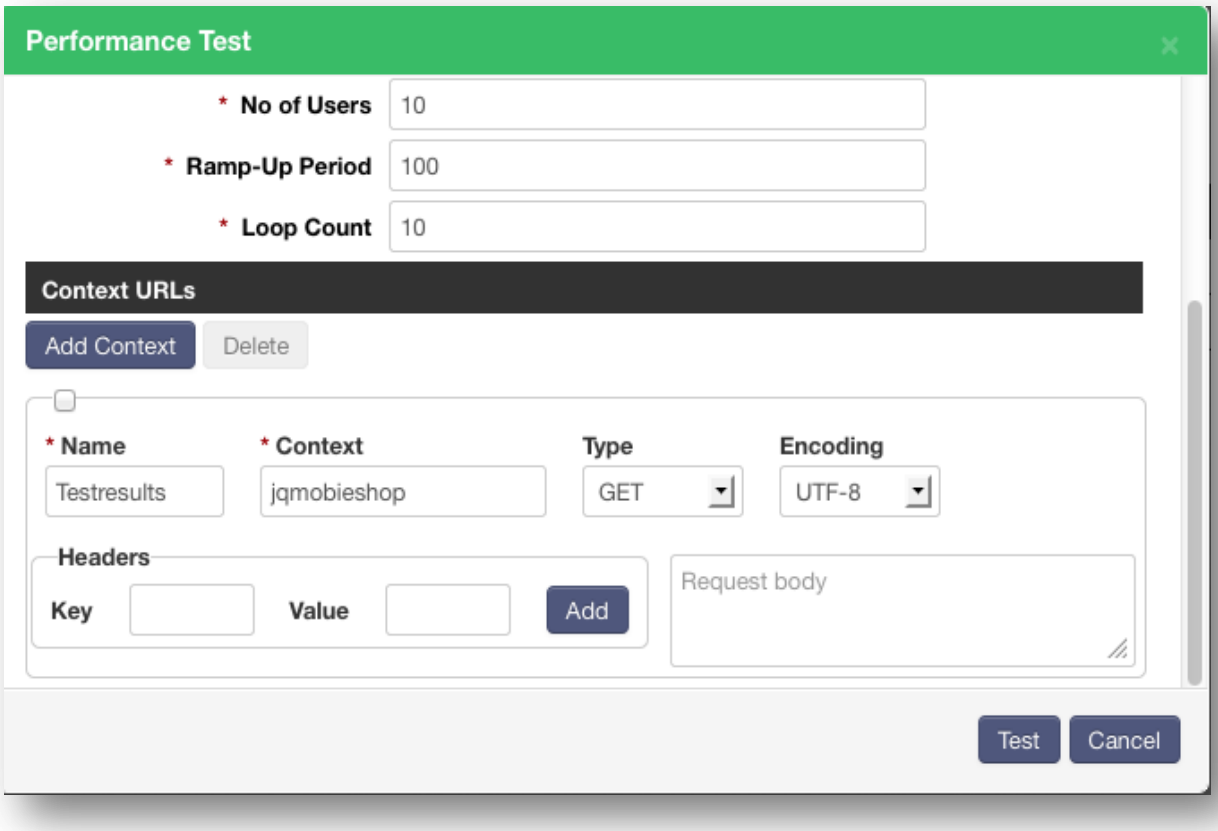

Figure 5-41: Adding Context URL's in Performance Testing

Note:

A PDF report of the executed tests can also be generated using "Report" icon as shown

5.8.4 Load Testing

Load testing refers to modeling the expected usage of a project by simulating the access of multiple users to the same project concurrently. Load and performance testing is usually conducted in a test environment identical to the production environment before a project is permitted for real time usage.

Load testing also uses jMeter. The result is given through static analysis and test cases. The test cases will point out the error and this will be very useful for the testing team.
On clicking the **Test** button in the Load tab the following popup appears

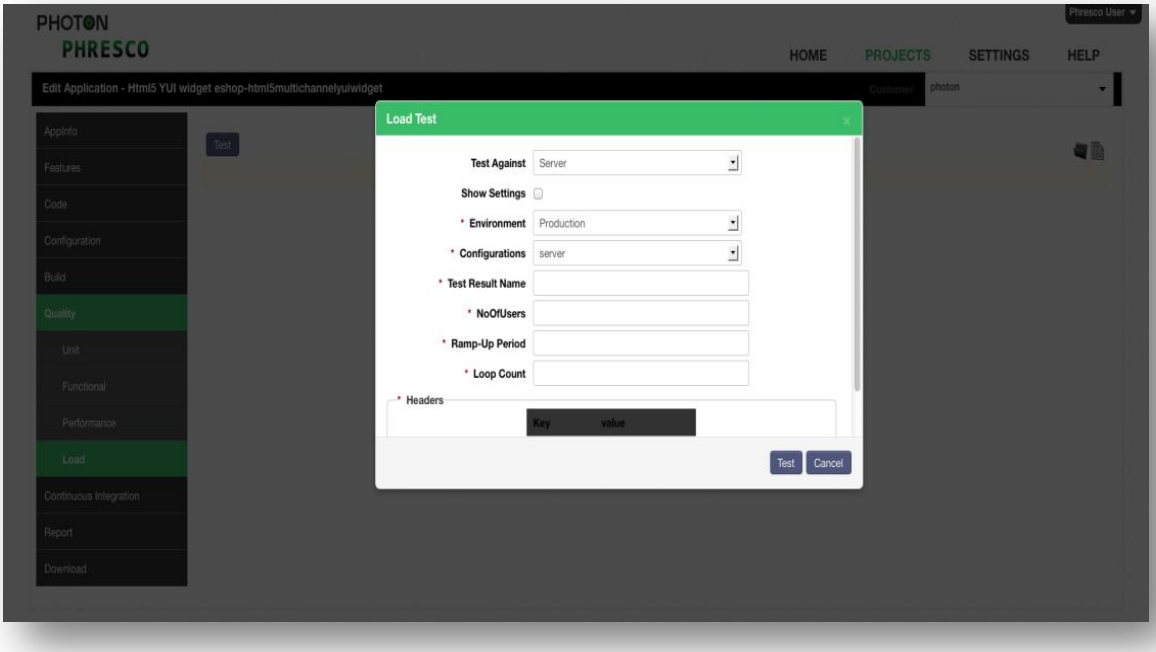

Figure 5-42: Load Testing

Test Against

To run Load test against the project that is deployed in the server or the test can to run againt the database or the web service used in the project. These values are available in the drop down and the user can select the one required.

Show Settings

The global configurations for server, database, web service and email can be selected using the show settings option.

Environment

Created environments can be selected from the drop down box. If the environment is not created default environment will be selected.

Configurations

The name of the configuration created in configuration page will be dispalyed in this field.

Test Result Name

The name of the Test can be entered in this field.

No Of Users

This field defines the Number of users to access the application at the same time.

Ramp-Up Period

This field defines the time period in ms within which the performance test has to be carried out.

Loop Count

Loop count determines the number of times each user needs to access the Application.

Headers

The key and the value details can be entered in Headers section.
Eg., Key- Content-Type
value - application/json

Note:

A PDF report of the executed tests can also be generated using "Report" icon as shown 

5.9 Continuous Integration

Continuous Integration process is used for generating the builds, deploying the generated build and testing the deployed project. The build, deploy and test process that are generated manually using build, deploy and test option respectively can be made automatic by scheduling the time. This automation process is done using Jenkins.

HeliOS Framework also has the option of sending the build result directly to the developers' inbox.

5.10 Report

Report tab in HeliOS is used to generate a site for the project. The generated project site includes the project's reports that were configured in the POM.

Below are the lists of reports that can be configured using HeliOS:

- Project-Info-Report

  Enabling this gives a report of all the information about the project. This report is common across all the technologies. Options listed under "Project-Info Report" are explained below:

  - index: index of the project. This is a default enabled option.
  - modules: list of description of each module used in the project.
  - dependencies: list of dependencies collected from POM.
  - cim: continuous integration management report.
  - scm: source configuration management report.
  - summary: summary of the project.
  - licence: licence that is used in the project.

- JavaDoc Report

  Enabling this option gives a report of Java source files and produces a JavaDoc which has browsable source code containing hyperlinked cross-references within and across that set of files. This is applicable only for the java technology projects.

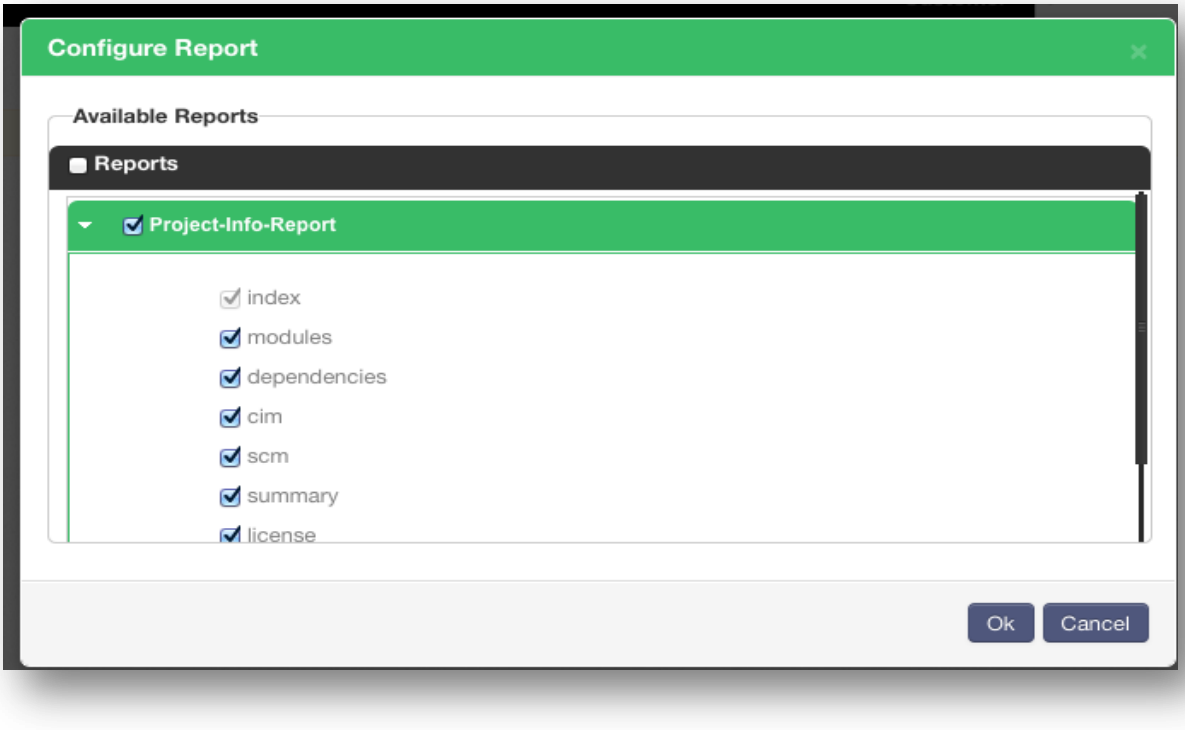

Figure 5-43: lists in Project-Info-Report

- jDepend Report

  Enabling this option generates a report of quality metrics for each java package. Degree of dependencies, its extensibility and reusability is measured and a report is generated on it. This is applicable only for the java technology projects.

- JXR Report

  Enabling this option generates a report of the source code in the html format which can be viewed in the browser. Cross-reference of the project's source is generated which allows the users to find the specific lines of code. This is applicable only for the java technology projects.

- PMD Report

  Enabling this option generates a report of the metrics using the PMD tool. This tool helps in sorting out the test case that are mostly used and the ones that are not important. The unimportant sets can be ruled out to suppress the code. This is applicable only for the java technology projects.

- Surefire Report

  Enabling this option generates a report of the unit test results. This is applicable only for the java technology projects.

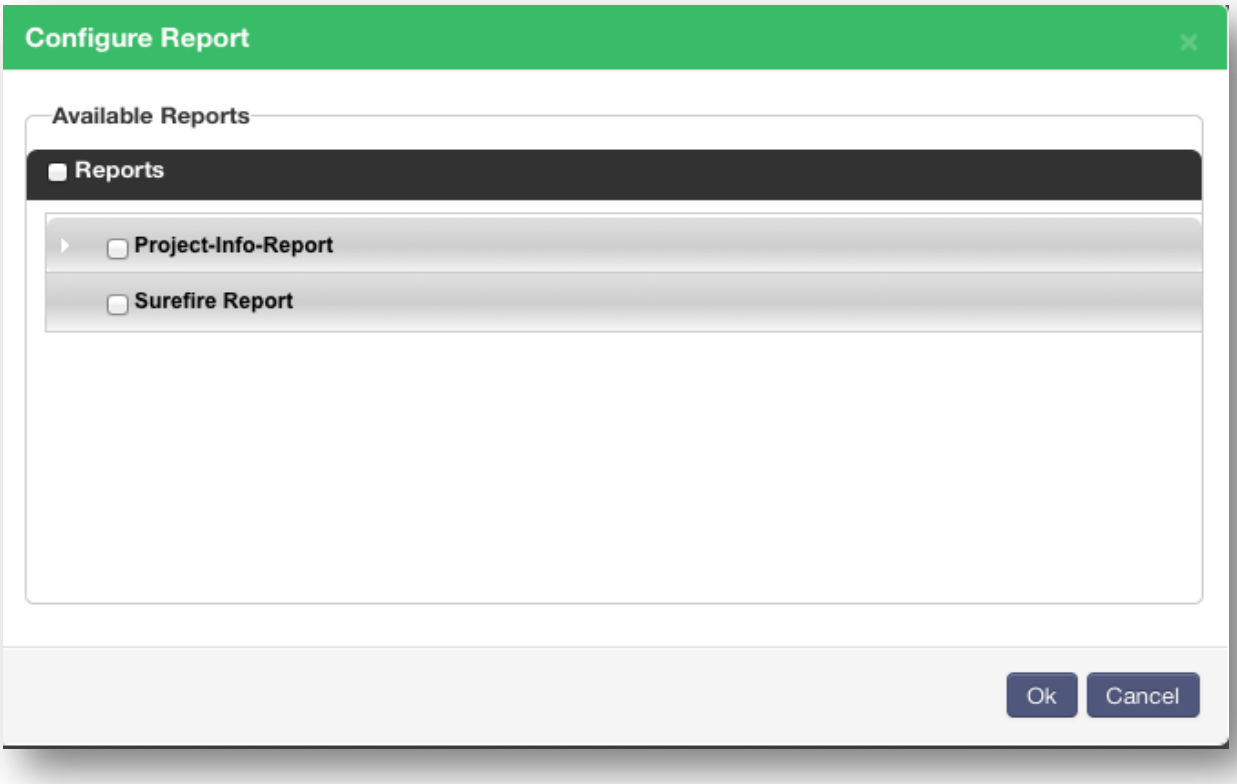

Figure 5-44: Configure Report tab

5.11 Knowledge Repository

A knowledge repository is a computerized system that systematically captures, organizes and categorizes an organization's knowledge.

Knowledge repository in HeliOS is the central repository which contains the application types, archetypes and features. It is a common repository where the admin can perform changes or modifications that will impact the user interface.
A Common Knowledge repository exists where any changes made will be reflected across all the account holders. Each customer will have their own knowledge repository for their artifacts. Any changes in Customer Knowledge repository can be witnessed in the customer's user interface.

5.12 Download

Downloads in HeliOS are the configurations that are provided to the developers out of the box. Developers can choose the servers or databases or even editors that are available in the download.

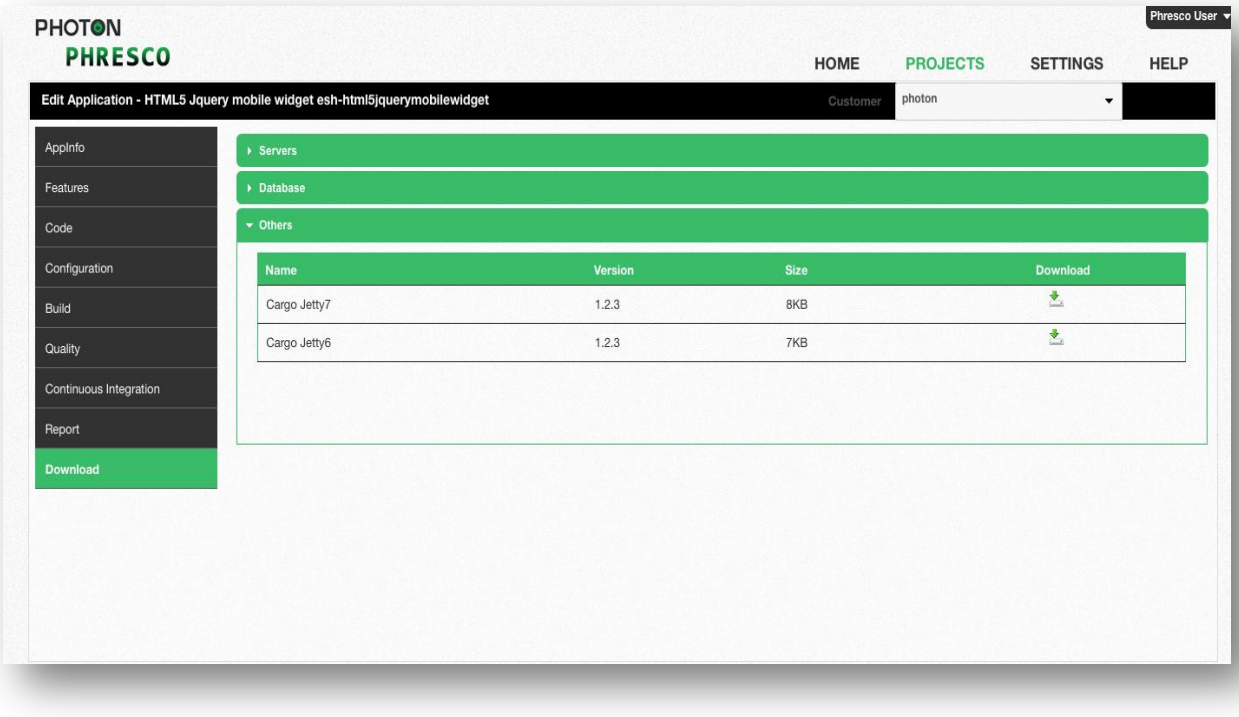

Figure 5-45: Third party downloads

5.13 iPhone Prerequisites

iOS Sim

To perform application test for iPhone, **iOS-Sim** tool needs to be added in the tools folder and path should be set.

*export IOS_SIM=/**Users**/**tools**/**ios-sim**/build/Release*
export PATH=$IOS_SIM:$PATH

In the command prompt, go to ios-sim folder and execute the following command to install iOS-Sim. This is a onetime install.

rake install prefix=/usr/local/

Wax Sim

While deploying and testing the IOS application, HeliOS Framework provides the option of selecting iPhone simulator or iPad simulator from the dropdown box listed under family. This is done by integrating waxsim plugin in the Framework.

Waxsim tool should be installed manually from the directory path *phresco-framework/workspace/tools/waxsim.*

Go to command prompt and execute the below command to install waxsim in the local machine.

xcodebuild install DSTROOT=

For developers

1. Mac machine
2. Mac OS 10.7 & above
3. Xcode tool 4.2 & above
4. iPhone Simulator version - 4.3, 5.0 and above
5. iPad simulator version - 4.3, 5.0 and above
6. iPad Devices - iPad 2 and iPad 3(Only for Native apps and support not given to iPad 3 to develop Phonegap applications)

For testers

1. Mac machine
2. Mac OS 10.7 & above
3. Xcode tool 4.2 & above
4. iPhone Simulator version - 4.3, 5.0 and above
5. iPad simulator version - 4.3, 5.0 and above
6. iPad Devices - iPad 2 and iPad 3(Only for Native apps and support not given to iPad 3 to develop Phonegap applications)
5. Instrumentation tool for automation test

Deploying Devices

1. iPhone 4, 4s, iPad 2

Note:

- Developers or testers should have registered in develop.apple.com or should have an id for downloading the following software.
- Development certificate is required for deploying a project in the device.

5.14 Android Prerequisites

To build a project:

HeliOS supports versions like 2.2, 2.3.3 and 4.0.3 for building a project in ANDROID

To deploy a project

Project can be deployed depending on the <minSdkVersion> which is defined in the manifest.xml file.

- Manifest.xml code:

```
<?xml version="1.0" encoding="utf-8" ?>
- <manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.photon.Phresco.nativeapp" android:versionCode="1"
android:versionName="1.0">
<uses-sdk android:minSdkVersion="7" />
<application android:icon="@drawable/icon"
android:label="@string/app_name">
< activity android:name=".activity.MainActivity"
android:label="@string/app_name">
<intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
</application>
</manifest>
```

Depending on the versions given in the manifest.xml project can be deployed. In the above example

"<uses-sdk android:minSdkVersion="7" /> "

7 is the version mentioned while a project can be deployed in the versions of 7 and above 7.

Table 2: Android API levels

Platforms	API Level
Android 1.5	**3**
Android 1.6	**4**
Android 2.1	**7**
Android 2.2	**8**
Android 2.3- 2.3.2	**9**
Android 2.3.3-2.3.7	**10**
Android 3.0	**11**

Android 3.1	**12**
Android 3.2	**13**
Android 4.0-4.0.2	**14**
Android 4.0.3-4.0.4	**15**
Android 4.1	**16**
Android 4.1.2	**16**
Android 4.2	**17**

Prerequisites for android for developers and testers

1. User should install Eclipse 3.7[indigo] IDE or above.
2. Also Android SDK for 2.3.3 or higher platform (API level 10 or higher) should be installed
3. "ANDROID_HOME" environment variable should be set, and should point to android sdk folder /tools and /platform-tools should be system PATH variable.
4. For Android projects, avoid using Android versions 1.6 & 2.1_R1 for project creation and build generation

5.15 Blackberry Prerequisites

Prerequisites for running Blackberry application:

In order to start with BB hybrid application development, two steps needs to be followed:

- Necessary software installations.
- Code Signing key registration and installation

5.15.1 Software Installations:

Following software needs to be installed for BB hybrid development.

- BlackBerry WebWorks SDK for Smart Phones
- BlackBerry Desktop Software
- BlackBerry JDE Plugin

BlackBerry WebWorks SDK for Smart Phones:

BlackBerry WebWorks SDK for Smart Phones has to be installed on machine.

Windows

https://developer.blackberry.com/html5/downloads/fetch/BlackBerryWebWorksSDK.exe

Mac

https://developer.blackberry.com/html5/downloads/fetch/BlackBerryWebWorksSDK.zip

Create Environment variable

Name: BB_WEBWORK_SDK_HOME
Value: <PATH TO SDK INSTALLATION> [E.g: C:\Program Files\Research In Motion\BlackBerry WebWorks SDK 2.3.1.5]

Add following 2 variables to Path

%BB_WEBWORK_SDK_HOME%;
%BB_WEBWORK_SDK_HOME%\bin;

Blackberry Desktop Software

Install the BlackBerry Desktop Software to simulate the usb connection with simulator. Go through following links and install the software.

Windows

https://www.blackberry.com/Downloads/contactFormPreload.do?code=A8BAA56554F96369AB93E4F3BB068C22&dl=A2C0D61EB187AB3AFD247A852FAD3647

Mac

https://www.blackberry.com/Downloads/contactFormPreload.do?code=CBC462E27100DAD71CDBF606D396DDAD&dl=3C4BB676CED2EDE17E0996B0A4A20B01

BlackBerry JDE Plugin Installation

Get the BB JDE plugin from following links:

Windows

https://developer.blackberry.com/java/downloads/fetch/BlackBerry_JDE_PluginFull_2.0.0_indigo.exe.

Mac

https://developer.blackberry.com/java/downloads/fetch/BlackBerry_JDE_PluginFull_2.0.0_indigo.zip

Download the BB plugin and install.

Once installed, start the IDE and follow below mentioned steps: [Note: steps mentioned below are specific for windows installation. Kindly follow the appropriate steps for Mac as instructed on screen]

- Go to File -> New -> BlackBerry Project. The wizard will help developer to create new project.
- Enter Project name in next step.
- Under Template selection window, choose BlackBerry Application.
- Enter necessary information in Application Details step. Click Finish.
- New BlackBerry application will be available in list.
- Right click on project, select Run As -> BlackBerry Simulator. This will launch BB simulator.

Note for simulator: Once the simulator is launched, select "Simulate->USB Cable Connected" and then you can use javaloader as if it was an actual device connected via USB.

5.15.2 Code signing key registration and installation:

Once the components are installed, Code Signing Keys need to be requested and installed on development machine. Code signing key installation process is one time process.

Getting the Key Files:

- Open

  https://www.blackberry.com/SignedKeys/codesigning.htmlhttps://www.blackberry.com/SignedKeys/codesigning.html

- Enter required details as described in image below, and click submit. [Note: Kindly note down the Registration PIN. It will be required in key installation process later on]

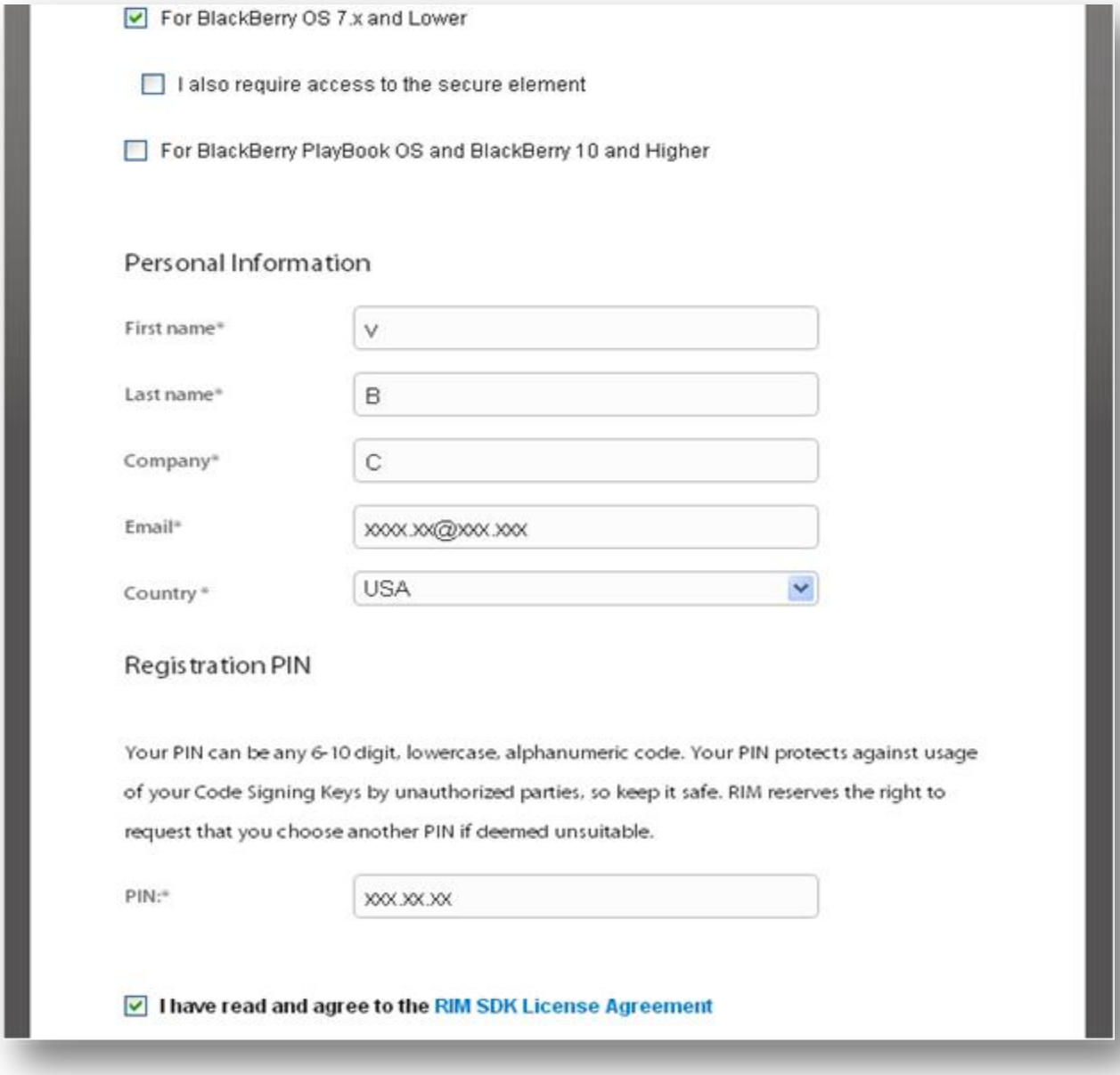

Figure 5.46: Getting Key details

- Once details are submitted, you will get confirmation page on screen, stating that .csi files will be sent within 2 hours. Probably the email will be sent within 15-20 mins.

- Once the email is delivered in your inbox with 3 .csi files, go to https://developer.blackberry.com/CodeSigningHelp/codesignhelp.html . Select the highlighted option as shown in image, and click Next

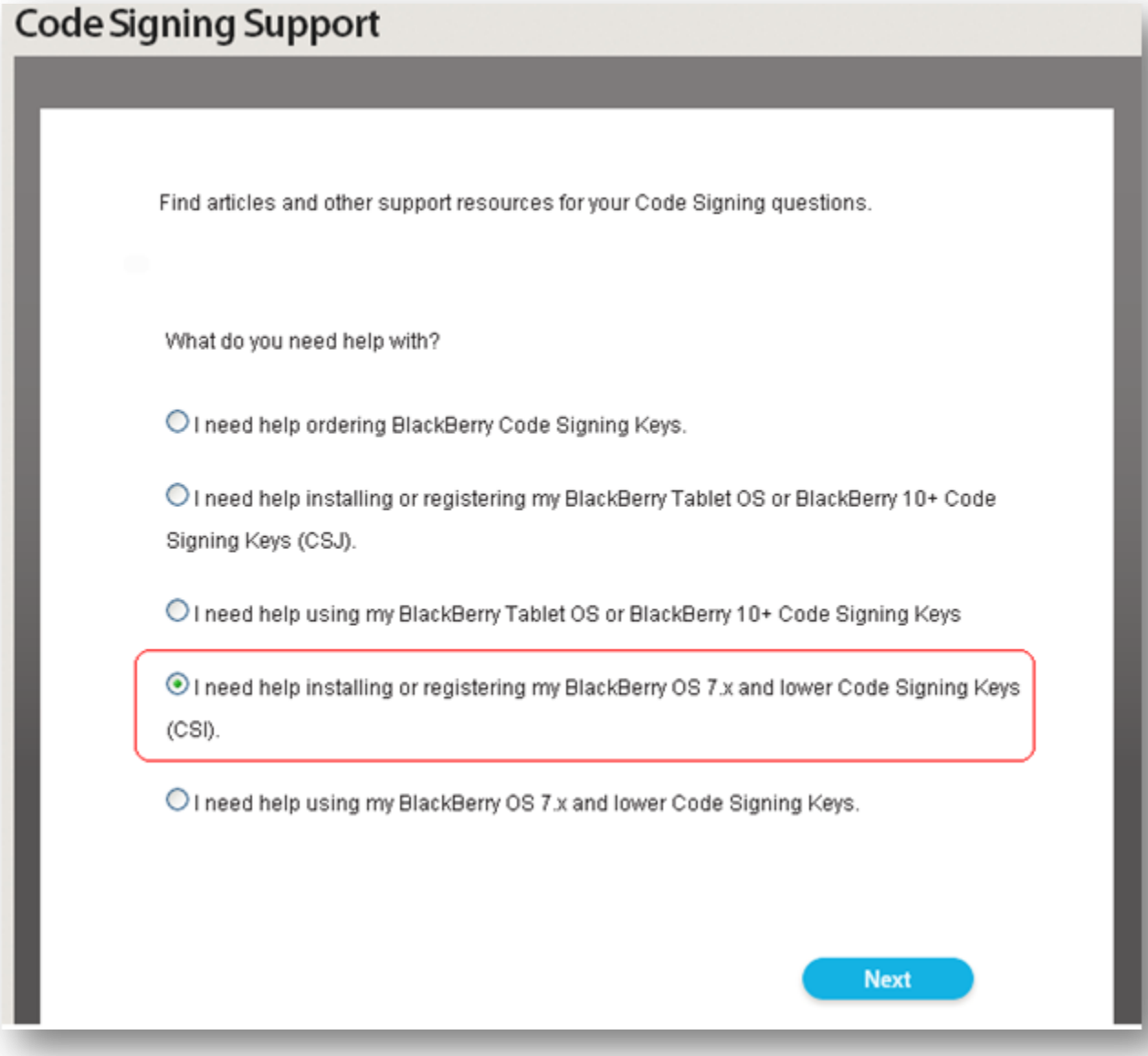

Figure 5.47: Code Signing support

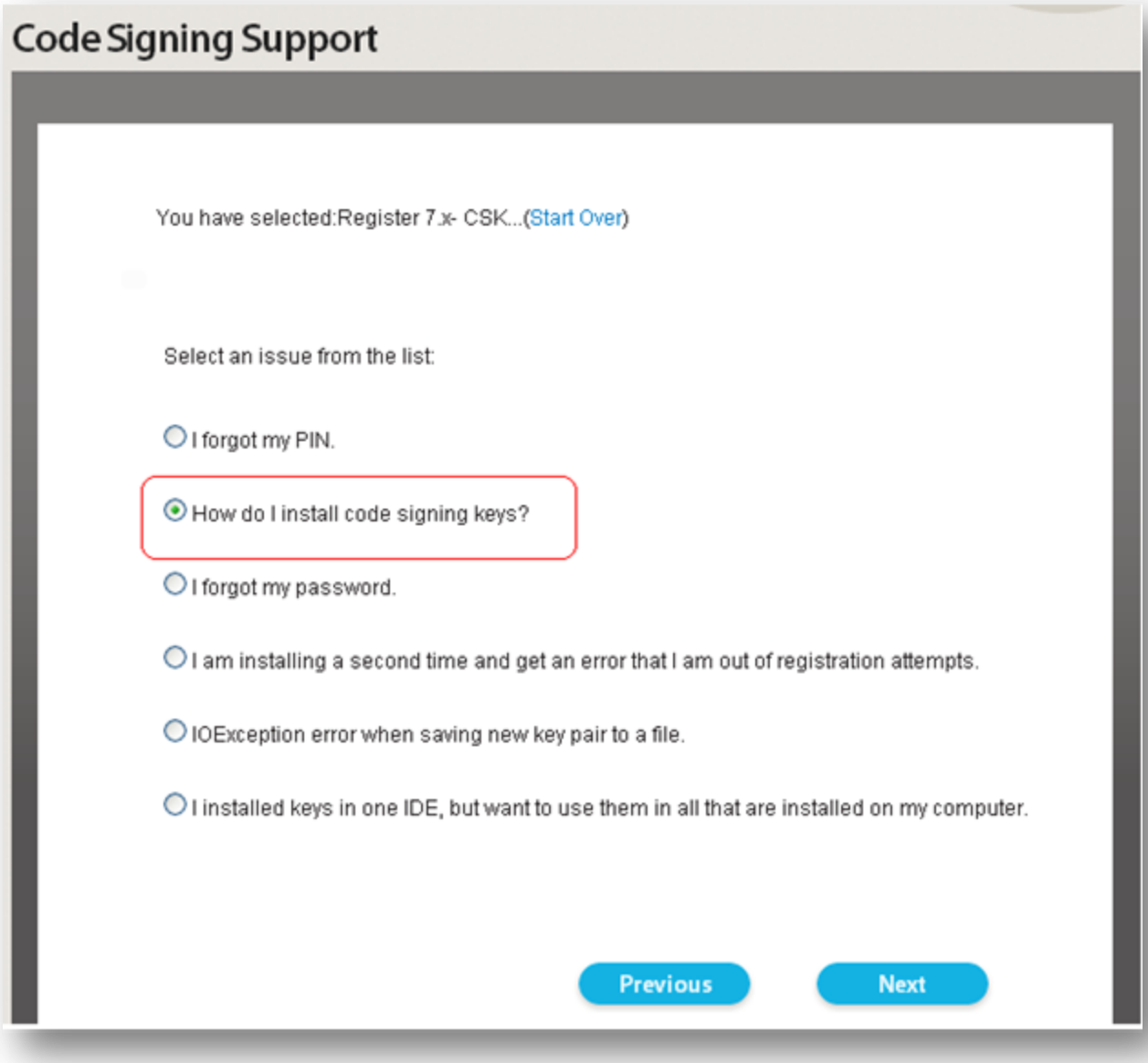

Figure 5.48: Code Signing Support

Figure 5.49: Code signing support

- Scroll down to the bottom of the opened page, and click on the highlighted option, as shown in image below.

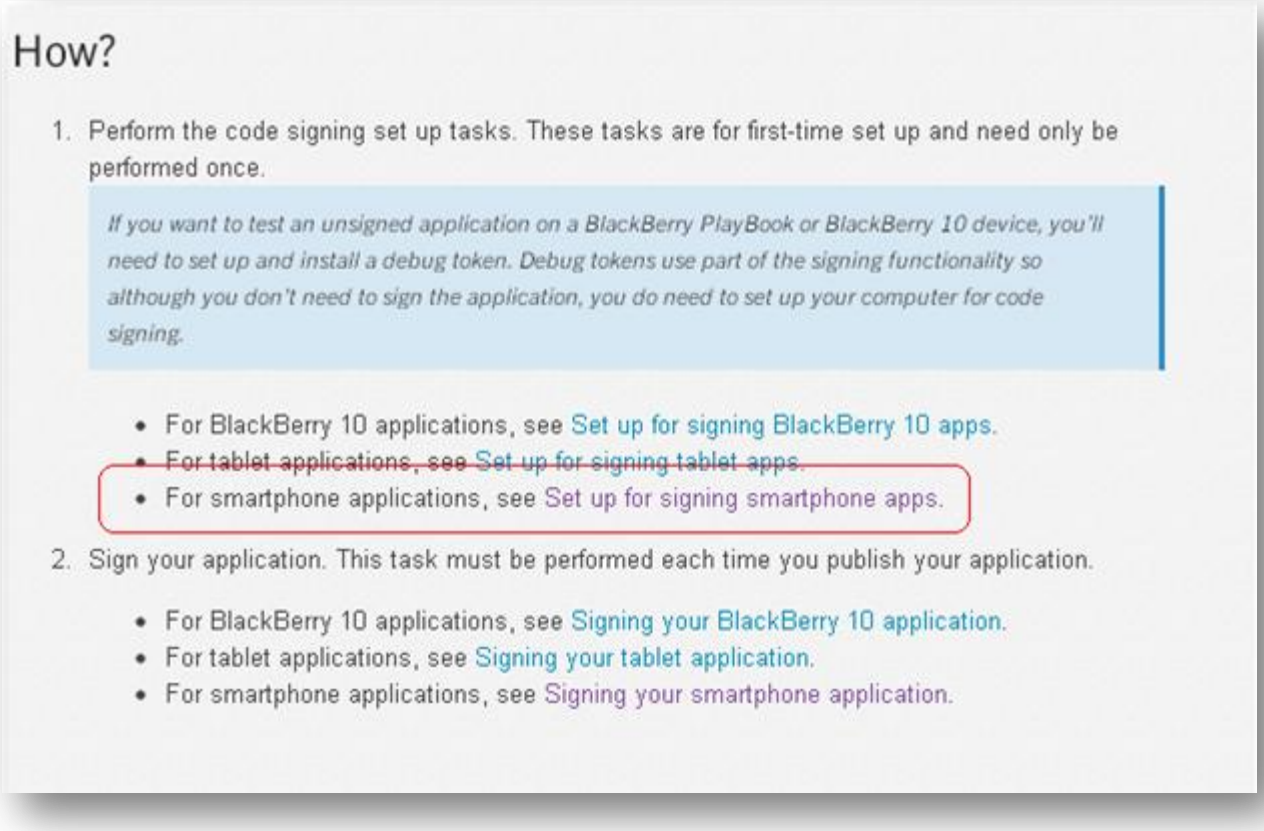

Figure 5.50: Setup for signing smartphone apps

- This will open "**Set up for signing smartphone apps**" page. Follow all the 10 steps mentioned in **Requesting and Registering your keys** section for all the 3 .csi files that you have received in emails earlier.

Note:
This is required to get the keys in order to run the application on BB device.

5.16 Prerequisites for Windows Phone

- Visual Studio 2010 or higher should be installed
- Windows Phone SDK 7.0 or higher should be installed [Note: For Using Windows phone SDK 8.0, 64bit OS is required
- Create MSBUILD environment variable pointing to C:\Windows\Microsoft.NET\Framework\<Framework_version> directory on your machine, containing the MSBuild.exe file

  E.g., C:\Windows\Microsoft.NET\Framework\v3.5 (Or)
  C:\Windows\Microsoft.NET\Framework\v4.0.30319

- Add MSBUILD environment variable in your Path
  1. Edit your existing Path variable.
  2. If semi colon (;) is not present at the end, place one semi colon(;) at the end.
  3. Append %MSBUILD%; at the end.

For 32-bit,

- The console tools are available under the Downloads section in HeliOS
- Download and extract it at desired location on your machine
- Create WPTOOLS_HOME environment pointing to extracted directory (containing wptools.exe)
- Add WPTOOLS_HOME environment variable in your Path

  1. Edit your existing Path variable.
  2. If semi colon (;) is not present at the end, place one semi colon(;) at the end.
  3. Append %WPTOOLS_HOME%; at the end.

For 64-bit,

- The console tools are available under the Downloads section in HeliOS
- Download and extract it at desired location on your machine
- Extract it at desired location on your machine
- Create XAPDEPLOYCMD_HOME environment pointing to extracted directory (containing wptools.exe)
- Add XAPDEPLOYCMD_HOME environment variable in your Path

  4. Edit your existing Path variable.
  5. If semi colon (;) is not present at the end, place one semi colon(;) at the end.
  6. Append % XAPDEPLOYCMD_HOME_HOME%; at the end.

5.17 Prerequisites for Windows Metro

- Windows 8 OS
- .Net Framework 4.0 or higher should be installed
- Visual Studio Express 2012 RC for Windows 8 should be installed (IDE
- Create MSBUILD environment variable pointing to C:\Windows\Microsoft.NET\Framework\<Framework_version> directory on your machine, containing the MSBuild.exe file.

  E.g., C:\Windows\Microsoft.NET\Framework\v4.0.30319

- Add MSBUILD environment variable in your Path

  1. Edit your existing Path variable
  2. If semi colon (;) is not present at the end, place one
  3. Append %MSBUILD%; at the end.

5.18 Prerequisites for NodeJS

- Download and install Node JS. (NodeJS is available under Downloads tab in the HeliOS Framework). The NodeJS version should be equal to or greater than node-v0.8.7-x86.msi
- Set the path of installed Node Js in the environment variables
- Mocha tool is essential for performing code validation, build and unit test. Open the command prompt and execute the following commands inside the installed Node Js

  npm install -g mocha
  npm install -g mocha_lcov_reporter
  npm install -g should

- The code coverage tool, Jscoverage, will be available under the Others section of the Downloads tab in the HeliOS Framework. After creating a Node Js project using HeliOS, download and unzip it at any system location
- Set the path of this jscoverage folder in the environment variables.

Note:

Installing NodeJS is a one time event

---

6 Archetypes

Archetype, provided in HeliOS, is an ideal project from which similar projects can be created. HeliOS Archetypes help the developers follow the best practices in creating projects by providing basic templates for any technology. Developers can create archetypes and deploy it in their organization's repository which will be available for the use of all developers within their organization. Users can also upload archetypes required for their projects with the help of admin console.

6.1 List of Available Archetypes

Archetypes are classified under application, web and mobile layers. Under each applications there are list of Archetypes available as given below.

List of Archetypes for Web Applications

- PHP
- Drupal6
- Drupal7
- SharePoint
- ASP.Net
- SiteCore
- WordPress
- Java Standalone
- HTML5 Multichannel YUI Widget
- HTML5 Multichannel jQuery Widget
- HTML5 YUI Mobile widget
- HTML5 jQueryMobile Widget

List of Archetypes for Mobile Applications

- iPhone Native
- iPhone Hybrid
- iPhone Library
- iPhone Workspace
- Android Native
- Android Hybrid
- Android Library
- Blackberry Hybrid
- Windows Metro
- Windows Phone

List of Archetypes for Web Services Applications

- Java Web service
- NodeJS

7 Reusable Components

HeliOS promotes reusability of components by providing a knowledge repository that systematically captures, organizes and categorizes an organization's knowledge.

Archetype:

Archetype, provided in HeliOS, is an ideal project from which similar projects can be created. HeliOS Archetypes help the developers follow the best practices in creating projects by providing basic templates for any technology. Users can also upload archetypes required for their projects with the help of admin console.

Features:

The below mentioned resources are represented in HeliOS as features that can be selected when creating a project.

- Java libraries
- Android libraries
- iPhone libraries
- ASP.NET Libraries
- SharePoint Features
- PHP Modules
- Drupal Modules
- WordPress Modules
- NodeJS Modules
- JS Libraries

Pilot Projects:

Pilot project is an actual project built with the best practices of project development, libraries, components and validated code structures. HeliOS has included Pilot projects for most of the technology it supports. The inclusion is intended to cut maintenance time by synchronizing application and design. HeliOS also allows the pilot projects to be re-branded, reused and redelivered when published in the repository.

Third Party Libraries:

HeliOS's repository server is the main vital repository and any changes and modifications made by the administrators have a significant impact on the framework's user interface.

It is a common knowledge repository that controls the content accessed by the entire team. Organizations can move the artifacts and features that they desire to reuse across platforms in to the repository.

What happens when you select a pilot project?

Once a pilot project for the desired technology is selected, it will be added into HeliOS Framework/workspace/projects. Projects a user may create using HeliOS archetypes will be located in the above location.

What happens when you select an Archetype?

Archetypes provided by HeliOS can be adapted to the user's project and the completed project can be hosted in HeliOS's repository for access by other projects in the organization.

7.1 What happens when you select a feature in your project?

Table 3: Features and Js Libraries for the Technologies

Technology	Selecting a feature	Selecting a JS library
Drupal	The modules will be added into <PROJECT_HOME>/source/sites/all/modules	N/A
SharePoint	The features will be added into <PROJECT_HOME>/source	N/A
NodeJS	The modules will be added into <PROJECT_HOME>/source/node_modules	The modules will be added into <PROJECT_HOME>/source/lib
iPhone Native	The libraries will be added into <PROJECT_HOME>/source/Thirdparty	N/A
iPhone Hybrid	The libraries will be added into <PROJECT_HOME>/source/Thirdparty	N/A

PHP	The libraries will be added into <PROJECT_HOME>/source	The libraries will be added into <PROJECT_HOME>/source/public_html/js
WordPress	The modules will be added into <PROJECT_HOME>/source/wp-content	N/A
ASP.NET	The features will be added into <PROJECT_HOME>/source/src	N/A

Note:

For Java, Java WebService, HTML5 and Android the features are updated in the pom.xml as dependencies.

---

8 Testing

HeliOS provides standardized structure for testing all the technologies.

8.1 Test Cases for Java Technology

8.1.1 Unit Test

8.1.1.1 Structure of AllTest and Test Cases

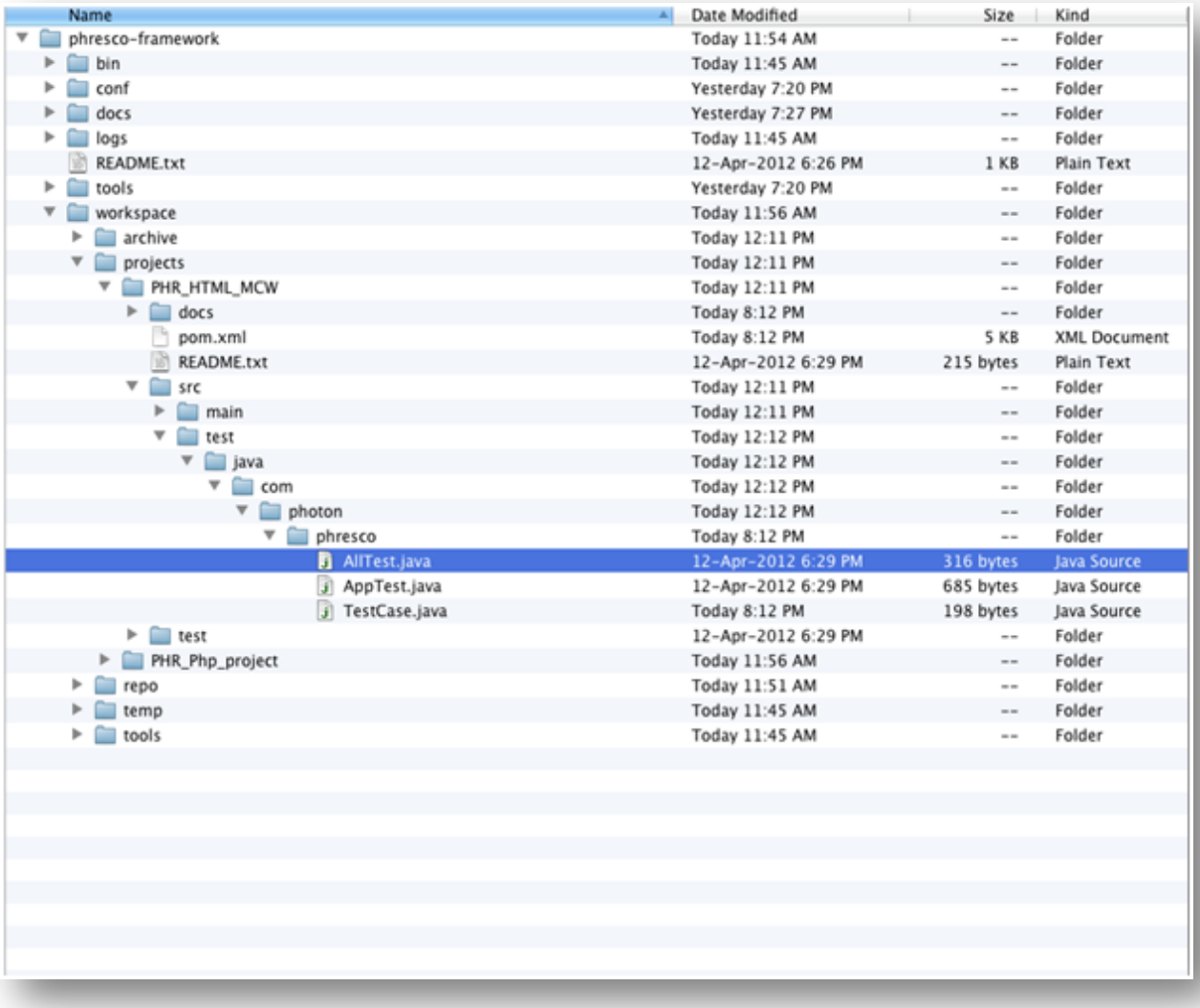

Figure 8-1: Java unit tests structure

a. **AllTest :** AllTest is the root file that carries all the test cases to initiate the testing process. Each test case can be called separately to run the unit test.

b. **Test case:** In unit test, Test cases are written in order to test the source code.

8.1.1.2 Existing Test Cases Out Of the Box in HeliOS

AllTest for Java Technology

AllTest contains a bunch of test cases, each of which performs a unique scenario on the application. Test developers can start writing new suite classes and test cases by following the syntax and structure of HeliOS frameworks out of the box class structure. AllTest class calls all the suite classes and the test cases within it. Once suite classes and test cases are written developers can execute those from HeliOS Framework and can see the report. Following is the AllTest file which shows up how to add / include the class inside it.

```
package com.photon.Phresco;

import junit.framework.Test;
import junit.framework.TestSuite;


public class AllTest {

	public static Test suite() {
		TestSuite suite = new TestSuite(AllTest.class.getName());
		//$JUnit-BEGIN$
		suite.addTestSuite(AppTest.class);
		//$JUnit-END$
		return suite;
	}

}
```

Test Case Example for AppTest

Test cases examine all the aspects including inputs and outputs. It gives the detailed steps that should to be followed during the testing process. Testing process follows only the steps that have been written for each test case.

```
package com.photon.Phresco;

import junit.framework.Test;
import junit.framework.TestCase;
import junit.framework.TestSuite;

/**
 * Unit test for simple App.
 */
public class AppTest
    extends TestCase{

    /**
     * Create the test case
     *
     * @param testName name of the test case
     */
    public AppTest( String testName ) {

        super( testName );
        System.out.println("Printed");
    }

    /**
     * @return the suite of tests being tested
     */
    public static Test suite() {

        return new TestSuite( AppTest.class );
    }

    /**
     * Rigourous Test :-)
     */
    public void testApp() {
        assertTrue( true );
    }
}
```

8.1.1.3 Report generated after execution

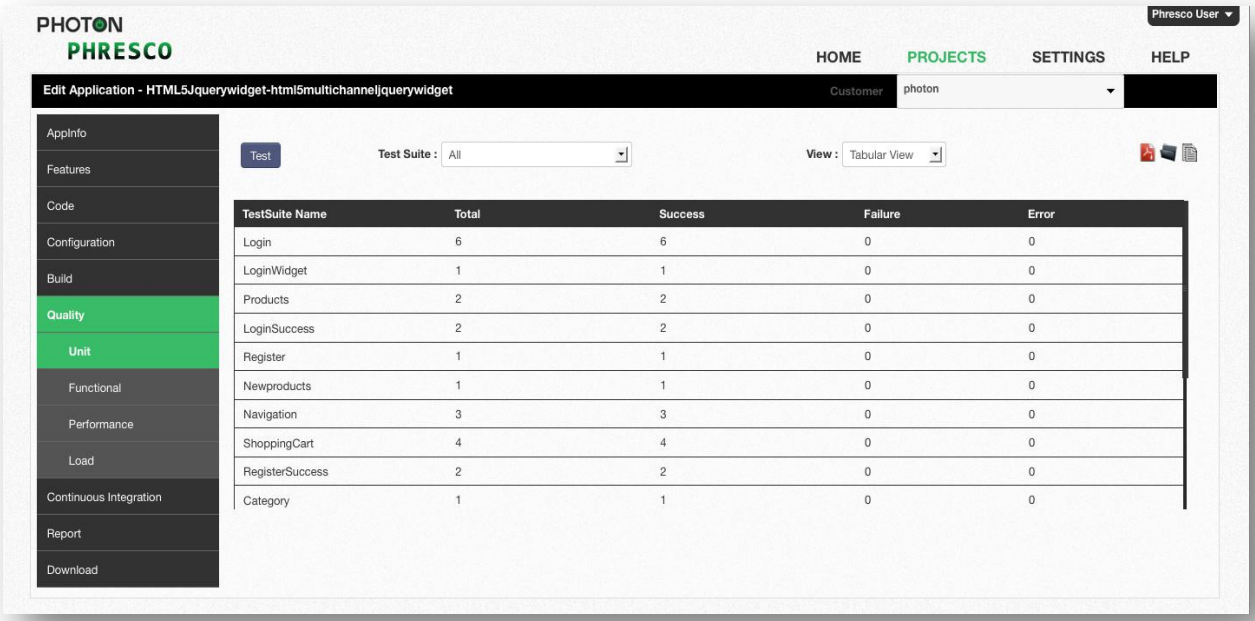

Figure 8-2: HTML5 widget unit test report for all test cases in tabular view

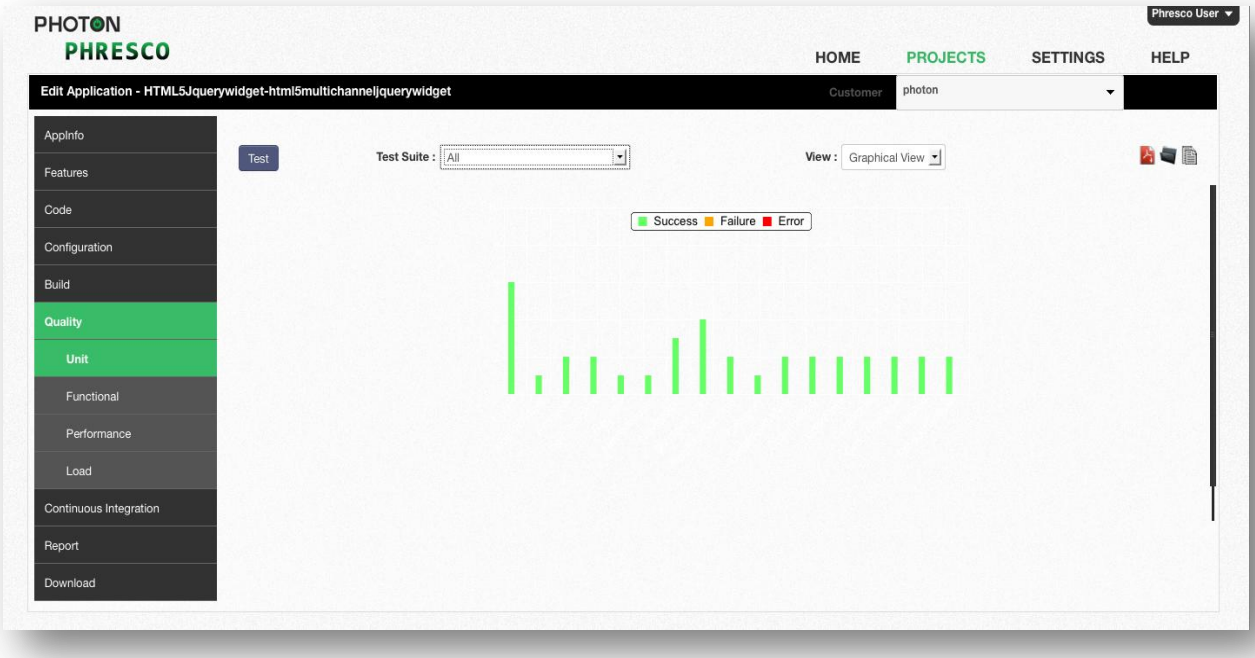

Figure 8-3: HTML5 widget unit test report for all test cases in graphical view

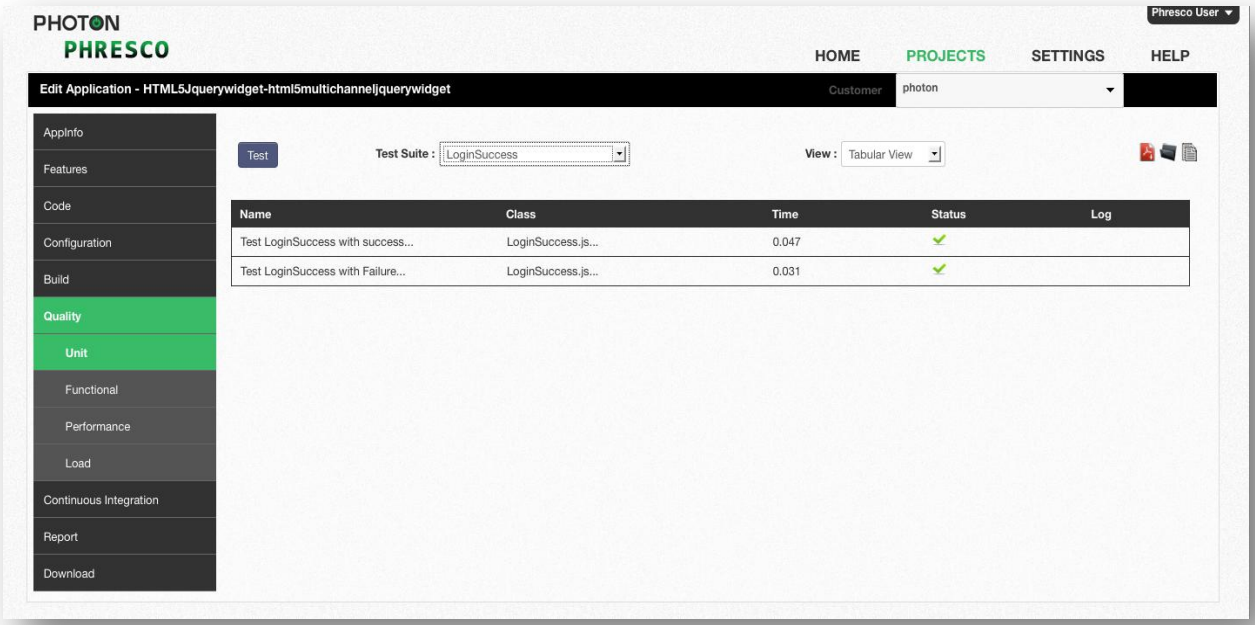

Figure 8-4: HTML5 widget unit test report for single test case in tabular view

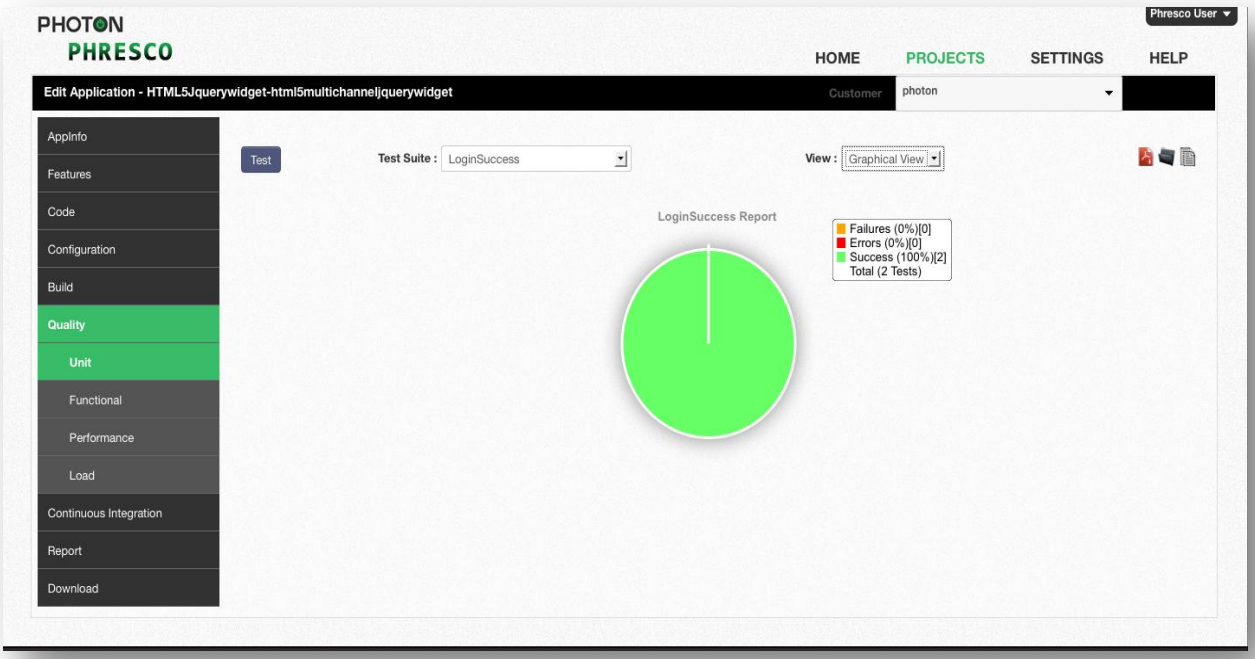

Figure 8-5: HTML5 widget unit test report for single test case in graphical view

8.1.2 Functional Test cases – Selenium Grid

8.1.2.1 Structure of Functional Test in Java for Selenium Grid

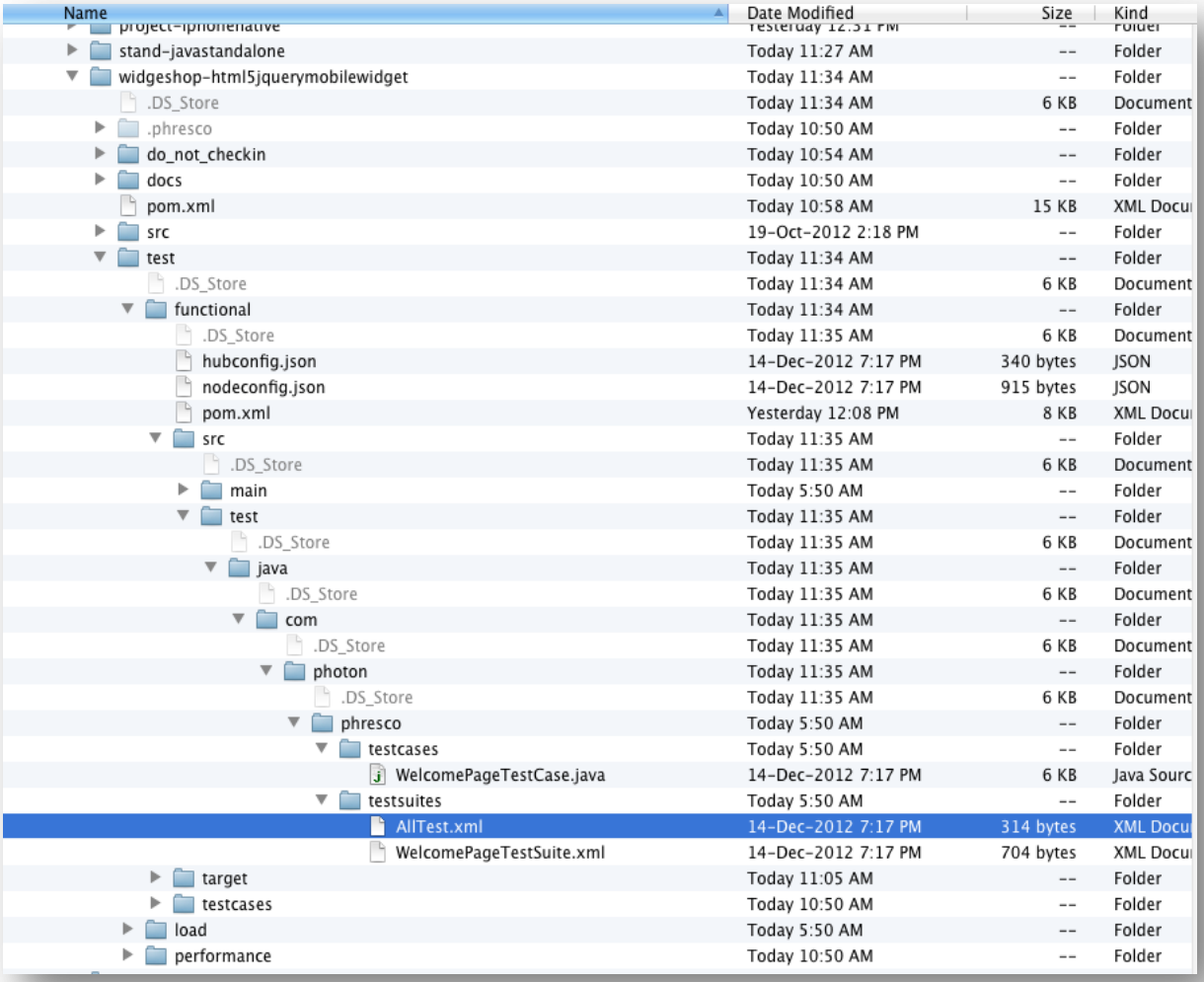

Figure 8.6: Java functional tests structure

a. **AllTest**: This is a root TestNG suite xml that can either call a suite of xmls or individual test cases.

b. **Suite Xml**: This is also a TestNG suite xml which calls the rest of the individual test cases.

c. **Test Cases**: These are individual java classes which perform unique testing scenarios against the application.

8.1.2.2 Existing Test Cases and Suites Out Of the Box in HeliOS

In HeliOS testing Framework a TestNG suite xml is named as “AllTest”.

TestSuite contains a bunch of test cases, each of which performs a unique scenario on the application. Test developers can start writing new suite xml and test cases by following the syntax and structure of HeliOS frameworks out of the box class structure. AllTest xml calls all the suite xmls and the test cases within it. Once suite xmls and test cases are written developers can execute those from HeliOS Framework and can see the report. Following is the AllTest xml file which shows up how to add / include the xmls inside it.

```
<?xml version="1.0" encoding="UTF-8"?>
        <suite name="AllTest"  parallel="tests" thread-count="10">
        <suite-files>
          <suite-file path="./WelcomePageTestSuite.xml" />
        </suite-files>
        </suite>
```

Suite xmls

Suite xml is a collection of test cases that are intended to be used to test a project code to show that it has some specified set of behaviors. A suite xml contains detailed instructions or goals for each collection of test cases. New test cases can also be added by using the same syntax.

```
<?xml version="1.0" encoding="UTF-8"?>
        <suite name="WelcomePageTestSuite" parallel="tests" thread-count="5"
        verbose="3">
        <test name="VerifyWelcomePageOnFirefox" junit="false preserve-order=
        "true">
        parameter name="browser" value="firefox" />
        <parameter name="platform" value="WINDOWS" />
        <classes>
        <class name="com.photon.phresco.testcases.WelcomePageTestCase"/>
        </classes>
        </test>
        </suite>
```

Test cases

A test case is a set of conditions under which a tester will determine whether an application is meeting the requirement. This helps the user to verify a particular functionality during the testing process.

```
package com.photon.phresco.testcases;
import java.io.IOException;
import org.testng.Assert;
import org.testng.Reporter;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Parameters;
import org.testng.annotations.Test;
import com.photon.phresco.Screens.WelcomeScreen;
import com.photon.phresco.uiconstants.PhrescoUiConstants;
import com.photon.phresco.uiconstants.UIConstants;
import com.photon.phresco.uiconstants.WidgetData;
```

```
public class WelcomePageTestCase {
private  UIConstants uiConstants;
private  PhrescoUiConstants phrescoUIConstants;
private  WelcomeScreen welcomeScreen;
private  String methodName;
private  String selectedBrowser;
private  WidgetData WidgetConstants;
 // private Log log = LogFactory.getLog(getClass());
 @Parameters(value = { "browser", "platform" })
 @BeforeTest
 public void setUp(String browser, String platform) throws Exception {
  try {
            phrescoUIConstants = new PhrescoUiConstants();
            uiConstants = new UIConstants();
            WidgetConstants = new WidgetData();
             String selectedBrowser = browser;
             String selectedPlatform = platform;

methodName=Thread.currentThread().getStackTrace()[1].getMethodName(
);
    Reporter.log("Selected Browser to execute testcases--->>"+
selectedBrowser);
                String applicationURL = phrescoUIConstants.PROTOCOL +
"://"
```

```
                + phrescoUIConstants.HOST + ":" + phrescoUIConstants.PORT
                        + "/";
 welcomeScreen = new WelcomeScreen(selectedBrowser, selectedPlatform,
applicationURL, phrescoUIConstants.CONTEXT, WidgetConstants,
uiConstants);
                } catch (Exception exception) {exception.printStackTrace();
        }
    }

@Test
 Public void testWelcomePageScreen() throws InterruptedException,
IOException, Exception {
  try {
     System.out.println("---------testWelcomePageScreen-------------");
      Assert.assertNotNull(welcomeScreen);
     Thread.sleep(1000);
     } catch (Exception t) {
     t.printStackTrace();
                }
     }


@Test
 public void testToVerifyTheAudioDevicesAddToCart() throws
InterruptedException, IOException, Exception {
  try {
 System.out.println("---------testToVerifyTheAudioDevicesAddToCart()-----
------");
      welcomeScreen.AudioDevices(methodName);
      welcomeScreen.billingInfo(methodName);
       } catch (Exception t) {
        t.printStackTrace();
       }
      }

  @AfterTest
     public  void tearDown() {
     welcomeScreen.closeBrowser();
   }
}
```

8.1.2.3 To Add A New Test Suite in Alltest Xml

User can add a new test suite to the AllTest xml as follows.

Example

```
<?xml version="1.0" encoding="UTF-8"?>
   <suite name="AllTest"  parallel="tests" thread-count="10">
          <suite-files>
          <suite-file path="./WelcomePageTestSuite.xml" />
          <suite-file path="./LoginPageTestSuite.xml" />
          </suite-files>
   </suite>
```

8.1.2.4 To Add New Test Case in Test Suite Xml

You can add a new test case to the Test suite using the following method. Here is an example for adding a new test case in the name “CamerasAddcart

```
    <suite name="WelcomePageTestSuite" parallel="test” thread-count="5" verbose="3">
       <test name="VerifyWelcomePageOnFirefox" junit="false" preserve-order="true">
         <parameter name="browser" value="firefox" />
         <parameter name="platform" value="WINDOWS" />
         <classes>
          <class name="com.photon.phresco.testcases.WelcomePageTestCase" />
           <class name="com.photon.phresco.testcases.LoginPageTestCase" />
         </classes>
        </test>
      </suite>
```

8.1.2.5 Report generated after execution

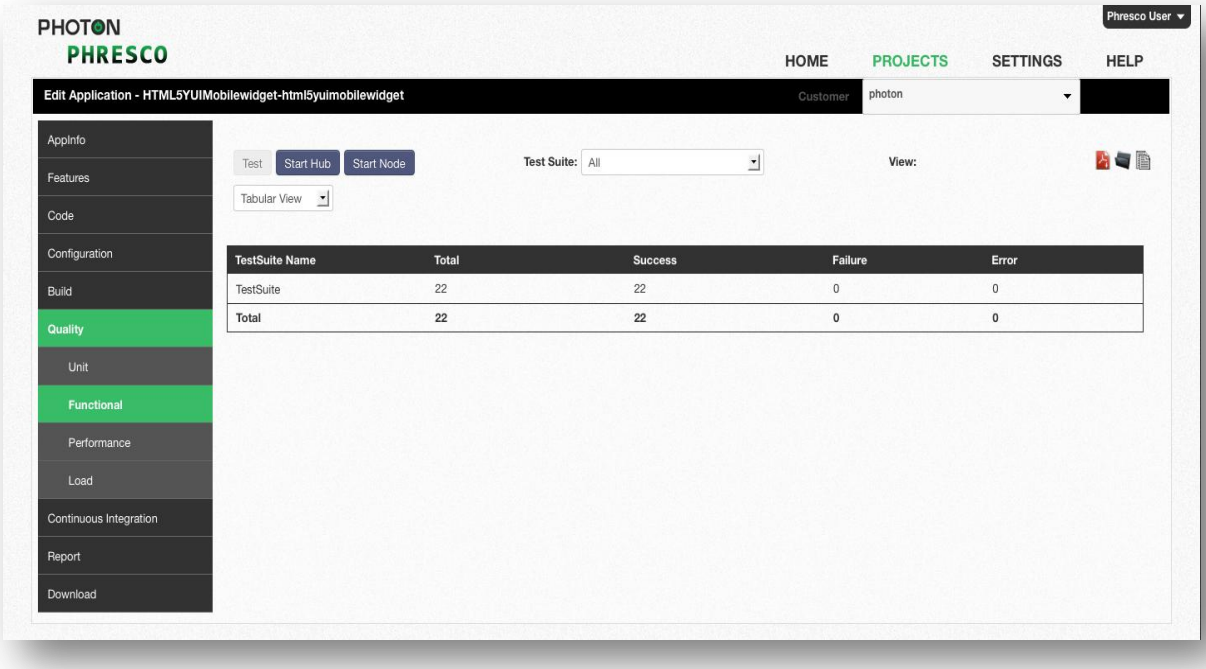

Figure 8.7: HTML5 widget functional test report for all test cases in tabular view

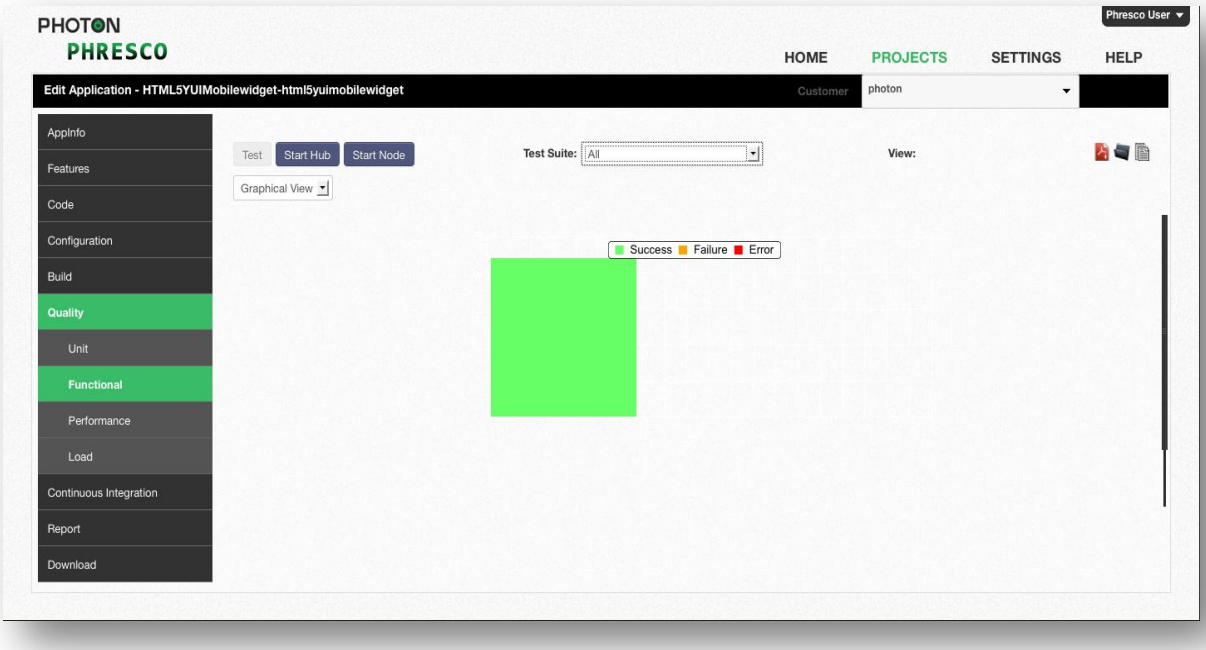

Figure 8.8: HTML5 widget functional test report for all test cases in graphical view

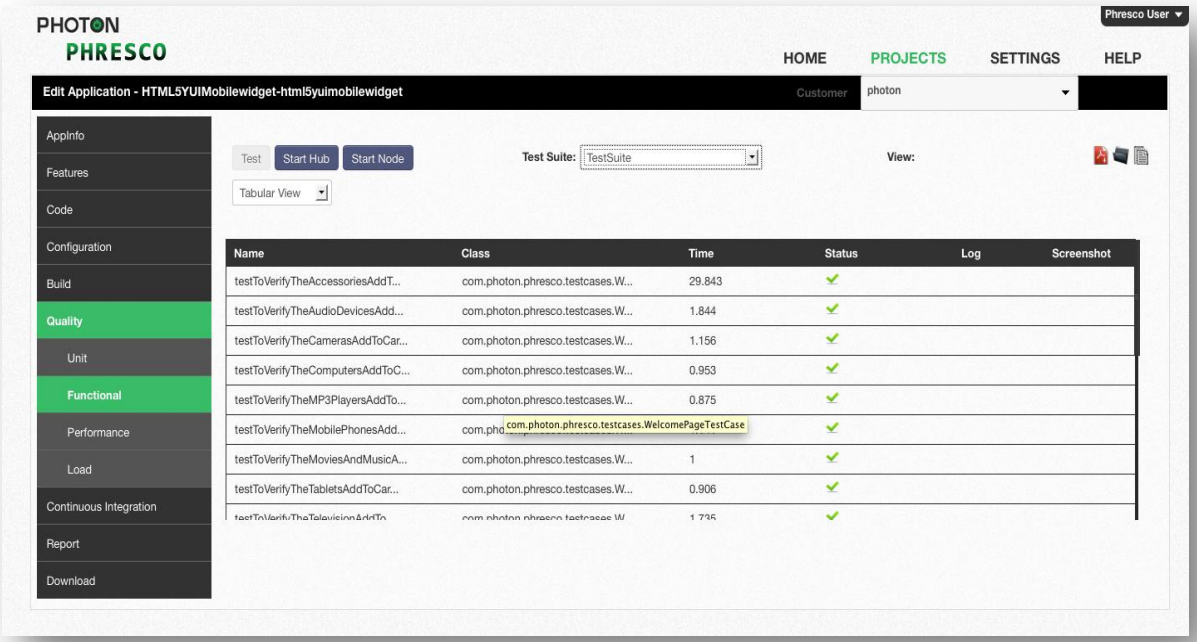

Figure 8.9: HTML5 widget functional test report for single test case in tabular view

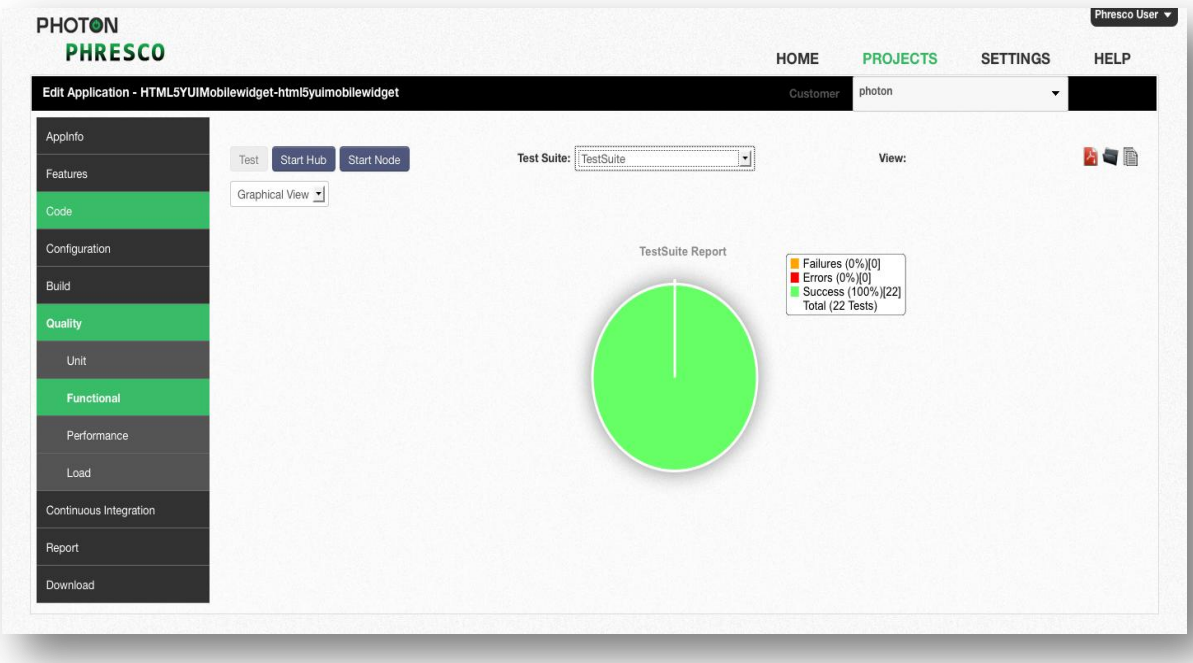

Figure 8.10: HTML5 widget functional test report for single test case in graphical view

Note

- HeliOS supports Functional Tests for Widgets, PHP Technologies, Nodejs Webservices, Java Webservices in Selenium Grid format.
- Refer section 8.1.2 for the Functional Test case structure which is similar to Selenium Grid Functional Test case structure.
- In addition to this,HeliOS also supports Cucumber for Widgets.

8.2 Functional Test cases for Selenium Webdriver

8.2.1.1 Structure of Functional Test in Java

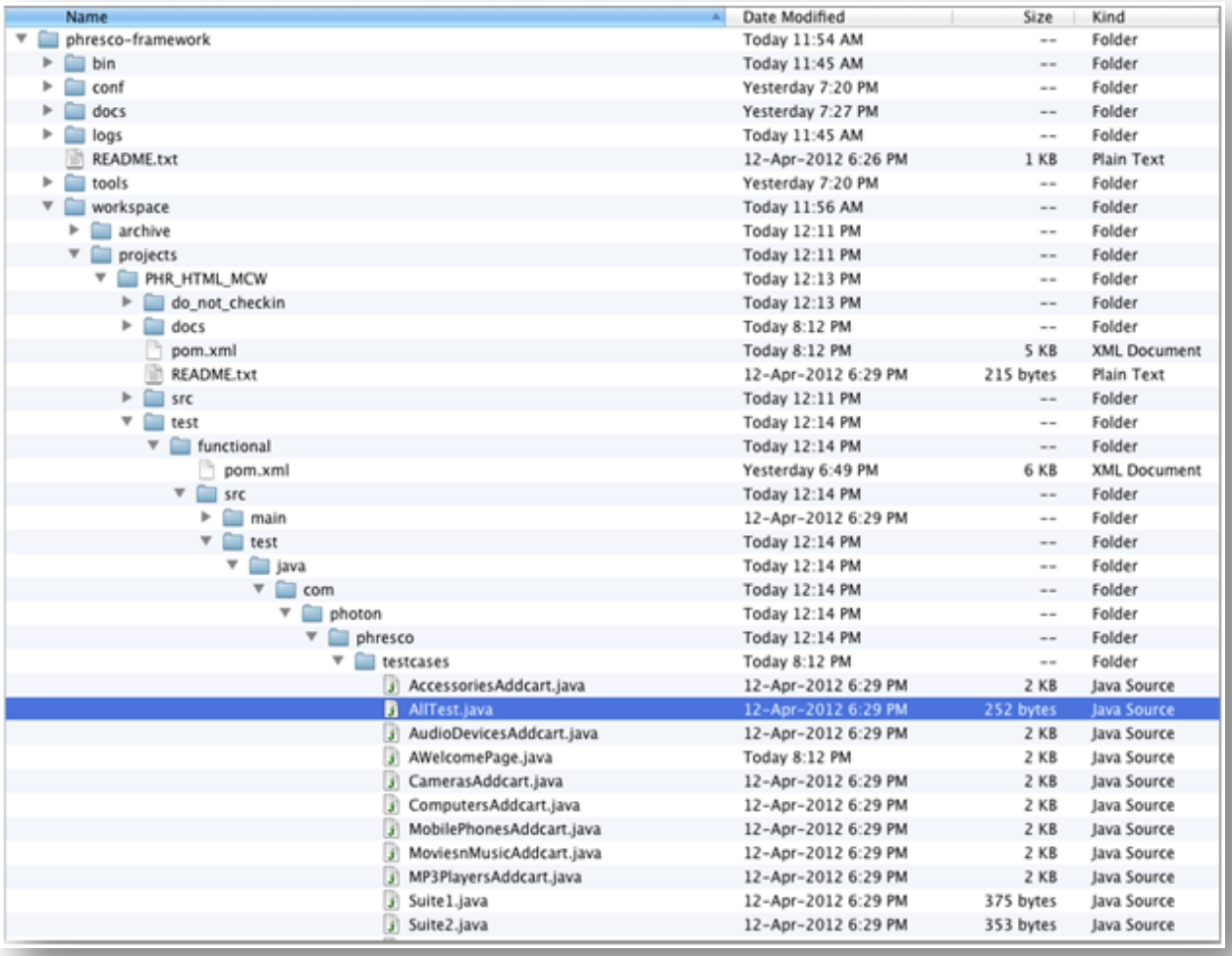

Figure 8-11: Java functional tests structure

a. AllTest: This is a root JUnit suite class that can either call a suite of classes or individual test cases.

b. Suite Class: This is also a JUnit suite class which calls the rest of the individual test cases.

c. Test Cases: These are individual java classes which perform unique testing scenarios against the application.

8.2.1.2 Existing Test Cases and Suites Out Of the Box in HeliOS

In HeliOS testing Framework a JUnit suite class is named as "AllTest".

The TestSuite contains a bunch of test cases, each of which performs a unique scenario on the application. Test developers can start writing new suite classes and test cases by following the syntax and structure of HeliOS frameworks out of the box class structure. AllTest class calls all the suite classes and the test cases within it. Once suite classes and test cases are written developers can execute those from HeliOS Framework and can see the report. Following is the AllTest file which shows up how to add / include the class inside it.

```
package com.photon.Phresco.testcases

import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)
@SuiteClasses({Suite1.class,Suite2.class})
public class AllTest {
}
```

Suite class

Suite class is a collection of test cases that are intended to be used to test a project code to show that it has some specified set of behaviors. A suite class contains detailed instructions or goals for each collection of test cases. New test cases can also be added by using the same syntax.

```
package com.photon.Phresco.testcases;
import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)
@SuiteClasses({ WelcomePage.class,TeleVisionAddcart.class,
                ComputersAddcart.class,
MobilePhonesAddcart.class,AudioDevicesAddcart.class,
CamerasAddcart.class

})
public class Suite1 {

}
```

Test cases

A test case is a set of conditions under which a tester will determine whether an application is meeting the requirement. This helps the user to verify a particular functionality during the testing process. Elements can be identified and screen shots can be captured when the test fails.

```
package com.photon.Phresco.testcases;

import java.io.IOException;

import junit.framework.TestCase;

import org.junit.Test;
//import static org.testng.AssertJUnit.*;
import org.openqa.selenium.server.SeleniumServer;

import com.photon.Phresco.Screens.MenuScreen;
import com.photon.Phresco.Screens.WelcomeScreen;
import com.photon.Phresco.selenium.report.Reporter;
import com.thoughtworks.selenium.Selenium;
import com.photon.Phresco.uiconstants.PhrescoUiConstants;

public class AWelcomePage extends TestCase {
private SeleniumServer serv;
```

```
 protected Selenium selenium;
 private PhrescoUiConstants phrsc;
 private int SELENIUM_PORT;
 private String browserAppends;

 @Test
 public void testWel() throws InterruptedException, IOException, Exception
{
        try {
                phrsc = new PhrescoUiConstants();
                String serverURL = phrsc.PROTOCOL + "://"
                                + phrsc.HOST + ":"
                                + phrsc.PORT + "/";
                browserAppends = "*" + phrsc.BROWSER;
                assertNotNull("Browser name should not be
null",browserAppends);
```

```
                SELENIUM_PORT = Integer.parseInt(phrsc.SERVER_PORT);
                assertNotNull("selenium-port number should not be null",
                        SELENIUM_PORT);
                WelcomeScreen wel=new
WelcomeScreen(phrsc.SERVER_HOST, SELENIUM_PORT,
                        browserAppends, serverURL, phrsc.SPEED,
                        phrsc.CONTEXT );
                assertNotNull(wel);
                MenuScreen menu = wel.menuScreen();
        assertNotNull(menu);

        } catch (Exception t) {
                t.printStackTrace();
                System.out.println("ScreenCaptured");
                selenium.captureEntirePageScreenshot("\\WelPageFails.png",
                        "background=#CCFFDD");
        }
}

@Override
public void setUp() throws Exception {

        serv = new SeleniumServer();
        try {
                serv.start();
        } catch (Exception e) {
                clean();
                throw e;
        }
}

@Override
public void tearDown() {
        clean();
}

private void clean() {
        if (serv != null) {
                serv.stop();
        }
        if (selenium != null) {
                selenium.stop();
        }
```

```
    }

}
```

8.2.1.3 To Add A New Test Suite in Alltest Class

You can add a new test suite to the AllTest class as follows.

Example

```
package com.photon.Phresco.testcases;

import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)
@SuiteClasses({Suite1.class,Suite2.class})
public class AllTest {
}
```

8.2.1.4 To Add New Test Case in Test Suite

You can add a new test case to the Test suite using the following method. Here is an example for creating a new test case in the name “CamerasAddcart”

```
package com.photon.Phresco.testcases;

import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)
@SuiteClasses({
WelcomePage.class,TeleVisionAddcart.class,ComputersAddcart.class,
                    MobilePhonesAddcart.class,AudioDevicesAddcart.class,
                    CamerasAddcart.class
```

```
                    })
public class Suite1 {

}
```

8.2.1.5 Report generated after execution

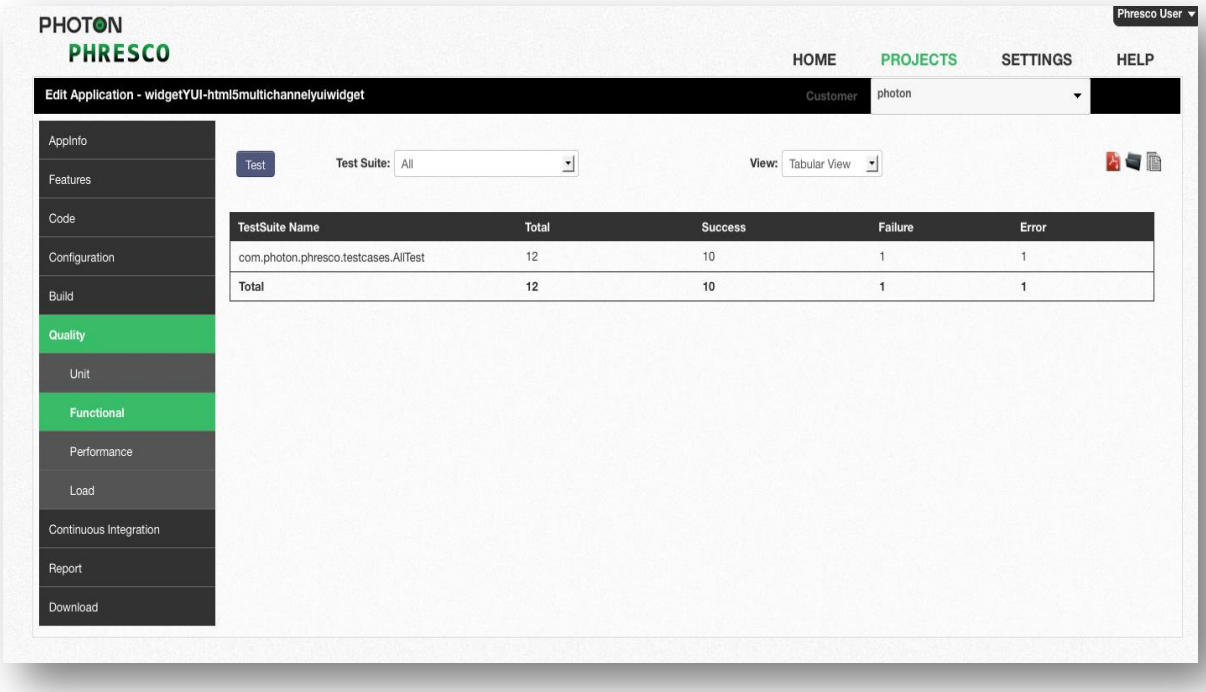

Figure 8-12: HTML5 widget functional test report for all test cases in tabular view

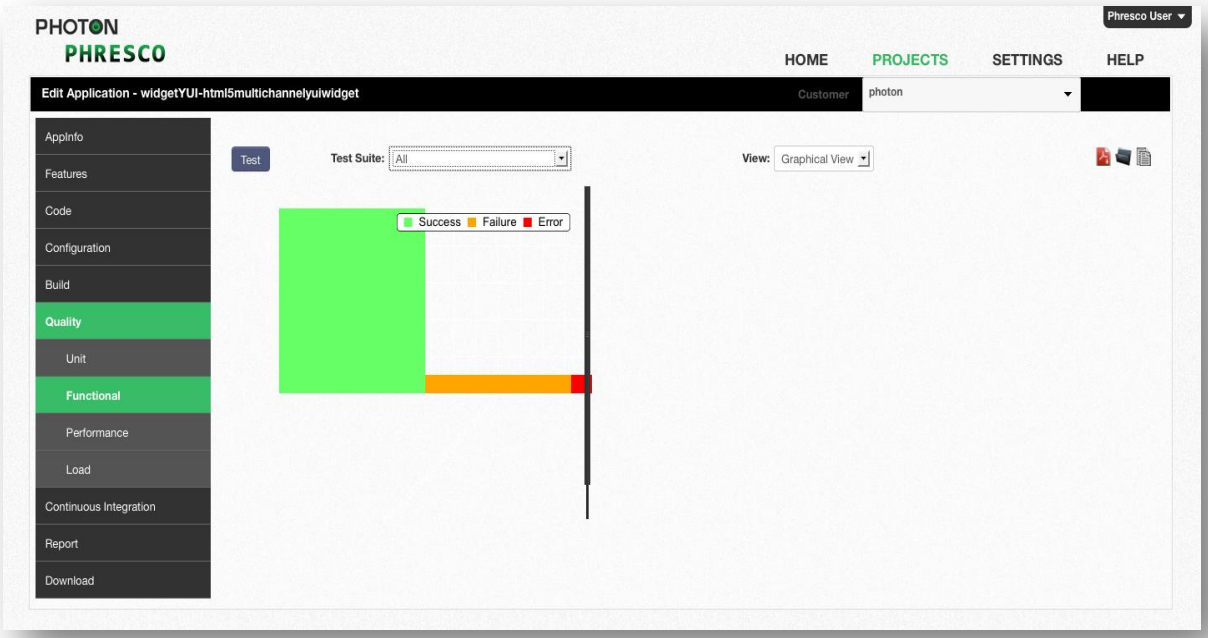

Figure 8-13: ***HTML5 widget functional test report for all test cases in graphical view***

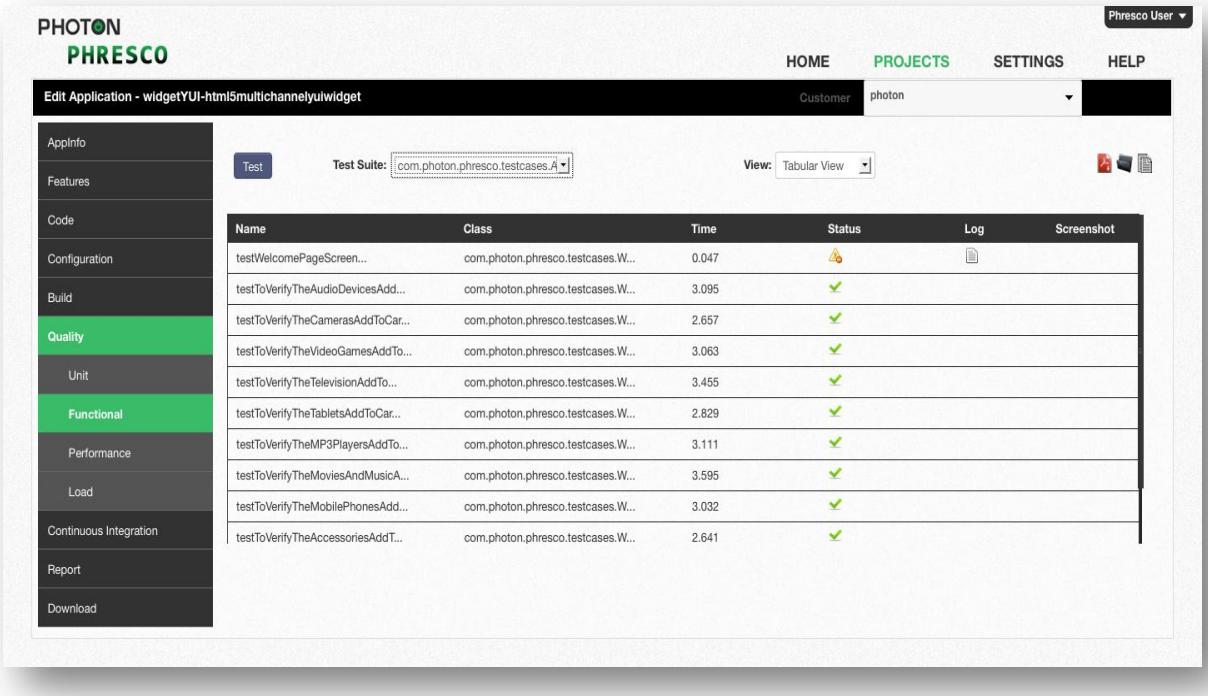

Figure 8-14: ***HTML5 widget functional test report for single test case in tabular view***

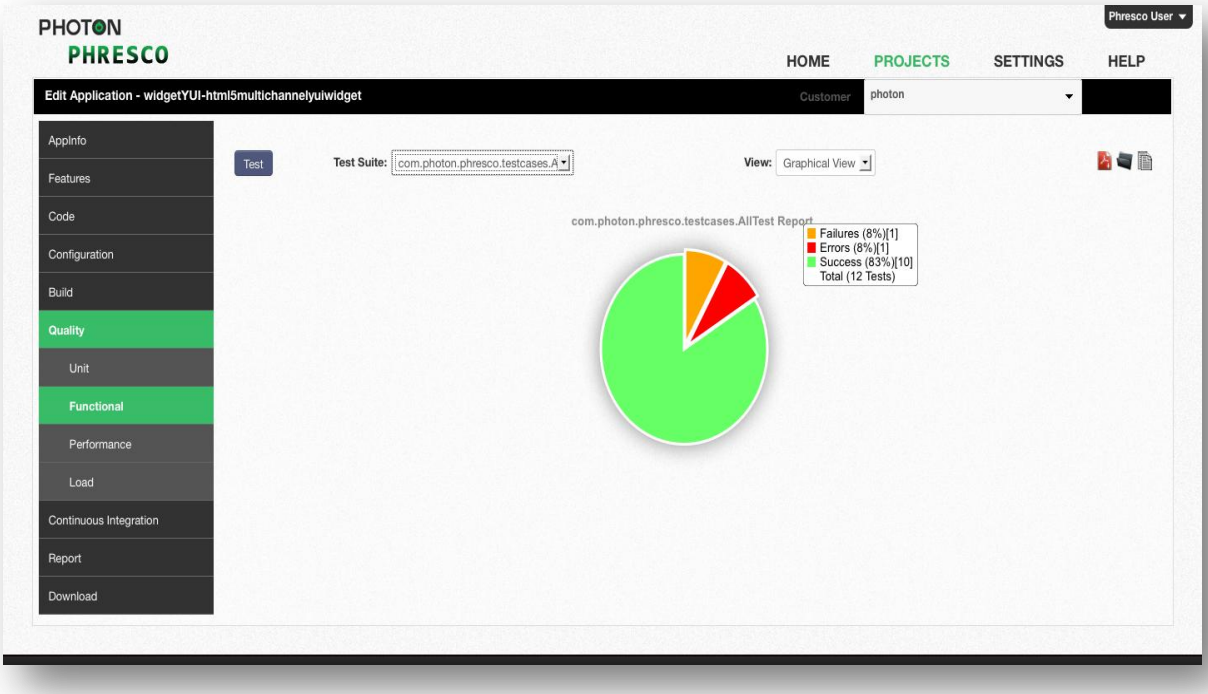

Figure 8-15: ***HTML5 widget functional test report for single test case in graphical view***

8.3 Test Cases for Php Technology

Selenium web driver is used to execute the test cases for Php, Drupal, and WordPress technologies in the latest version.

8.3.1 Unit Test Cases

8.3.1.1 Structure of Alltest and Test Cases

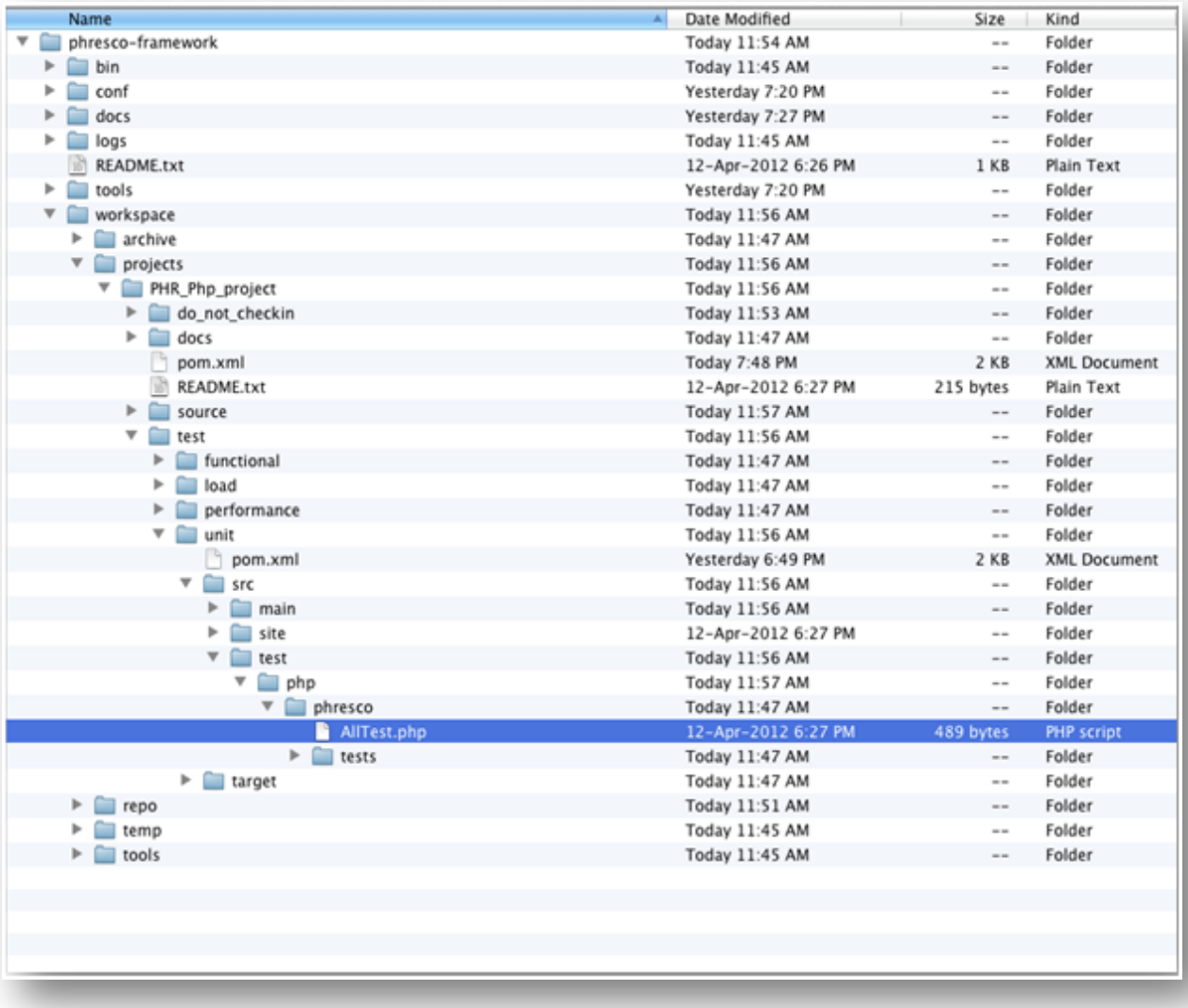

Figure 8-66: PHP unit tests structure

a. **AllTest:** It is the root file that carries all the php suite file to initiate the testing process. Each test case can be called separately to run the unit test.

b. **Test cases**: These are individual test classes which perform unique testing scenarios against the application.

8.3.1.2 Existing Test Cases and Suites Out Of the Box in HeliOS

AllTest for PHP

AllTest contains a bunch of test cases, each of which performs a unique scenario on the application. Test developers can start writing new suite classes and test cases by following the syntax and structure of HeliOS framework's out of the box class structure. AllTest class calls all the suite classes and the test cases within it. Once suite classes and test cases are written developers can execute those from HeliOS Framework and can see the report. Following is the AllTest file which shows up how to add / include the class inside it.

```
<? php

require_once 'tests/UsernameValidation.php';
require_once 'tests/DateValidation.php';

class AllTest extends PHPUnit_Framework_TestSuite{

protected function setUp(){

}
public static function suite(){
$testSuite = new AllTest('Phpunittest');
$testSuite->addTest(new UsernameValidation("testValidation"));
$testSuite->addTest(new DateValidation("testDate"));
return $testSuite;
 }
 protected function tearDown(){

        }

}
```

Test case

A test case is a set of conditions under which a tester will determine whether an application is meeting the requirement. This helps the user to verify a particular functionality during the testing process.

This test case is written for testing the characters of the username to be entered manually. The characters are defined in the base screen and only if it matches with manually entered characters, the unit test case will pass. Testing the username is one unit test case and there can be 'n' number of test cases.

```
<?php

require_once 'PHPUnit/Framework.php';
require_once 'Phresco/tests/BaseScreen.php';
class UsernameValidation extends PHPUnit_Framework_TestCase {
 public function setUp() {

        $this->objValidator = new BaseScreen;

    }
    public function testValidation() {

        $this->assertEquals(true,
        $this->objValidator->check_username('jhonson'));
    }
}
?>
```

8.3.1.3 To Add New Test Case in Alltest

You can add new test cases to the AllTest. An example for creating a new test case is given below.

```
 public static function suite(){
 $testSuite = new AllTest('Phpunittest');
 $testSuite->addTest(new UsernameValidation("testValidation"));
 $testSuite->addTest(new DateValidation("testDate"));
 $testSuite->addTest(new EmailVerification("testEmail"));
```

8.3.1.4 Report generated after execution

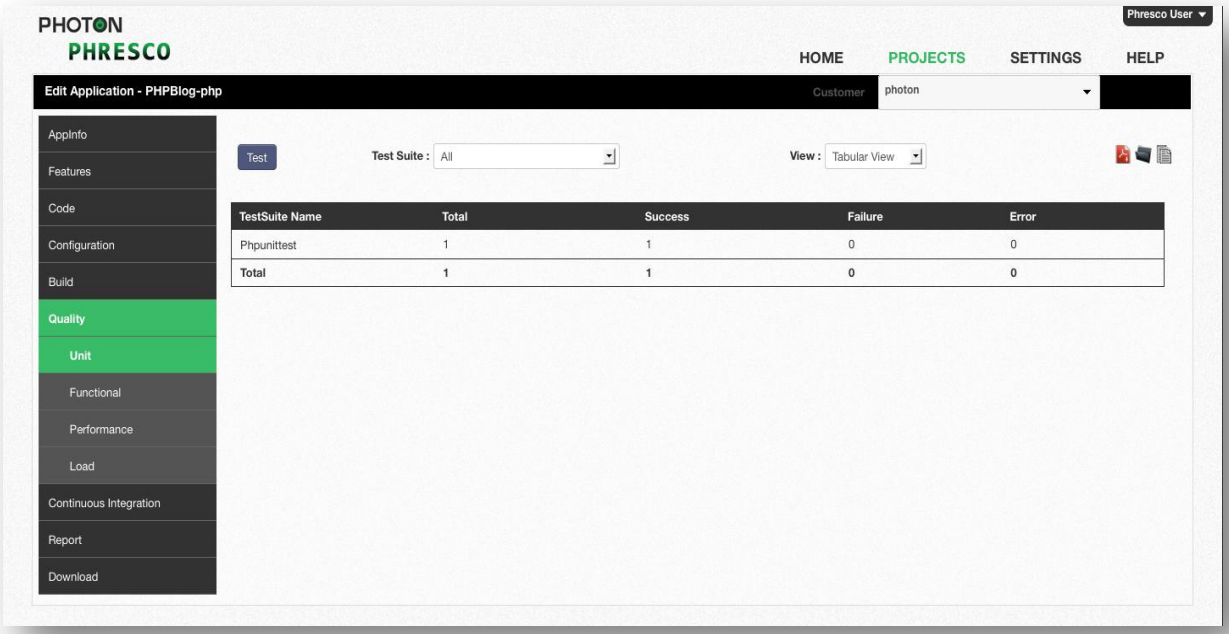

Figure 8-77: PHP unit test report for all test cases in tabular view

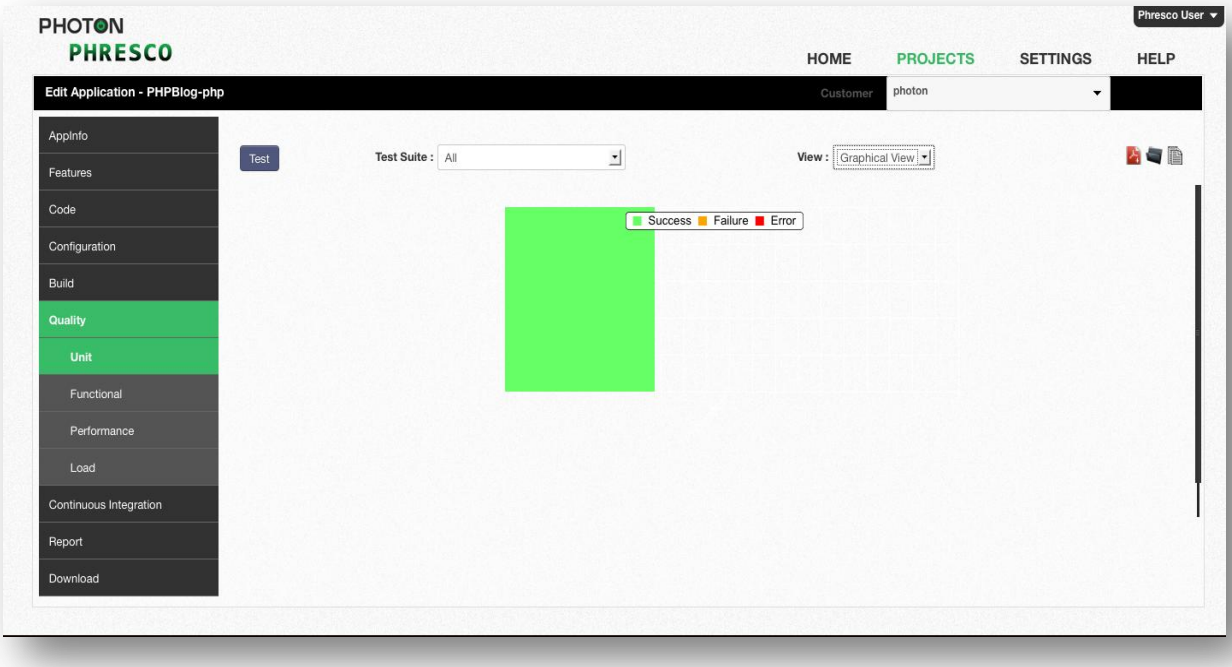

Figure 8-88: PHP unit test report for all test cases in graphical view

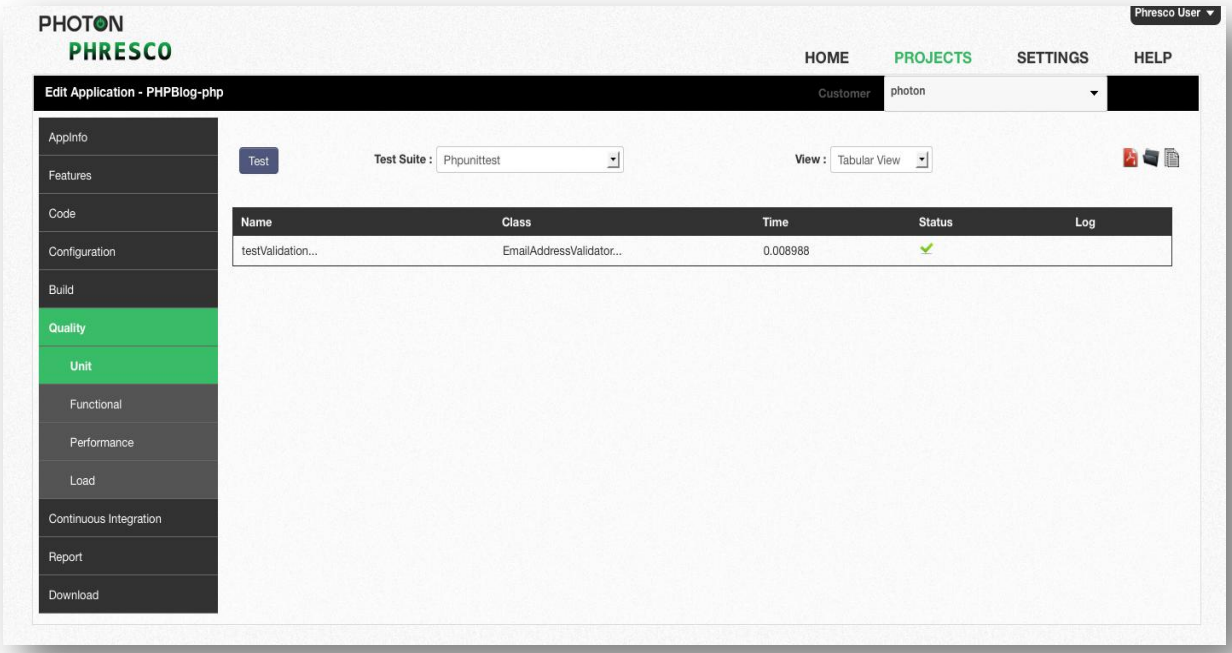

Figure 8-99: PHP unit test report for single test case in tabular view

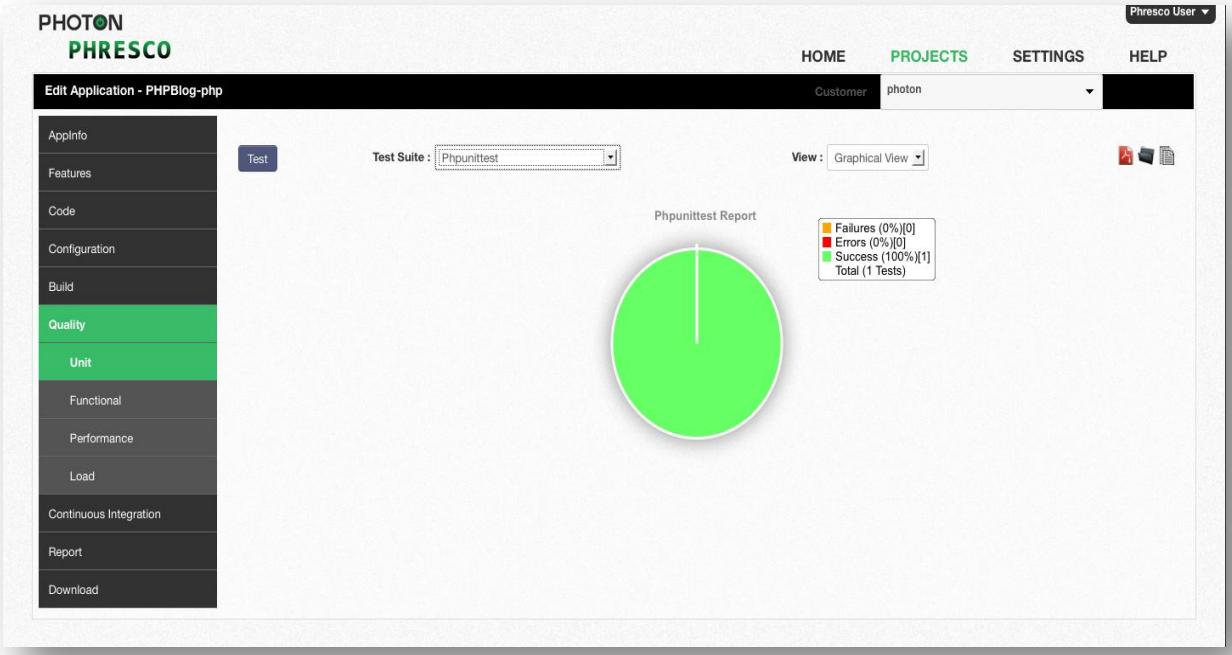

Figure 8-20: PHP unit test report for single test case in graphical view

8.3.2 Functional Test Case for Webdriver

8.3.2.1 Structure of Test Suites and Test Case

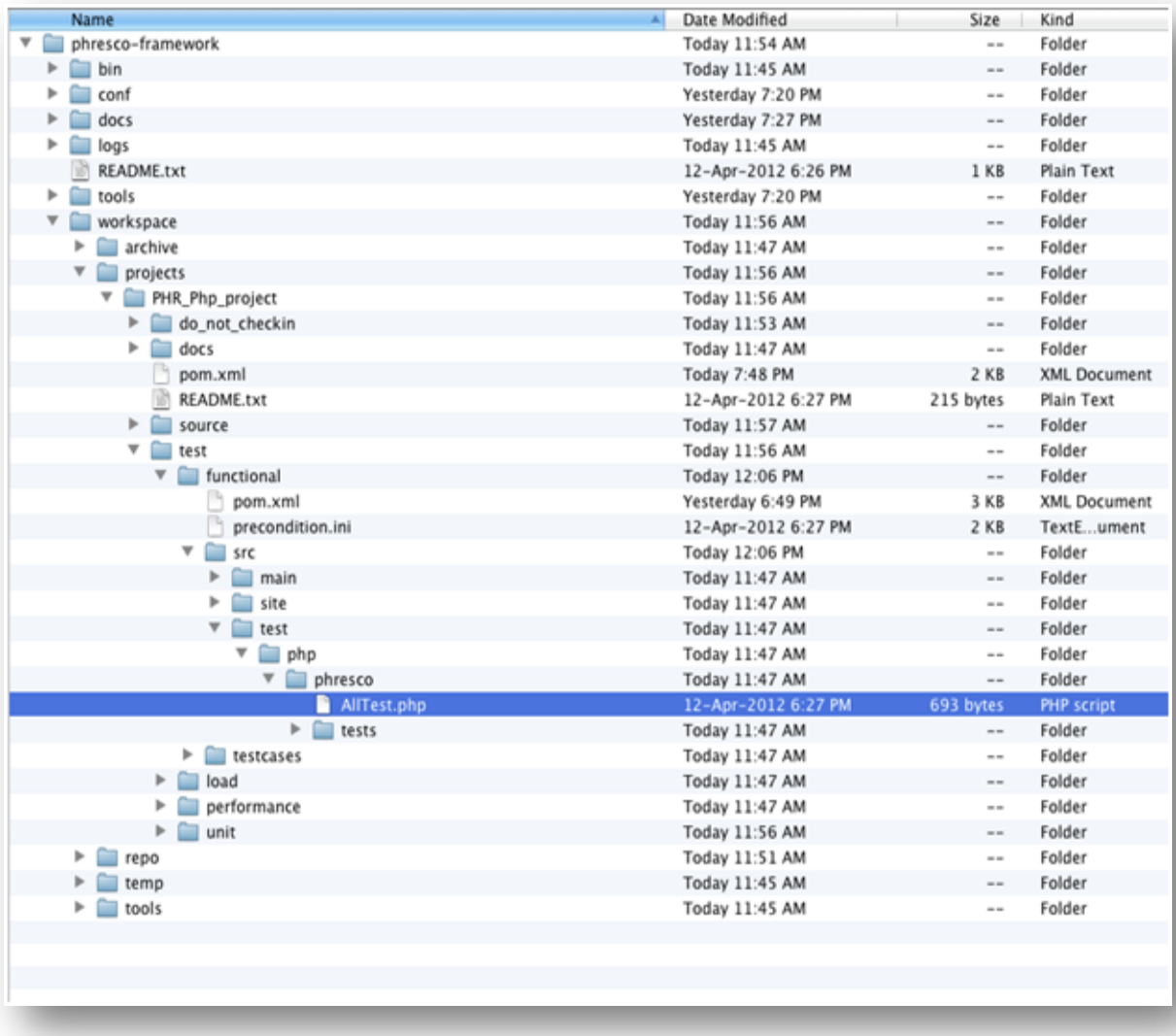

Figure 8-21: PHP functional tests structure

a. **AllTest**: It is the root file that carries all the test suites to initiate the testing.

b. **Test suite**: All the Test suites should be incorporated in the **AllTest**

c. **Test case**: These are individual test classes which perform unique testing scenarios against the application.

8.3.2.2 Existing Test Cases and Suites Out Of the Box in HeliOS

Alltest for Php

AllTest contains a bunch of test cases, each of which performs a unique scenario on the application. Test developers can start writing new suite classes and test cases by following the syntax and structure of HeliOS framework's out of the box class structure. AllTest class calls all the suite classes and the test cases within it. Once suite classes and test cases are written developers can execute those from HeliOS Framework and can see the report. Following is the AllTest file which shows up how to add / include the class inside it.

```
require_once 'tests/UserModules.php';
require_once 'tests/SearchModules.php';
require_once 'tests/DbUpdateModules.php';
require_once 'tests/DbDeleteModules.php';

class AllTest extends PHPUnit_Framework_TestSuite
{
protected function setUp(){
parent::setUp();
}
public static function suite(){

$suite = new AllTest();
$suite->setName('AllTestsuite');
$suite->addTest(UserModules::suite());
$suite->addTest(SearchModules::suite());
$suite->addTest(DbUpdateModules::suite());
$suite->addTest(DbDeleteModules::suite());
 return $suite;
}
protected function tearDown(){
parent::tearDown();
                }
        }
```

Test Suites Example for Usermodules

Test suite is a collection of test cases that are intended to be used to test a project code to show that it has some specified set of behaviors. A test suite contains detailed instructions or goals for each collection of test cases. New test cases can also be added by using the following syntax.

```
require_once 'Register_NewUser.php';
require_once 'LoginLogout_User.php';
require_once 'Account_Update.php';

class UserModules extends PHPUnit_Framework_TestSuite
        {
protected function setUp(){
parent::setUp();
                }
public static function suite(){

$testSuite = new UserModules();
$testSuite->setName('UserModules');
$testSuite->addTestSuite('Register_NewUser');
$testSuite->addTestSuite('LoginLogout_User');
$testSuite->addTestSuite('Account_Update');
return $testSuite;
            }
protected function tearDown(){

                }
        }
```

Test Case Example for Testregistration

Test cases examine all the aspects including inputs and outputs. It gives the detailed steps that should to be followed during the testing process. Testing process follows only the steps that have been written for each test case. It also helps in identifying an element and capturing screen shots when the test fails.

```
require_once 'PhpCommonFun.php';
class Register_NewUser extends PhpCommonFun
{
 protected function setUp(){

parent::setUp();
            }
public function testRegisters(){
parent::Browser();
$testcases = __FUNCTION__;
$doc = new DOMDocument();

$doc->load('test-classes/Phresco/tests/phpsetting.xml');
$users = $doc->getElementsByTagName("register");
foreach( $users as $register ){
$names = $register->getElementsByTagName("username");
$name = $names->item(0)->nodeValue;
$emails = $register->getElementsByTagName("email");
$email = $emails->item(0)->nodeValue;
$passwords = $register->getElementsByTagName("password");
$password = $passwords->item(0)->nodeValue;
        }
        $this->getElement(PHP_REG_LINK,$testcases);
        $this->clickandLoad(PHP_REG_LINK);
        $this->getElement(PHP_REG_UNAME_TBOX,$testcases);
        $this->type(PHP_REG_UNAME_TBOX,$name);
        $this->getElement(PHP_REG_EMAIL_TBOX,$testcases);
        $this->type(PHP_REG_EMAIL_TBOX,$email);
        $this->getElement(PHP_REG_PASS_TBOX,$testcases);
        $this->type(PHP_REG_PASS_TBOX,$password);
        $this->getElement(PHP_REG_SUBMIT,$testcases);
        $this->submit(PHP_REG_SUBMIT,$testcases);
        try{
                $this->assertTrue($this->isTextPresent(PHP_REG_MSG));
                }
                catch (PHPUnit_Framework_AssertionFailedError $e) {
                        $this->doCreateScreenShot(__FUNCTION__);
                        }
    }
                public function tearDown(){
                        $this->closeWindow();
                }
```

8.3.2.3 To Add New Test Suite in Alltest

An example for creating new test suite in the name 'DbDeleteModules'.

```
$suite = new AllTest();
$suite->setName('AllTestsuite');
$suite->addTest(UserModules::suite());
$suite->addTest(SearchModules::suite());
$suite->addTest(DbUpdateModules::suite());
$suite->addTest(DbDeleteModules::suite());
```

8.3.2.4 To Add New Test Case in Test Suite

An example for creating a new test case in the name 'testAccountUpdate'

```
$testSuite = new UserModules();
$testSuite->setName('UserModules');
$testSuite->addTestSuite('Register_NewUser');
$testSuite->addTestSuite('LoginLogout_User');
$testSuite->addTestSuite('Account_Update');
```

Report generated after execution

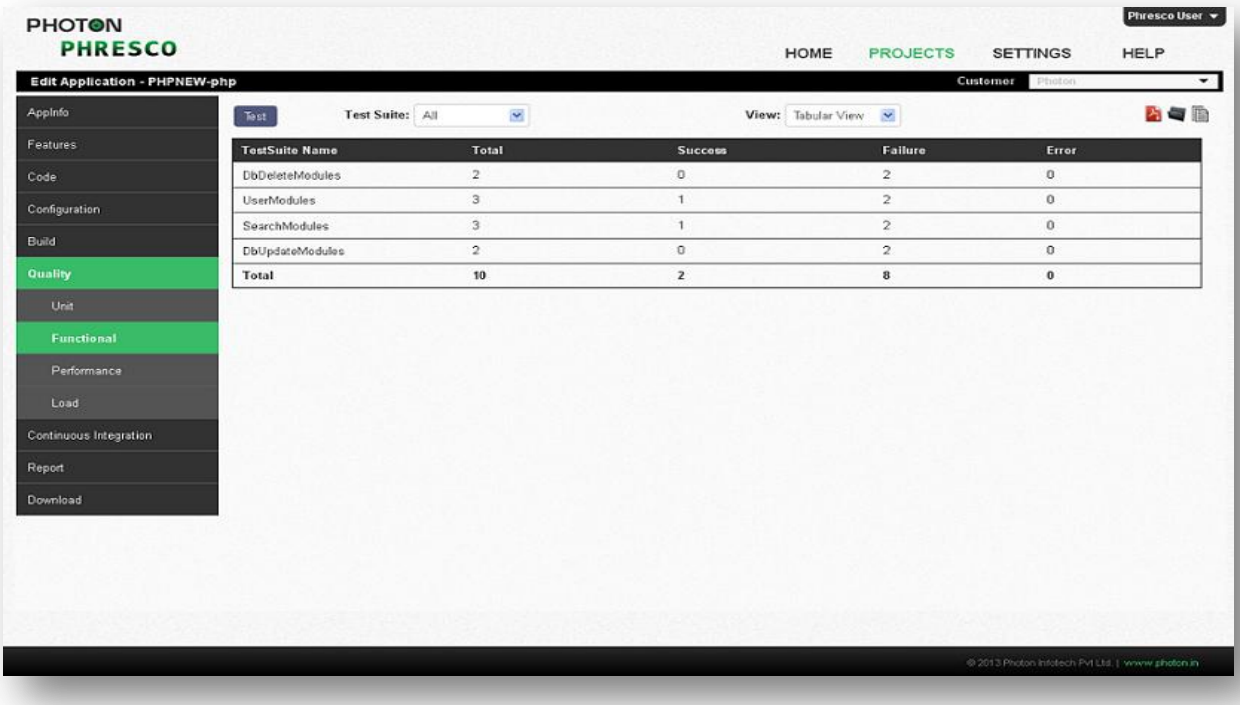

Figure 8-22: PHP functional test report for all test cases in tabular view

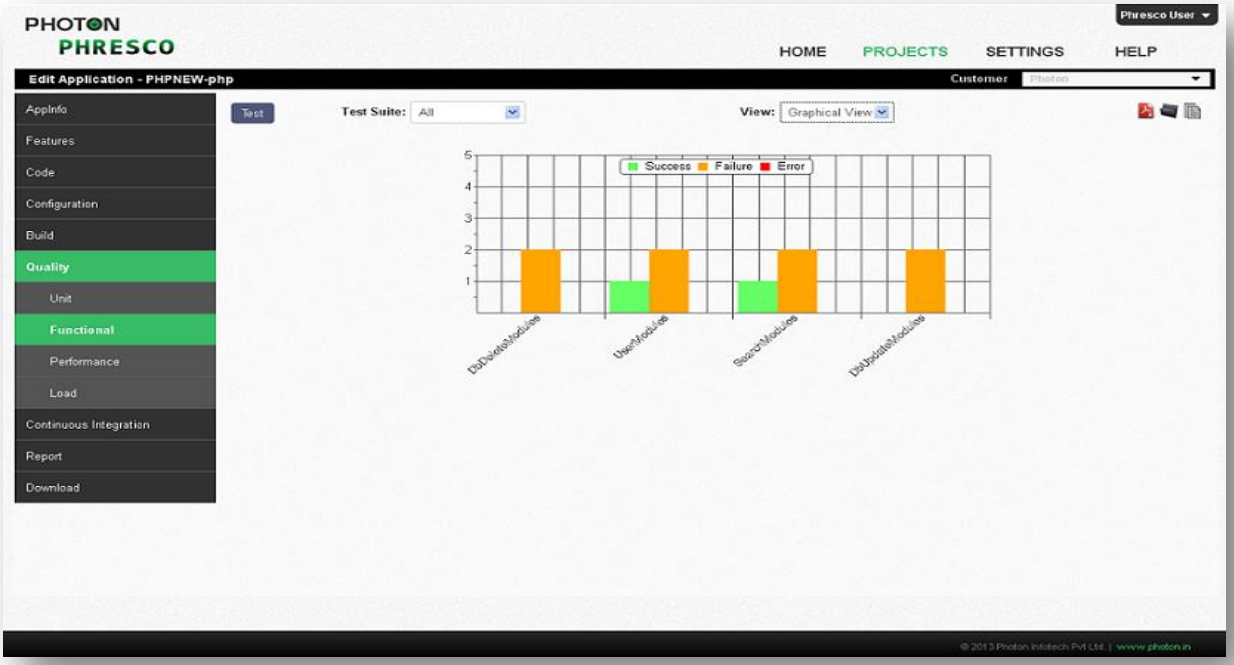

Figure 8-23: PHP functional test report for all test cases in graphical view

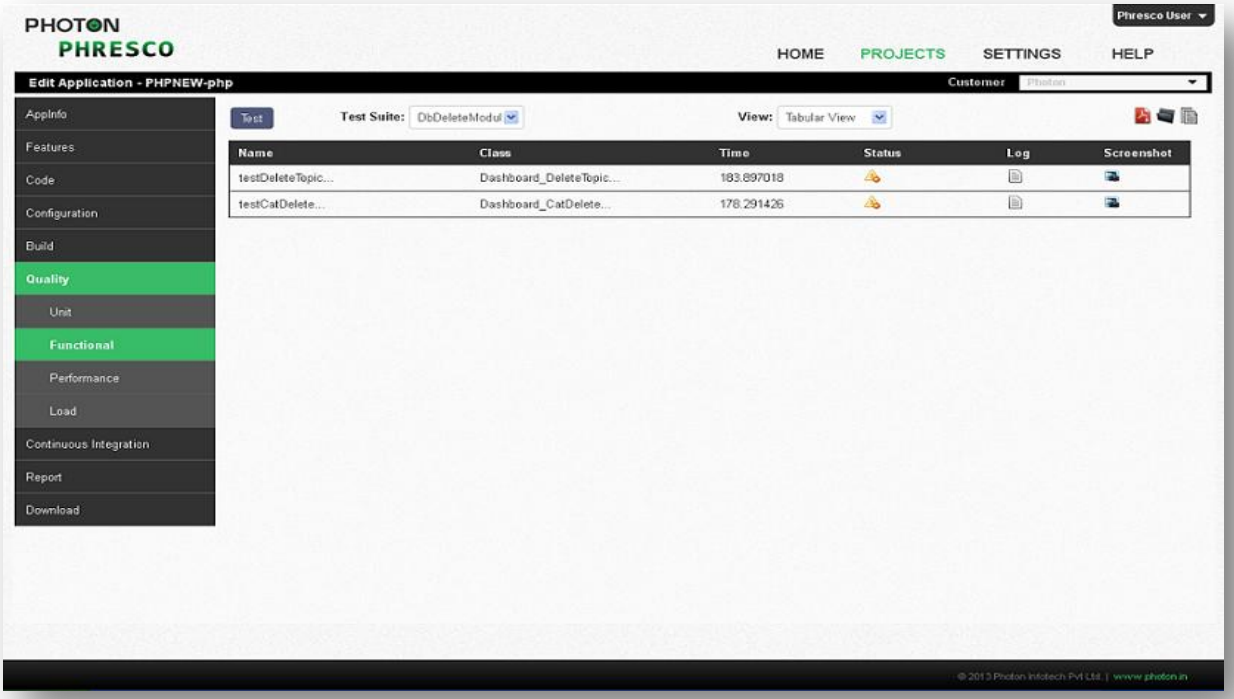

Figure 8-24: PHP functional test report for single test case in tabular view

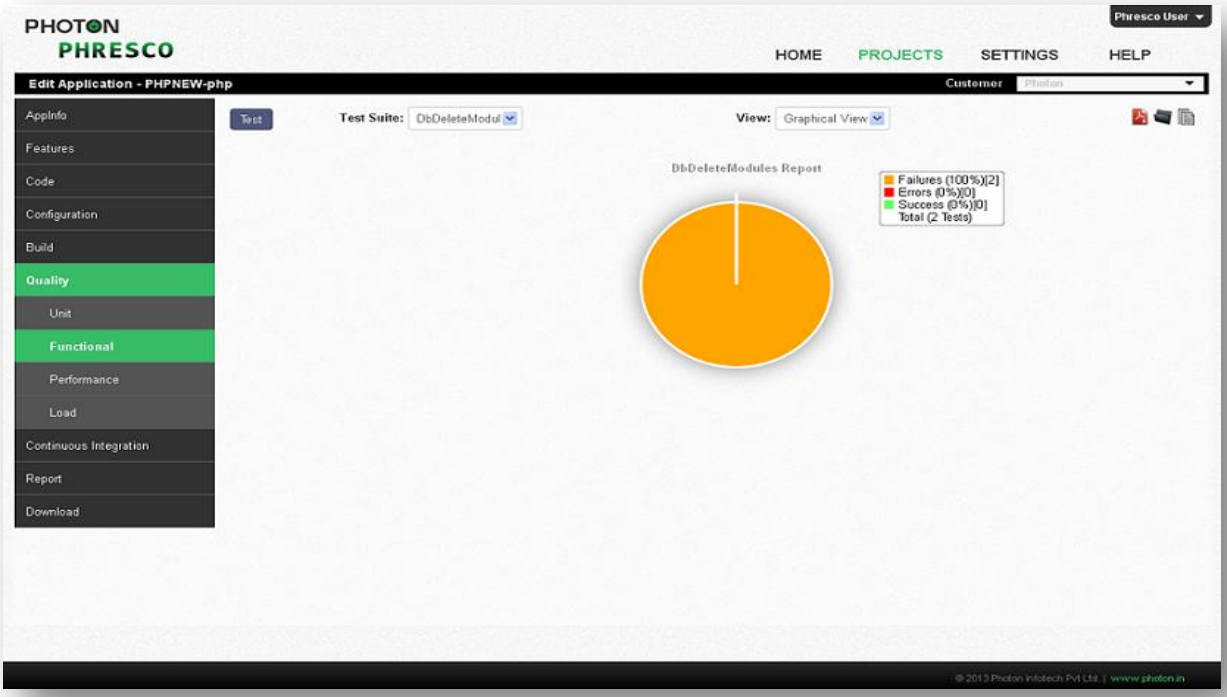

Figure 8-25: PHP functional test report for single test case in graphical view

8.4 Test Cases for SharePoint Technology

8.4.1 Unit Test Case

8.4.1.1 Structure Of Alltest And Test Cases

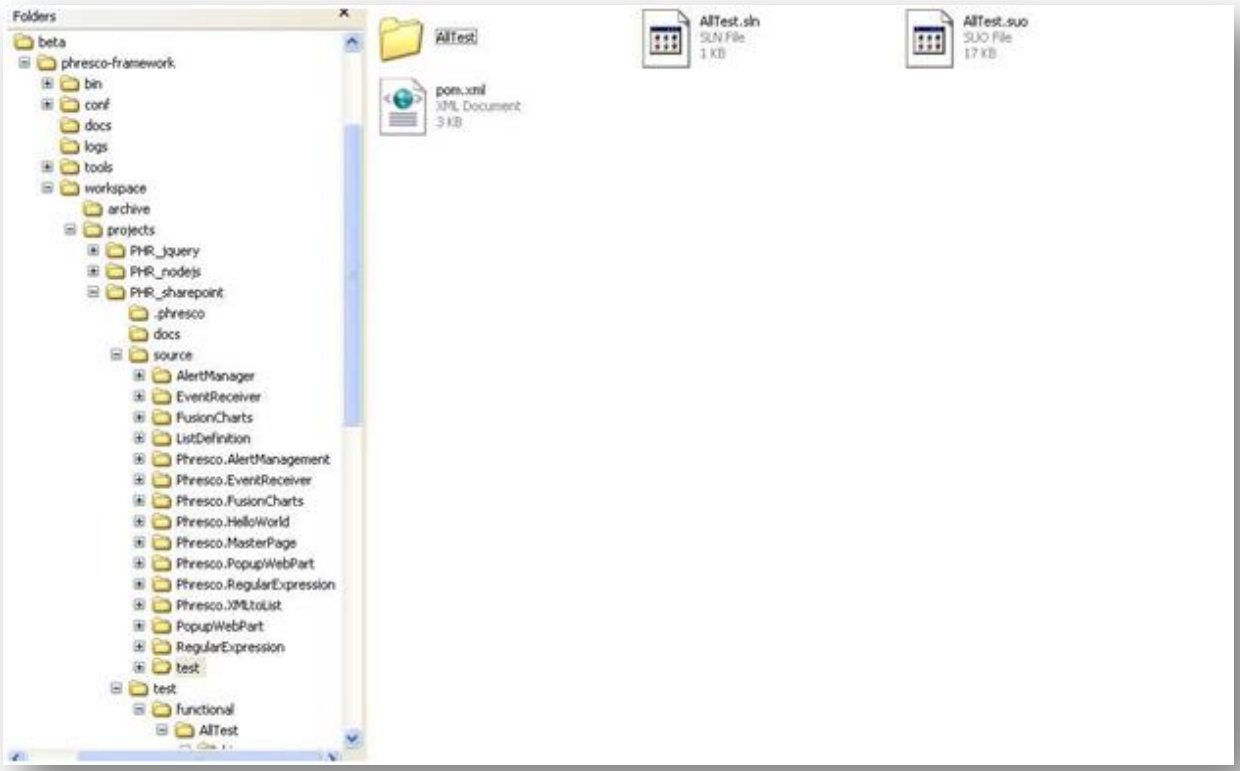

Figure 8-106: SharePoint unit tests structure

a. **AllTest**: This is a root folder/file that carries all the classes to initiate the testing.

b. **Class**: All the classes is incorporated in the AllTest

c. **Test Case**: All the test cases should be added within the Class

Alltest for SharePoint

AllTest contains a bunch of test cases, each of which performs a unique scenario on the application. Test developers can start writing new suite classes and test cases by following the syntax and structure of HeliOS framework's out of the box class structure. AllTest class calls all the suite classes and the test cases within it.

Once suite classes and test cases are written developers can execute those from HeliOS Framework and can see the report. Following is the AllTest file which shows up how to add / include the class inside it.

Class example

Class is a collection of test cases that are intended to be used to test a project code to show that it has some specified set of behaviors. A class contains detailed instructions or goals for each collection of test cases. New test cases can also be added by using the same syntax.

```
using System;
using System.Web;
using System.Text;
using System.Collections.Generic;
using System.Linq;
using NUnit.Framework;
using Phresco.EventReceiver;

namespace Phresco.UnitTesting{

   [TestFixture]
   public class ItemEventReceiver{

      [Test]
      public void ItemEventReceiverTests() {

         ItemEventReceiver itemEventReceiver = new ItemEventReceiver();
         // string itemEventReceiverMessage = "";
         //Assert.AreEqual("", itemEventReceiver.);
      }
      }

    }
```

Test case

Test cases examine all the aspects including inputs and outputs. It gives the detailed steps that should to be followed during the testing process. Testing process follows only the steps that have been written for each test case.

8.4.1.2 Report generated after execution

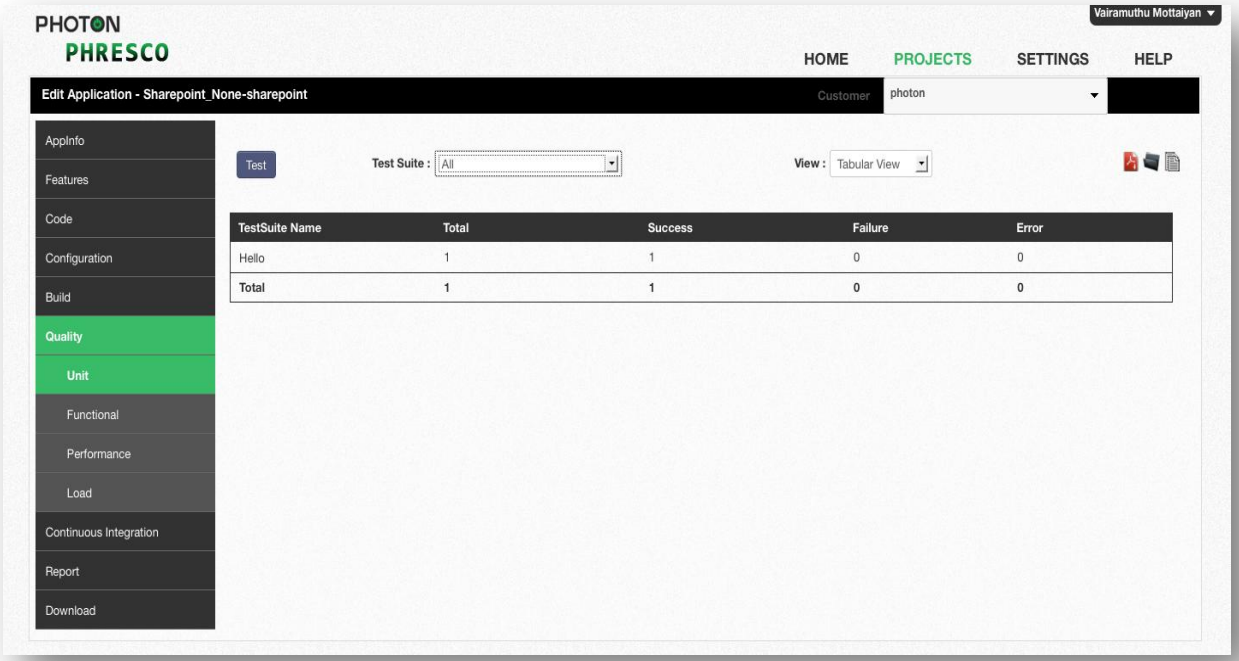

Figure 8-117: SharePoint unit test report for all test cases in tabular view

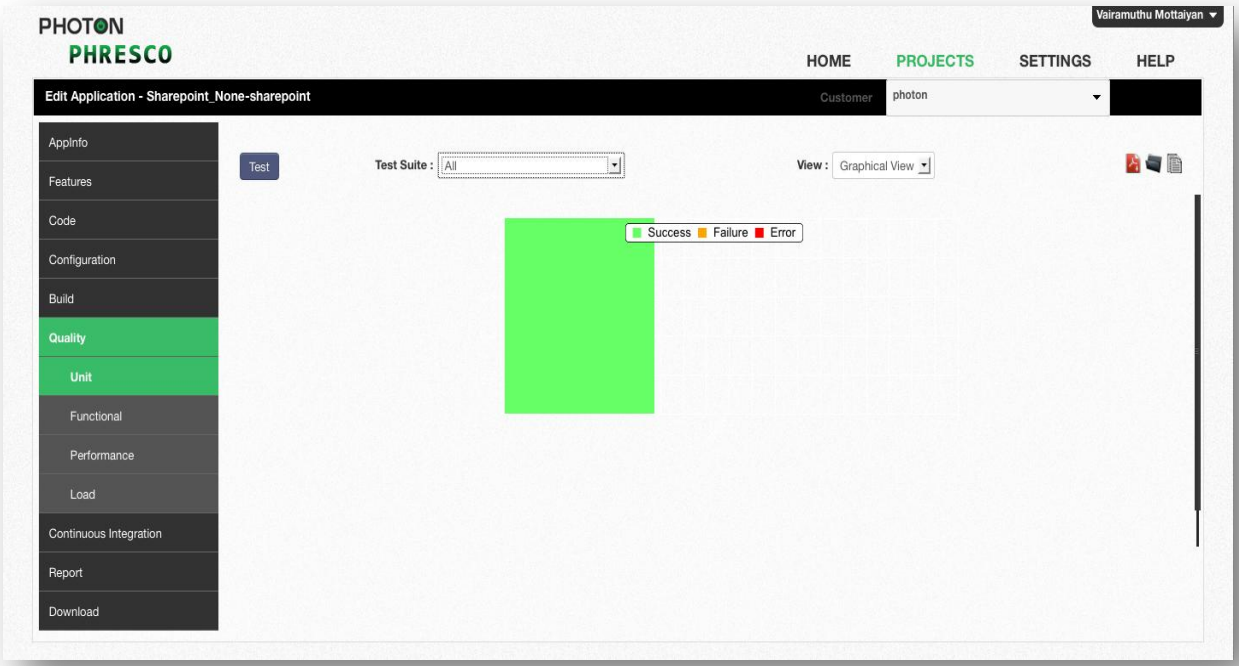

Figure 8-128: SharePoint unit test report for all test cases in tabular view

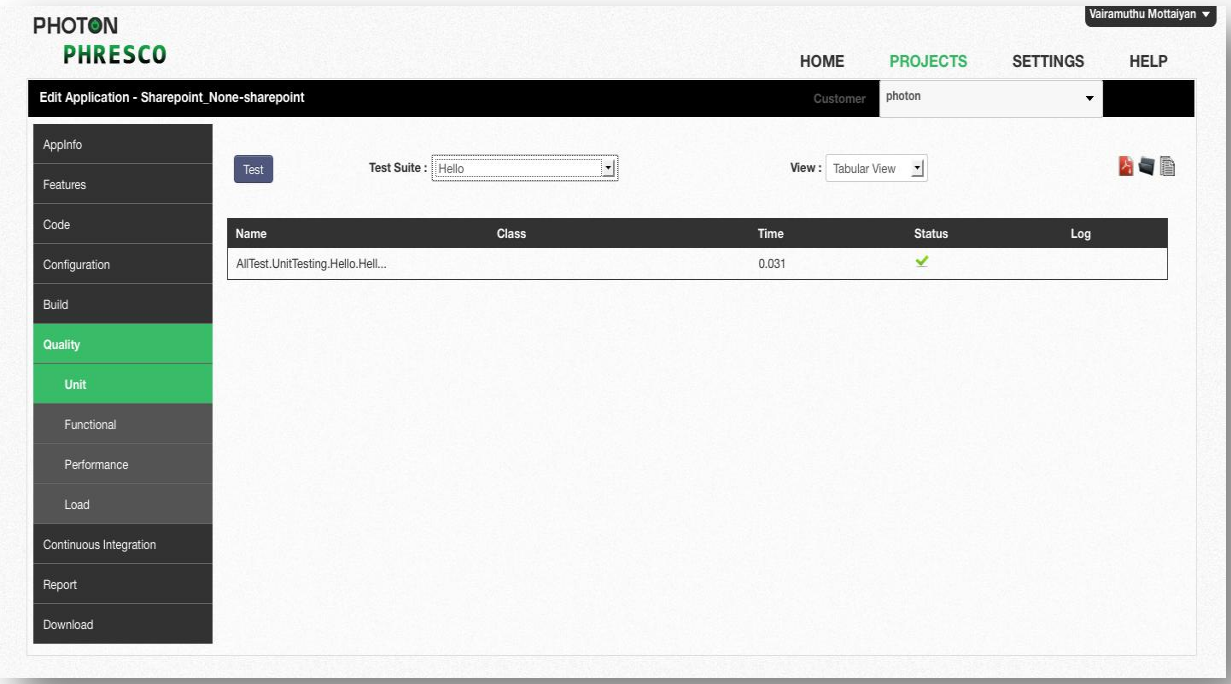

Figure 8-139: SharePoint unit test report for single test case in tabular view

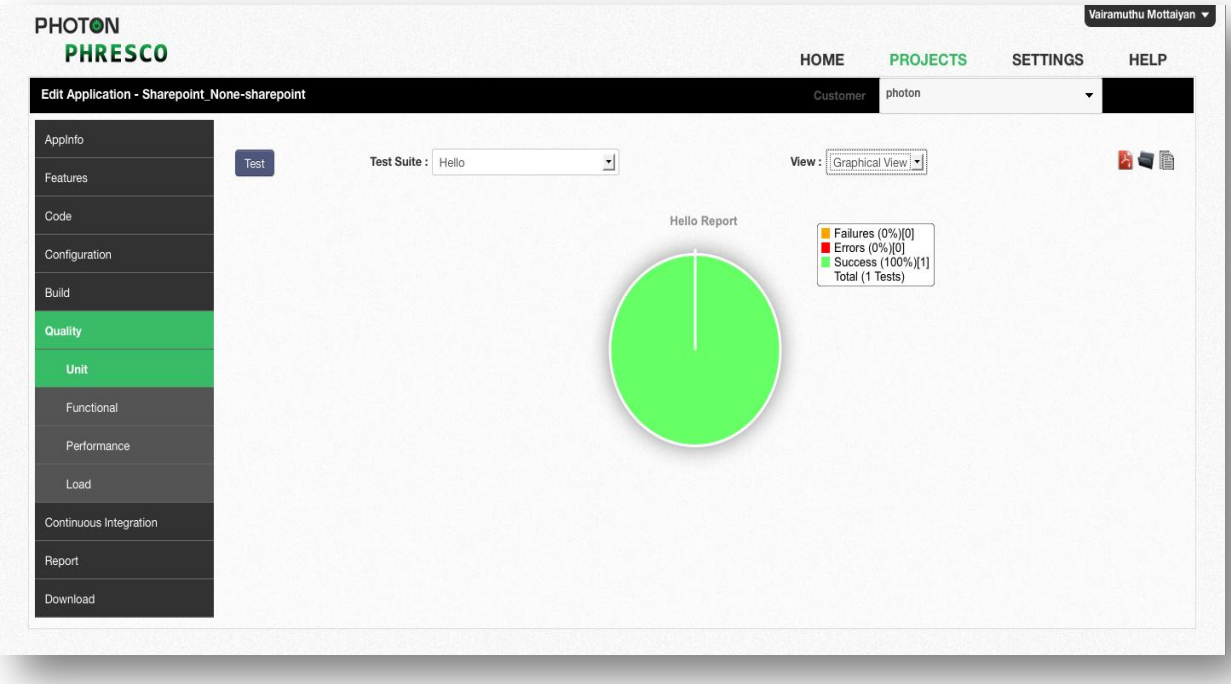

Figure 8-30: SharePoint unit test report for single test case in graphical view

Note

- HeliOS supports Functional Tests for Sharepoint and ASP.Net in TestNG and Webdriver format.
- Refer section 8.1.2 for the Functional Test case structure which is similar to Selenium Grid Functional Test case structure.

8.5 Java Standalone Application

8.5.1 Unit Test Case

8.5.1.1 Structure of Test Case

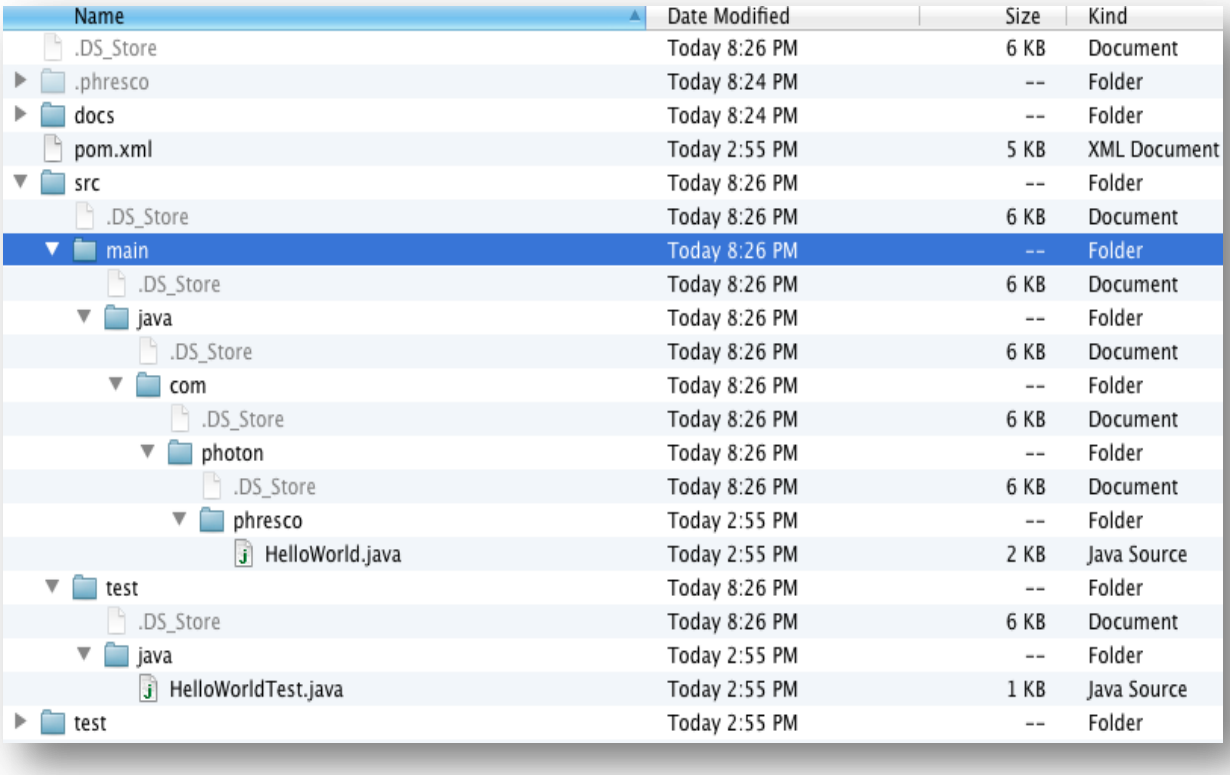

Figure 8-31: Java Standalone folder structure

a. **Test case**: All the Test cases should be added within the test suite.

8.5.1.2 Existing Test Cases and Suites Out Of the Box in HeliOS

Test case example

A test case is a set of conditions under which a tester will determine whether an application is meeting the requirement. This helps the user to verify a particular functionality during the testing process.

```
import junit.framework.Test;
import junit.framework.TestCase;
import junit.framework.TestSuite;

public class HelloWorldTest
   extends TestCase
{
      public HelloWorldTest( String testName )
   {
      super( testName );
   }
   public static Test suite()
   {
      return new TestSuite(HelloWorldTest .class );
   }
   public void testApp()
   {
      assertTrue( true );
  }
 }
```

8.5.2 Functional Test Case

8.5.2.1 Structure of Test Suites and Test Case

a. **AllTest**: Alltest is a java suite class that can either initiate a suite class or a group of test cases.

b. **Test suite**: Test suite is a collection of test cases.

c. **Test case**: All the Test cases should be added within the test suite.

8.5.2.2 Existing Test Cases and Suites Out Of the Box in HeliOS

Alltest for Java Standalone Application

AllTest is the root category which holds the test suite. AllTest contains a bunch of test suites, each of which performs a unique scenario on the application. Test developers can start writing new suite classes by following the syntax and structure of HeliOS framework's out of the box class structure. Once test suite classes and test cases are written developers can execute those from HeliOS Framework and can see the report. Following is the AllTest file which shows up how to add / include the class inside it.

```
package com.photon.Phresco.testcases;

import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)
@SuiteClasses({SampleHelloWorld.class})
public class AllTest {

}
```

Test Suites Example

Test suite is a collection of test cases that are intended to be used to test a project code to show that it has some specified set of behaviors. A test suite contains detailed instructions or goals for each collection of test cases. New test cases can also be added.

Test Case Example

A test case is a set of conditions under which a tester will determine whether an application is meeting the requirement. This helps the user to verify a particular functionality during the testing process. Functional test case for java standalone application is written using the FEST tool.

```
package com.photon.Phresco.testcases;

import static org.fest.assertions.Assertions.assertThat;
import static org.fest.swing.edt.GuiActionRunner.execute;

import org.fest.swing.annotation.RunsInEDT;
import org.fest.swing.edt.GuiQuery;
import org.fest.swing.fixture.FrameFixture;
import org.fest.swing.junit.testcase.FestSwingJUnitTestCase;
import org.junit.Test;
```

```
import com.photon.Phresco.HelloWorld;

public class HelloWorldTest extends FestSwingJUnitTestCase {

	private FrameFixture Phresco;

	protected void onSetUp() {
		Phresco = new FrameFixture(robot(), createNewEditor());
		Phresco.show();
		Phresco.maximize();
		System.out.println("**************Executed
onsetup**************");
	}

	@RunsInEDT
	private static HelloWorld createNewEditor() {
		return execute(new GuiQuery<HelloWorld>() {
			protected HelloWorld executeInEDT() throws
Throwable {
				return new HelloWorld();
			}
		});
	}

@Test
public void testHelloWorld() throws InterruptedException {
		Thread.sleep(5000);
		assertThat(Phresco.label().text()).contains("Hello World");
	}
}
```

8.5.2.3 To Add New Test Case In Test Suite

You can add a new test case to the Test suite using the following method.

```
@Test
public void testHelloWorld() throws InterruptedException {
        Thread.sleep(5000);
        assertThat(Phresco.label().text()).contains("Hello World");
```

8.5.2.4 Report generated after execution

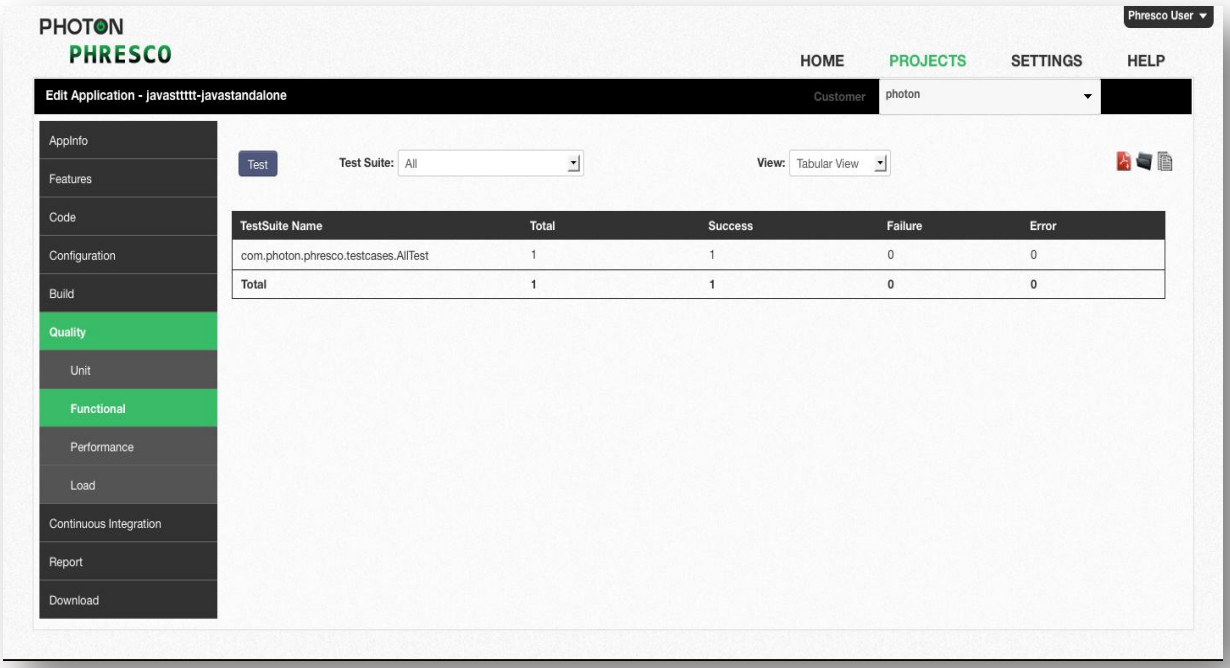

Figure 8-32: ***Java Standalone functional test report for all test cases in tabular view***

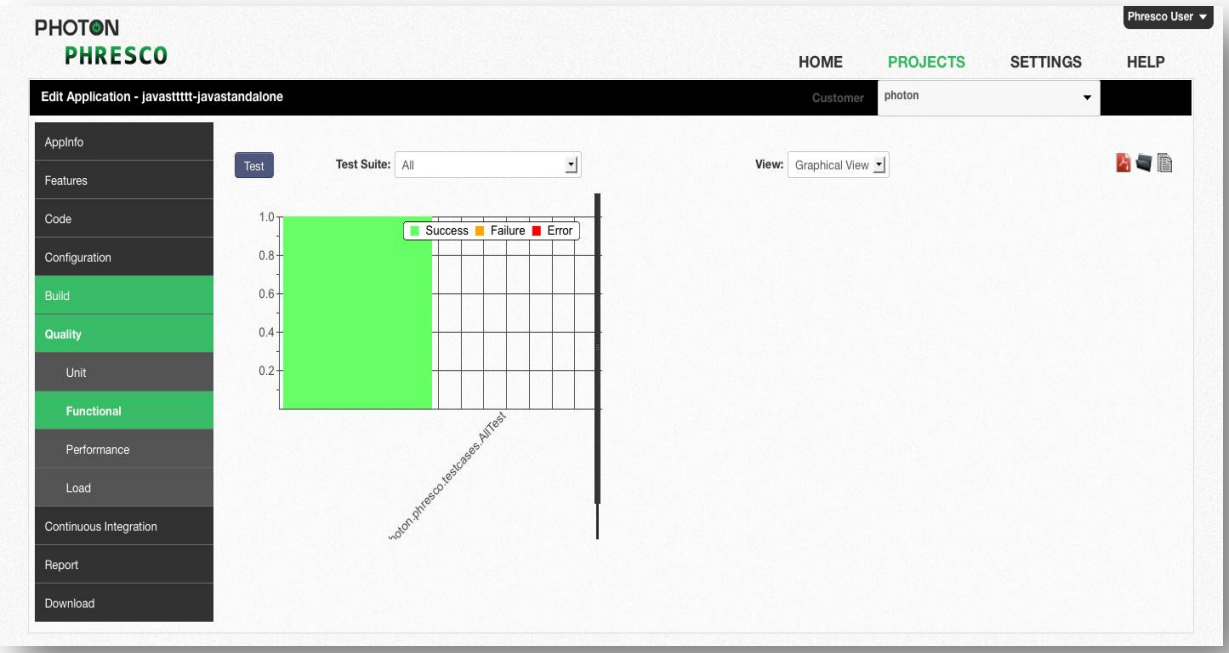

Figure 8-33: ***Java Standalone functional test report for all test cases in graphical view***

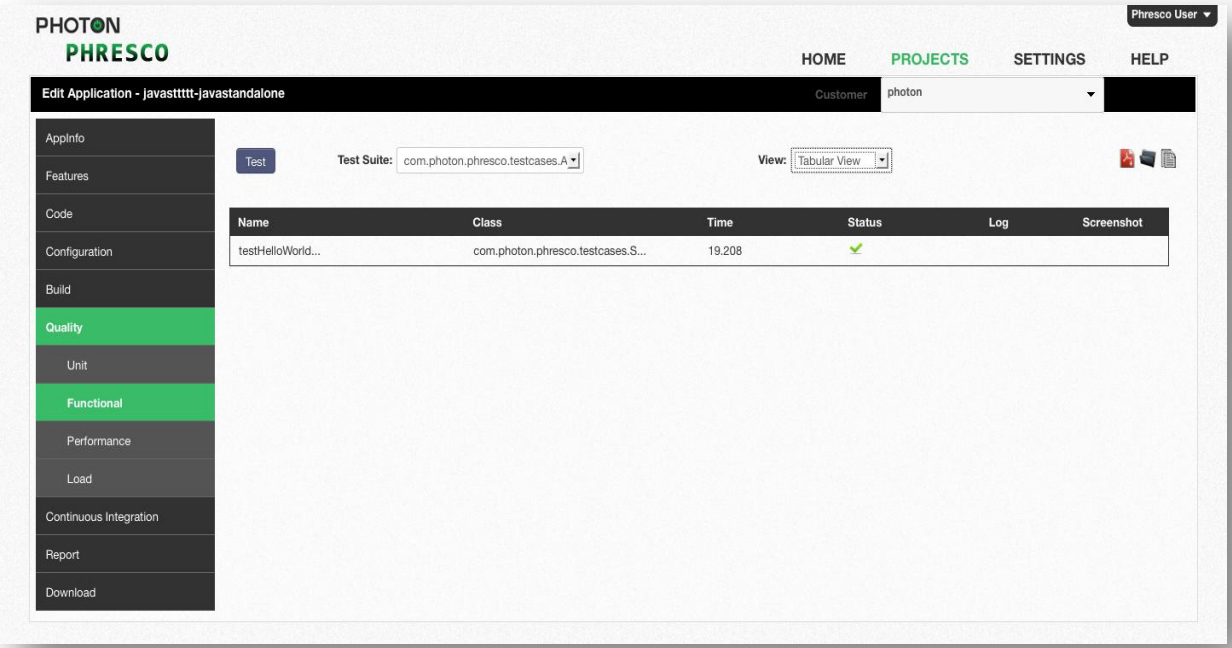

Figure 8-34: ***Java Standalone functional test report for single test case in tabular view***

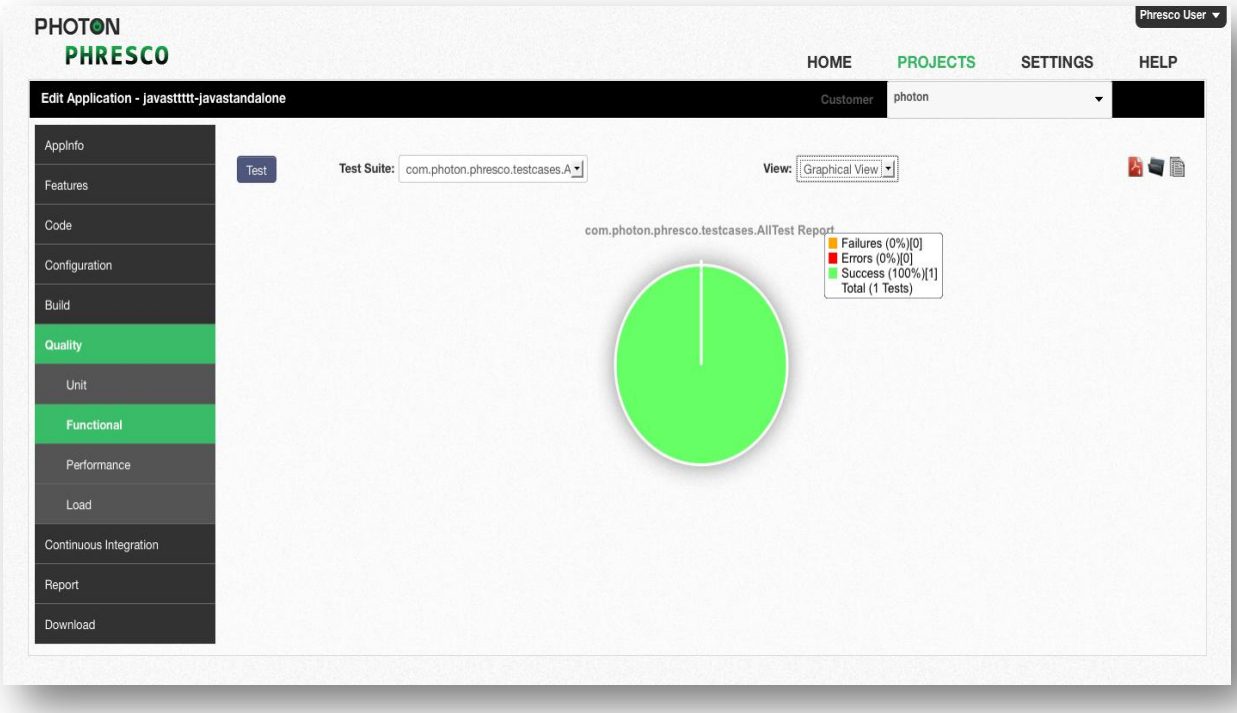

Figure 8-35: ***Java Standalone functional test report for single test case in graphical view***

8.6 Android application

8.6.1 Unit Test Case

8.6.1.1 Structure of Alltest and Test Cases

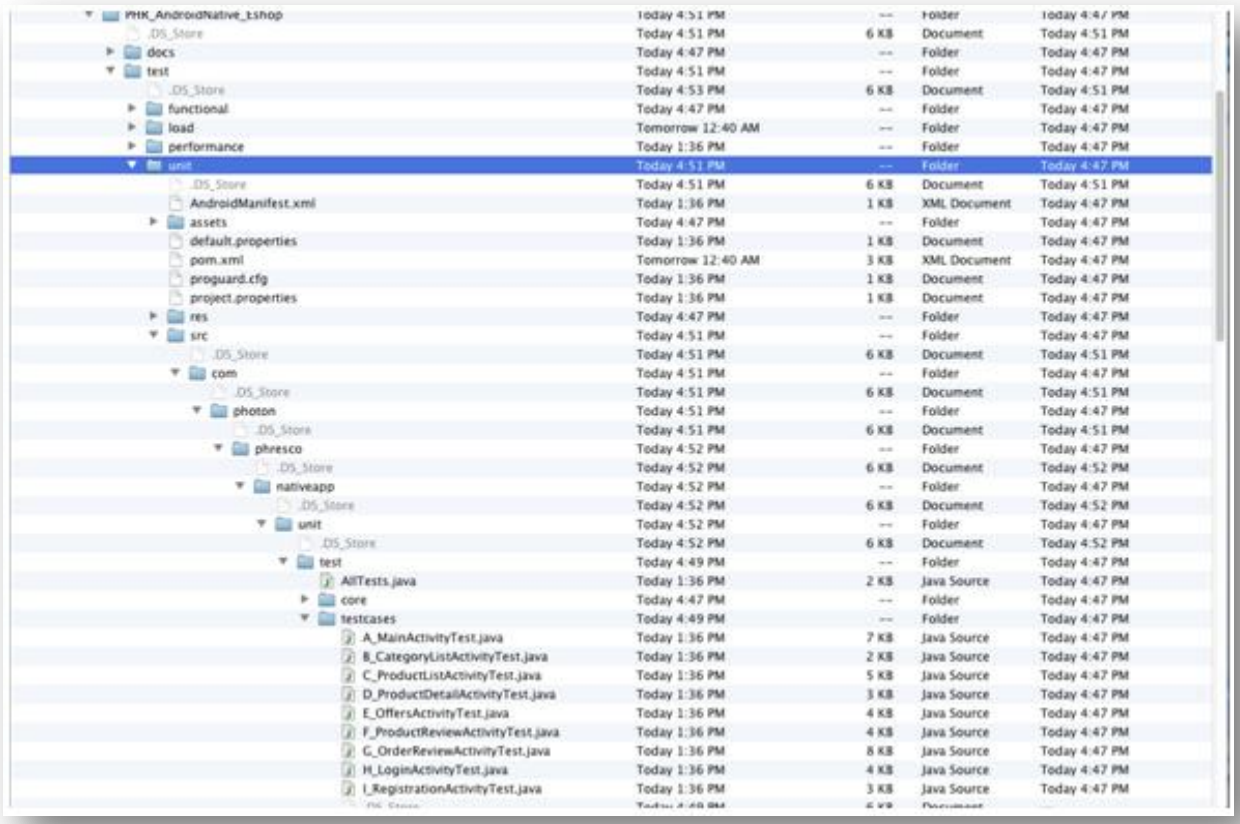

Figure 8-36: Android unit tests structure

a. **AllTest (test suite):** AllTest is the class that carries all the test classes to initiate the testing process. Each test class can be called within the AllTest.

b. **Test class:** Test classes hold different test methods

c. **Test methods/test cases:** Test cases are called in the test classes which test the source code.

8.6.1.2 Existing Test Cases Out Of the Box in HeliOS

AllTest for Android Technology

AllTest contains a bunch of test cases, each of which performs a unique scenario on the application. Test developers can start writing new suite classes and test cases by following the syntax and structure of HeliOS framework's out of the box class structure. AllTest class calls all the suite classes and the test cases within it. Once suite classes and test cases are written developers can execute those from HeliOS Framework and can see the report. Following is the AllTest file which shows up how to add / include the class inside it.

```
package com.photon.Phresco.nativeapp.unit.test;

import junit.framework.TestCase;
import junit.framework.TestSuite;

import com.photon.Phresco.nativeapp.unit.test.testcases.A_MainActivityTest;
import com.photon.Phresco.nativeapp.unit.test.testcases.B_CategoryListActivityTest;
import com.photon.Phresco.nativeapp.unit.test.testcases.C_ProductListActivityTest;
import com.photon.Phresco.nativeapp.unit.test.testcases.D_ProductDetailActivityTest;
import com.photon.Phresco.nativeapp.unit.test.testcases.E_OffersActivityTest;
import com.photon.Phresco.nativeapp.unit.test.testcases.F_ProductReviewActivityTest;
import com.photon.Phresco.nativeapp.unit.test.testcases.G_OrderReviewActivityTest;
import com.photon.Phresco.nativeapp.unit.test.testcases.H_LoginActivityTest;

public class AllTests extends TestCase {
        public static TestSuite suite() {

                TestSuite suite = new TestSuite(AllTests.class.getName());

                suite.addTestSuite(A_MainActivityTest.class);
                suite.addTestSuite(B_CategoryListActivityTest.class);
                suite.addTestSuite(C_ProductListActivityTest.class);
                suite.addTestSuite(D_ProductDetailActivityTest.class);
                suite.addTestSuite(E_OffersActivityTest.class);
                suite.addTestSuite(F_ProductReviewActivityTest.class);
                suite.addTestSuite(G_OrderReviewActivityTest.class);
                suite.addTestSuite(H_LoginActivityTest.class);

                return suite;

        }
        public ClassLoader getLoader() {
             return AllTests.class.getClassLoader();
          }
}
```

Test Method Example for Loginactivitytest

This test method is written for testing the login activity. There can be n number of test methods under a test class. Test class calls the test methods. An example of a test class is given below

```
package com.photon.Phresco.nativeapp.unit.test.testcases;
import java.io.IOException;
import junit.framework.TestCase;
import org.json.JSONException;
import org.json.JSONObject;
import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import com.photon.Phresco.nativeapp.eshop.json.JSONHelper;
import com.photon.Phresco.nativeapp.eshop.logger.PhrescoLogger;
import com.photon.Phresco.nativeapp.unit.test.core.Constants;

public class H_LoginActivityTest extends TestCase{
	private static final String TAG = "H_LoginActivityTest  ********** ";

	/**
	 * @throws java.lang.Exception
	 */
	@BeforeClass
	public static void setUpBeforeClass() {
	}

	/**
	 * @throws java.lang.Exception
	 */
	@AfterClass
	public static void tearDownAfterClass() {
	}

	/**
	 * @throws java.lang.Exception
	 */
	@Before
	public void setUp() {
	}
```

```
	/**
	 * @throws java.lang.Exception
	 */
	@After
	public void tearDown() {
	}

	/**
	 *  send valid login email id and password to web server
	 *
	 */
	@Test
	public final void testLogin() {

		JSONObject jObjMain = new JSONObject();
		JSONObject jObj = new JSONObject();

	try {
	PhrescoLogger.info(TAG + " testLogin ---------- START ");

		jObj.put("loginEmail", "tester@Phresco.com");
		jObj.put("password", "123");
		jObjMain.put("login", jObj);

		JSONObject responseJSON = null;

	responseJSON=JSONHelper.postJSONObjectToURL(Constants.getW
ebContextURL() + Constants.getRestAPI() + Constants.LOGIN_POST_URL,
jObjMain.toString());
assertNotNull("Login response is not null",responseJSON.length() > 0);

PhrescoLogger.info(TAG + " testLogin ---------- END ");

} catch (IOException ex) {
PhrescoLogger.info(TAG + " - testLogin  - IOException : " + ex.toString());
PhrescoLogger.warning(ex);
} catch (JSONException ex) {
PhrescoLogger.info(TAG + " - testLogin  - JSONException : " + ex.toString());
PhrescoLogger.warning(ex);
}
}
```

Test Case Example for Test Login

A test case is a set of conditions under which a tester will determine whether an application is meeting the requirement. This helps the user to verify a particular functionality during the testing process.

Example of test case in login activity is given below:

```
@Test
	public final void testLogin() {

		JSONObject jObjMain = new JSONObject();
		JSONObject jObj = new JSONObject();

		try {
		PhrescoLogger.info(TAG + " testLogin ---------- START ");

			jObj.put("loginEmail", "tester@Phresco.com");
			jObj.put("password", "123");
			jObjMain.put("login", jObj);

			JSONObject responseJSON = null;

	responseJSON=JSONHelper.postJSONObjectToURL(Constants.getWebCont
extURL() + Constants.getRestAPI() + Constants.LOGIN_POST_URL,
jObjMain.toString());
	assertNotNull("Login response is not null",responseJSON.length() > 0);
	PhrescoLogger.info(TAG + " testLogin ---------- END ");

	} catch (IOException ex) {
	PhrescoLogger.info(TAG + " - testLogin  - IOException : " + ex.toString());
	PhrescoLogger.warning(ex);
	} catch (JSONException ex) {
	PhrescoLogger.info(TAG + " - testLogin  - JSONException : " + ex.toString());
			PhrescoLogger.warning(ex);
		}
}
```

8.6.1.3 To Add New Test Method in Alltest Class

You can add a new test method to the AllTest class as follows.

Here is an example for creating a new test suite in the name 'RegistrationActivityTest'

```
import com.photon.Phresco.nativeapp.unit.test.testcases.A_MainActivityTest;
import com.photon.Phresco.nativeapp.unit.test.testcases.B_CategoryListActivityTest;
import com.photon.Phresco.nativeapp.unit.test.testcases.C_ProductListActivityTest;
import com.photon.Phresco.nativeapp.unit.test.testcases.D_ProductDetailActivityTest;
import com.photon.Phresco.nativeapp.unit.test.testcases.E_OffersActivityTest;
import com.photon.Phresco.nativeapp.unit.test.testcases.F_ProductReviewActivityTest;
import com.photon.Phresco.nativeapp.unit.test.testcases.G_OrderReviewActivityTest;
import com.photon.Phresco.nativeapp.unit.test.testcases.H_LoginActivityTest;
import com.photon.Phresco.nativeapp.unit.test.testcases.I_RegistrationActivityTest;
```

8.6.1.4 Report generated after execution

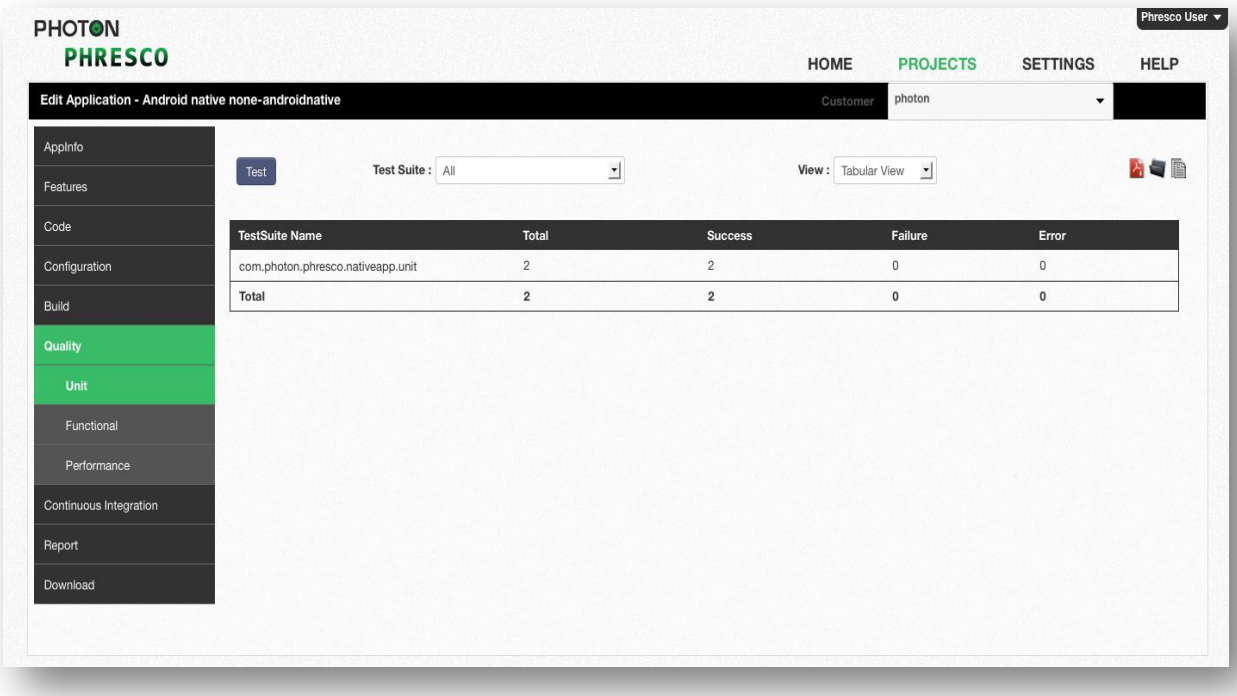

Figure 8-147: Android unit test report for all test cases in tabular view

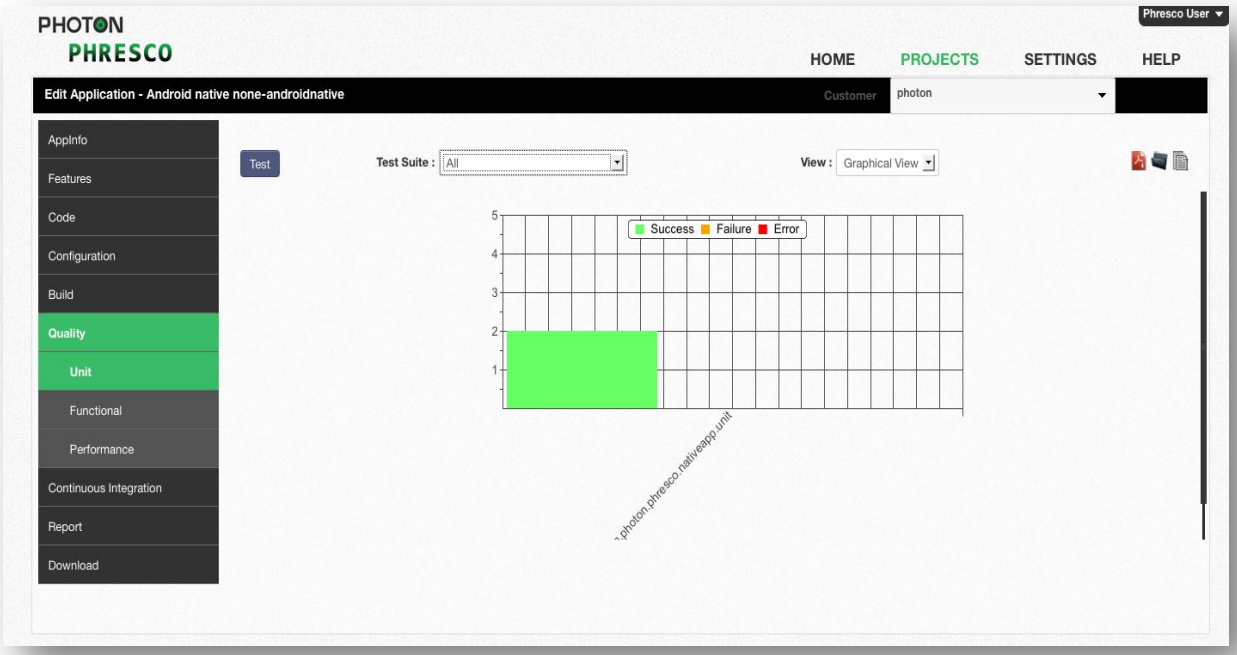

Figure 8-158: ***Android unit test report for all test cases in graphical view***

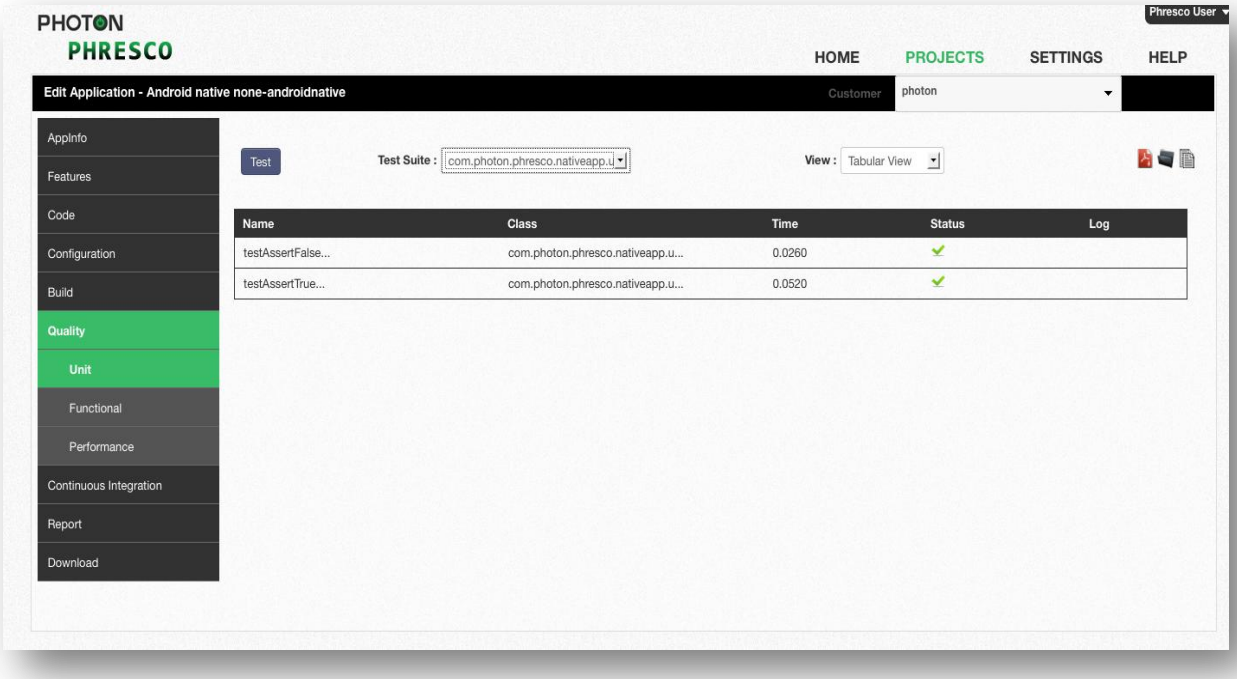

Figure 8-169: ***Android unit test report for single test case in graphical view***

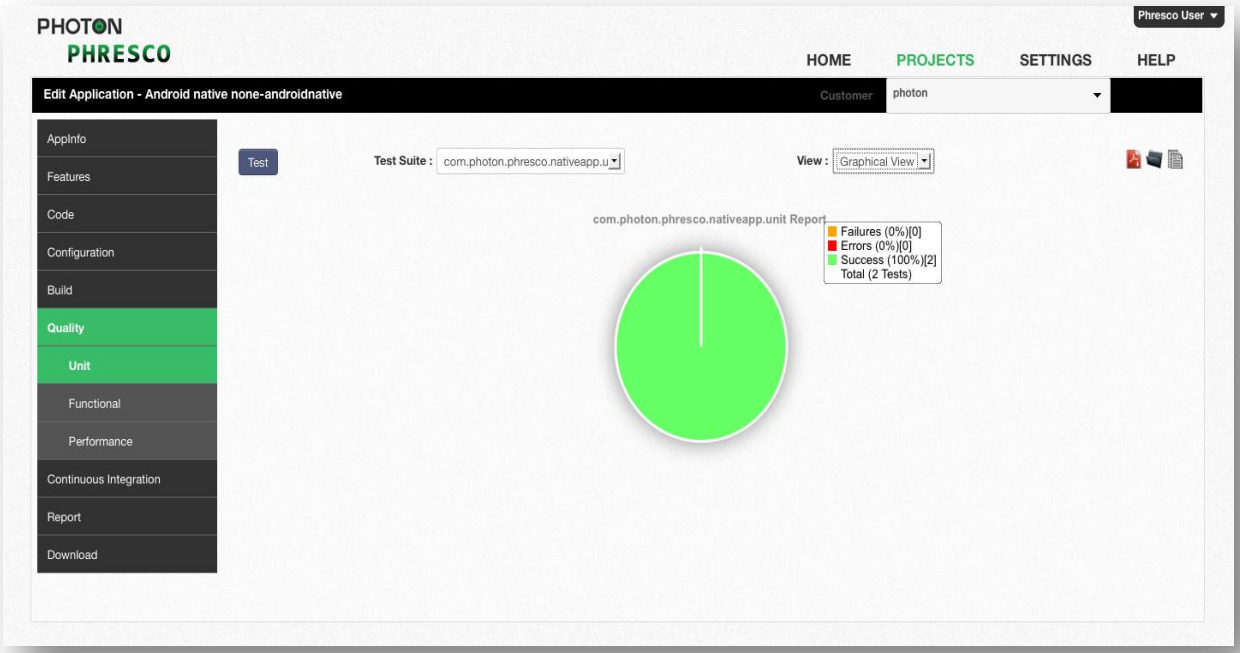

Figure 8-40: ***Android unit test report for single test case in graphical view***

8.6.2 Functional Test Case for Android Native - Robotium

8.6.2.1 Structure of Functional Test of Android Native

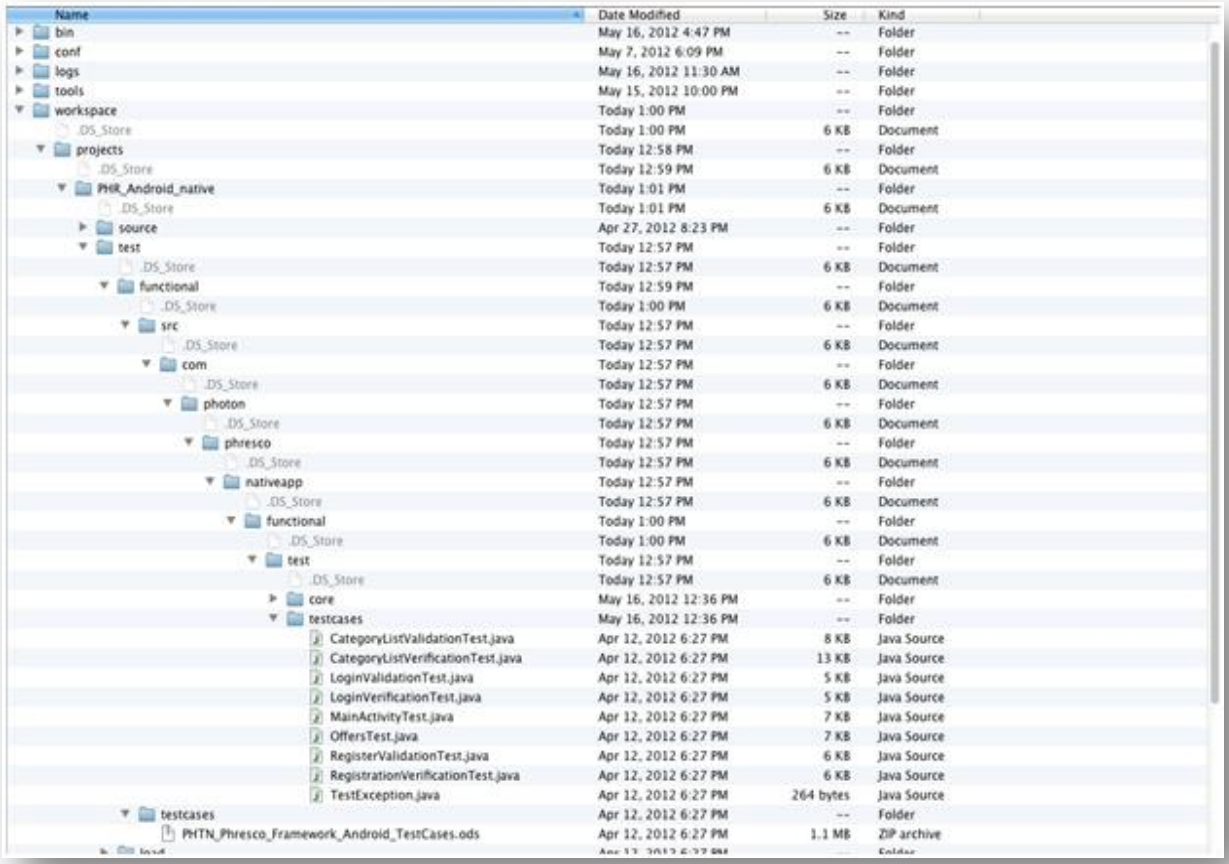

Figure 8-41: Android Native functional tests structure

a. **MainActivity test:** Main activity test is the root folder that calls all the test cases written separately.This initiates the testing process.

b. **Test cases :** Test cases are the test methods that are called by the main activity test. Test cases can be many in number

Main Activity Test

Main activity test is a class that calls all the test cases within itself. It contains a bunch of test cases, each of which performs a unique scenario on the application. Test developers can start writing new suite classes and test cases by following the syntax and structure of HeliOS framework's out of the box class structure. Once test cases are written developers can execute those from HeliOS Framework and

can see the report. Following is the file which shows up how to add / include the class inside it.

```
package com.photon.Phresco.nativeapp.functional.test.testcases;
import android.test.ActivityInstrumentationTestCase2;
import android.test.suitebuilder.annotation.Smoke;
import android.util.Log;
import com.jayway.android.robotium.solo.Solo;
import com.photon.Phresco.nativeapp.eshop.activity.MainActivity;

@SuppressWarnings("unchecked")
public class MainActivityTest extends
ActivityInstrumentationTestCase2<MainActivity> {

 /**
  * This is suite testcase by this testcase will call other testcases . In
  * static block we are loading the MainActivity class and from the
  * constructor will pass the package and activity full class name then in
  * setUp() created the Solo class object
  *
  */
 public static final String PACKAGE_NAME =
"com.photon.Phresco.nativeapp";
 private static final String LAUNCHER_ACTIVITY_FULL_CLASSNAME =
"com.photon.Phresco.nativeapp.eshop.activity.MainActivity";
 private static Class<MainActivity> mainActivity;
 private Solo soloMain;
 private LoginValidationTest loginValid;
 private final String TAG = "MainTestCase****";

 /**
  * This block will be executed first and it will loads the SplashActivity .
  */
 static {
        try {
                mainActivity = (Class<MainActivity>)
Class.forName(LAUNCHER_ACTIVITY_FULL_CLASSNAME);
        }

        catch (ClassNotFoundException e) {
                throw new RuntimeException(e);
        }
 }
```

```
/**
 * In this constructor , we have to send the packagename and activity full
 * class name.
 *
 * @throws Exception
 */
public MainActivityTest() throws Exception {
        super(PACKAGE_NAME, mainActivity);
}

/**
 * this method for create the Solo class object having two super class
 * methods.
 *
 */
@Override
public void setUp() {

        soloMain = new Solo(getInstrumentation(), getActivity());

}
           /**
 * This test method will cAllTestLoginValidation() method.It verifies
 * the Validation for Login screen.
 *
 */

public void testValidationLogin() throws TestException {

        try {
                Log.i(TAG, "testValidationLogin---------Start");
                // creating objectof the class LoginValidationTestCase
                loginValid = new LoginValidationTest(soloMain);
                loginValid.testLoginValidation();
                Log.i(TAG, "testValidationLogin---------End");
        } catch (TestException e) {
                e.printStackTrace();

        }

}
```

Test cases

A test case is a set of conditions under which a tester will determine whether an application is meeting the requirement. This helps the user to verify a particular functionality during the testing process. Elements can be identified and screen shots can be captured when the test fails. Test cases can be added by writing the test cases separately and by calling within the Main activity test.

```
package com.photon.Phresco.nativeapp.functional.test.testcases;
import java.util.ArrayList;
import java.util.Iterator;
import junit.framework.TestCase;
import android.util.Log;
import android.view.View;
import android.widget.EditText;
import android.widget.ImageView;
import com.photon.Phresco.nativeapp.R;
import com.jayway.android.robotium.solo.Solo;

public class LoginValidationTest extends TestCase {

 private Solo soloLoginValid;
 private String activityName;
 public ImageView clickCancel;
 private String TAG = "*****LoginValidationTestCase******";

 public LoginValidationTest(Solo solo) {
        this.soloLoginValid = solo;

 }

 public void testLoginValidation() throws TestException {

        try {
                Log.i(TAG, "------It is testLoginValidation()-----------");

                activityName =
soloLoginValid.getCurrentActivity().getClass().getSimpleName();

        if (activityName.equalsIgnoreCase("MainActivity")) {
                Log.i(TAG, "------It is MainActivity-----------" + activityName);
                        soloLoginValid.waitForActivity("HomeActivity", 2000);
```

```
		for (int i = 0; i < 40; i++) {
				activityName =
soloLoginValid.getCurrentActivity().getClass().getSimpleName();
		if (activityName.equalsIgnoreCase("HomeActivity")) {
				Log.i(TAG, "------for()-- loop-----");
				break;
			}
			soloLoginValid.waitForActivity("HomeActivity", 2000);
		}
		} else {
			Log.i(TAG, "------ testLoginValidation failed-----------");
			throw new TestException("Current Activity Failed----"
								+
soloLoginValid.getCurrentActivity().getClass().getSimpleName() + "failed");
		}
		if (activityName.equalsIgnoreCase("HomeActivity")) {
				Log.i(TAG, "------HomeActivity-----------");
				System.out.println(" Activity name ---->" +
soloLoginValid.getCurrentActivity());
				ArrayList<View> al = soloLoginValid.getViews();
				Iterator<View> it = al.iterator();
				while (it.hasNext()) {
			String viewName = it.next().getClass().getSimpleName();
				if (viewName.equalsIgnoreCase("ImageView")) {
				Log.i(TAG, "------ImageView found-----------");
				break;
				}
				continue;
			}
			} else {
				Log.i(TAG, "------HomeActivity not  found-----------");
				throw new TestException(TAG +
soloLoginValid.getCurrentActivity().getClass().getSimpleName() + "failed");
			}
			// click on Loginbutton
			soloLoginValid.waitForActivity("MainActivity", 5000);
			// get the login button view with id i.e home_login_btn
			ImageView loginButton = (ImageView) soloLoginValid
					.getView(R.id.home_login_btn);
			// click on login button with view id
			soloLoginValid.clickOnView(loginButton);
			// control waits for 2 seconds to activate the screen
			soloLoginValid.waitForActivity("SplashActivity", 2000);
```

```
                        // clears the text at first Editfield
                        EditText emailField = (EditText)
        soloLoginValid.getView(R.id.txt_email);
                soloLoginValid.clearEditText(emailField);
                // it will type the text at first field which i gave in method
                soloLoginValid.enterText(emailField, "android@");
                // clear the text at second Editfield
                EditText passwordField = (EditText) soloLoginValid
                                .getView(R.id.txt_password);
                soloLoginValid.clearEditText(passwordField);
                // finding the password field view
                // click the password field based on EditText view object
                soloLoginValid.clickOnView(passwordField);
                soloLoginValid.waitForActivity("MainActivity", 1000);
                soloLoginValid.enterText(passwordField, "*****");
                soloLoginValid.waitForActivity("MainActivity", 1000);
                // click on Login button
                ImageView clickLogin = (ImageView) soloLoginValid
                                .getView(R.id.login_btn);
                soloLoginValid.clickOnView(clickLogin);
                // soloSplash.clickOnImageButton(1);
                soloLoginValid.waitForActivity("LoginActivity");
                soloLoginValid.clickOnView(emailField);
                soloLoginValid.waitForActivity("LoginActivity", 1000);
                boolean valid = soloLoginValid.searchText("Invalid Email address!");
                if (valid) {
                        assertTrue("Invalid Email address!", valid);
                } else {
                        throw new TestException("Testcase failed");
                }
                soloLoginValid.waitForActivity("LoginActivity", 1000);
                // Get the view location of cancel button.
                clickCancel = (ImageView) soloLoginValid.getView(R.id.cancel_btn);
                soloLoginValid.waitForActivity("MainActivity", 1000);
                // click on cancel button
                soloLoginValid.clickOnView(clickCancel);
                soloLoginValid.waitForActivity("LoginActivity", 1000)
        } catch (TestException e) {
                e.printStackTrace();
        }
        }
}
```

8.6.2.2 To Add A New Test Case

You can add a new test case to the Main activity test using the following method. Here is an example for creating a new test case 'testRegististrationValidation'

```
public static final String PACKAGE_NAME = "com.photon.Phresco.nativeapp";
 private static final String LAUNCHER_ACTIVITY_FULL_CLASSNAME =
"com.photon.Phresco.nativeapp.eshop.activity.MainActivity";
        private static Class<MainActivity> mainActivity;
        private Solo soloMain;
        private LoginValidationTest loginValid;
  private testRegististrationValidation;
        private final String TAG = "MainTestCase****";

      @Smoke
public void testRegististrationValidation() throws TestException {

        try {
                Log.i(TAG, "testRegististrationValidation---------Start");
                // creating object of the testclass RegisterValidationTestCase
                registrationValid = new RegisterValidationTest(soloMain);
                registrationValid.testRegisterValidation();
                Log.i(TAG, "testRegististrationValidation---------End");

        } catch (TestException e) {
                e.printStackTrace();
        }
}
```

8.6.2.3 Report generated after execution

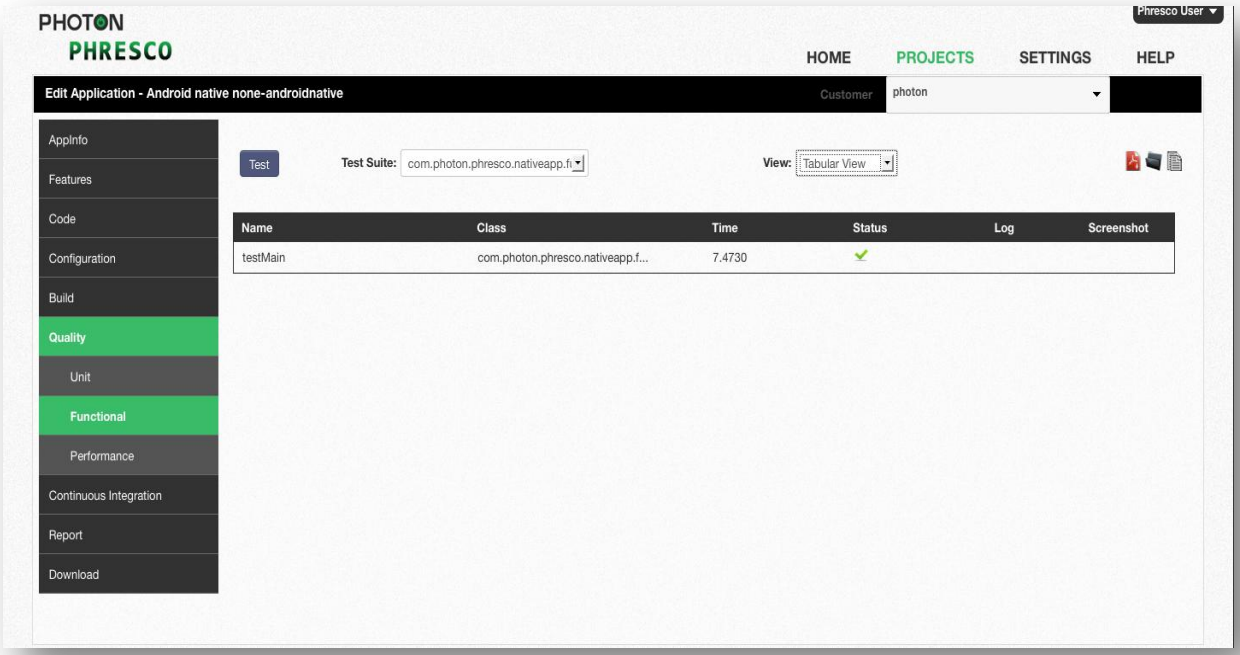

Figure 8-42: ***Android Native functional test report for all test cases in tabular view***

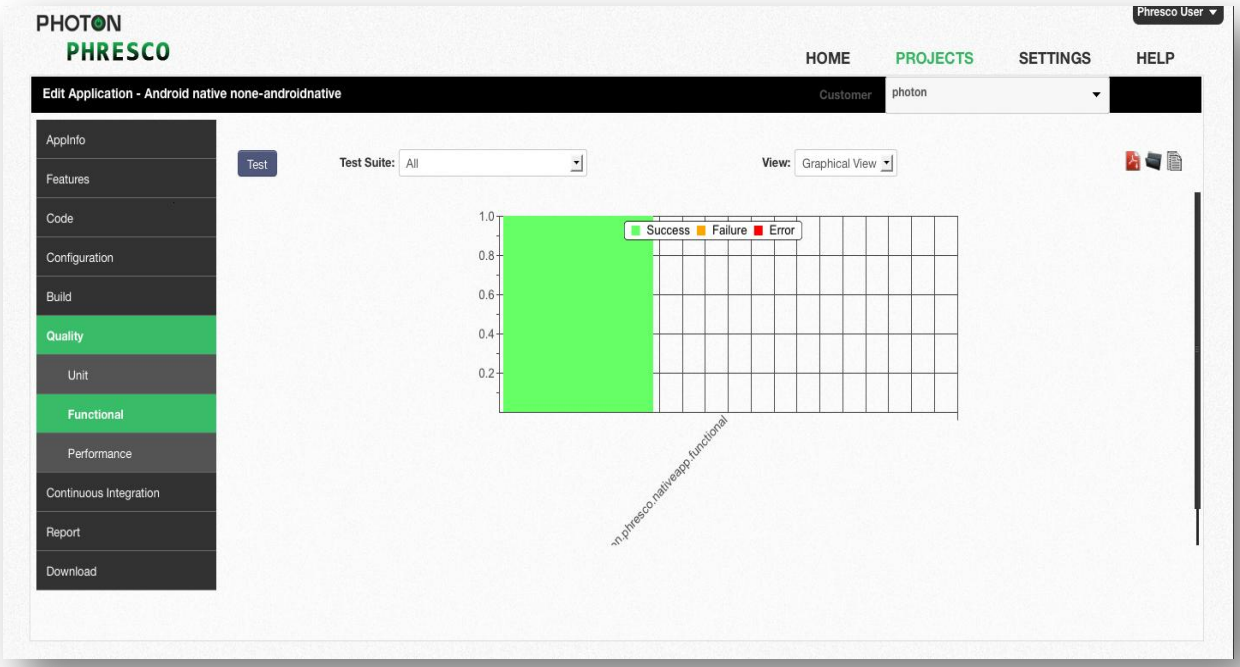

Figure 8-4317: ***Android Native functional test report for all test cases in graphical view***

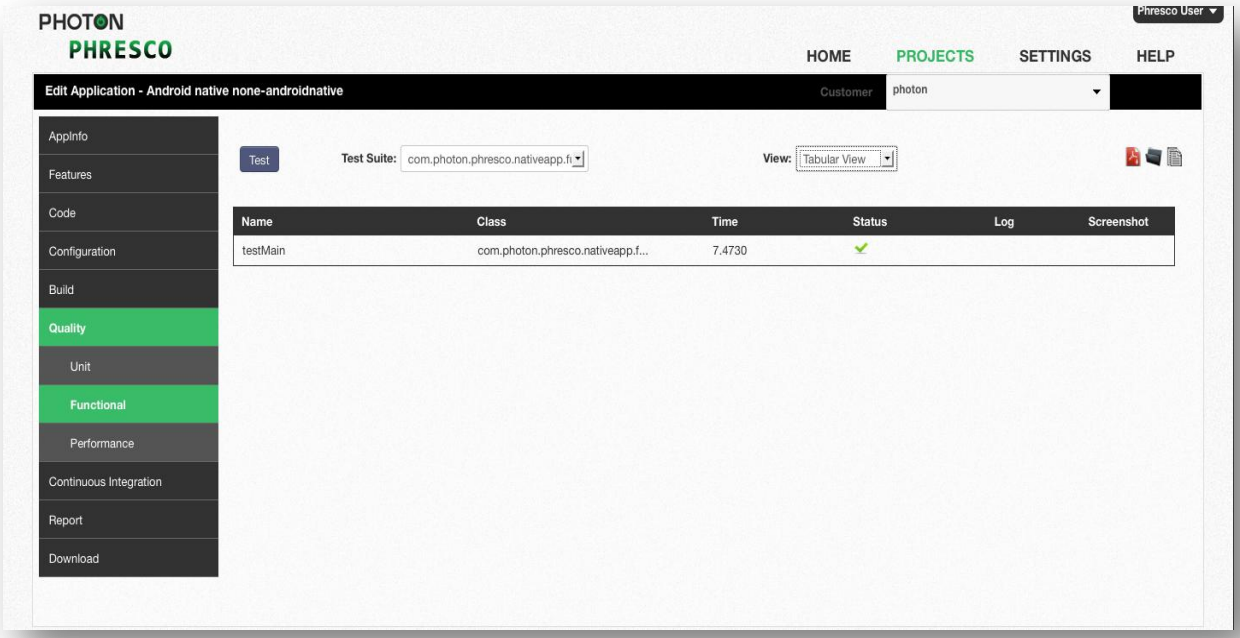

Figure 8-44: ***Android Native functional test report for single test case in tabular view***

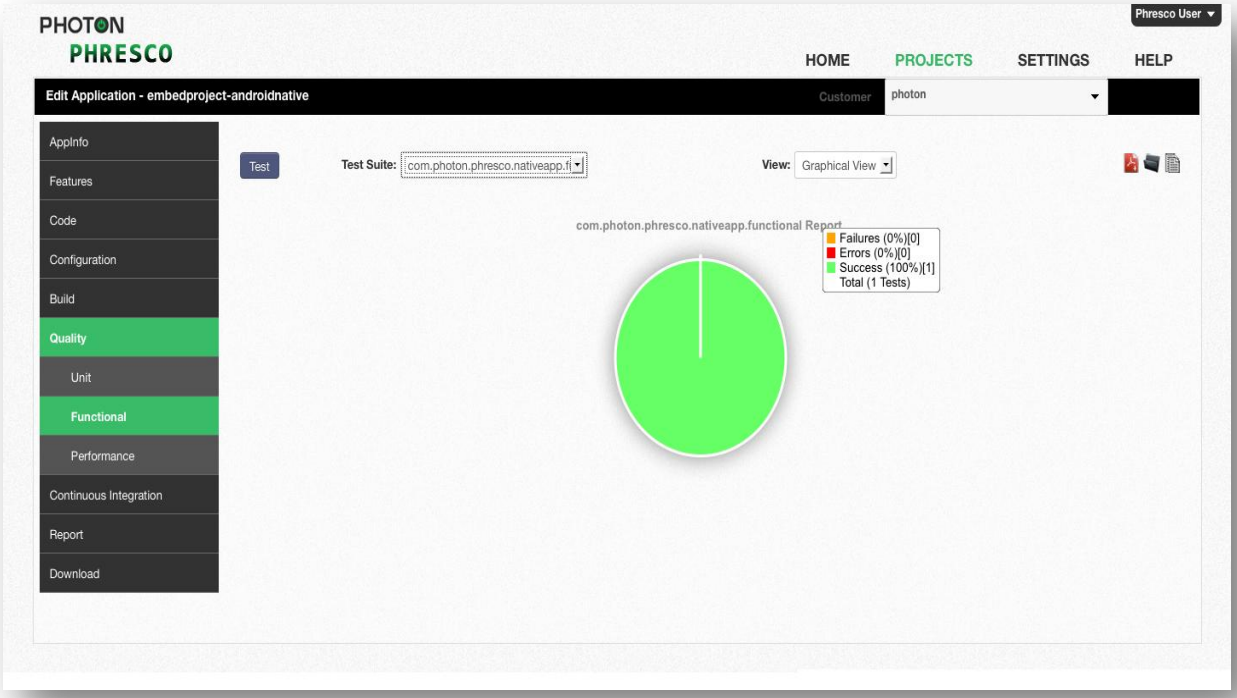

Figure 8-45: Android Native functional test report for single test case in graphical view

8.7 Functional Test cases for Android Hybrid – Monkey Talk

8.7.1 Structure of Functional Test for Android Hybrid

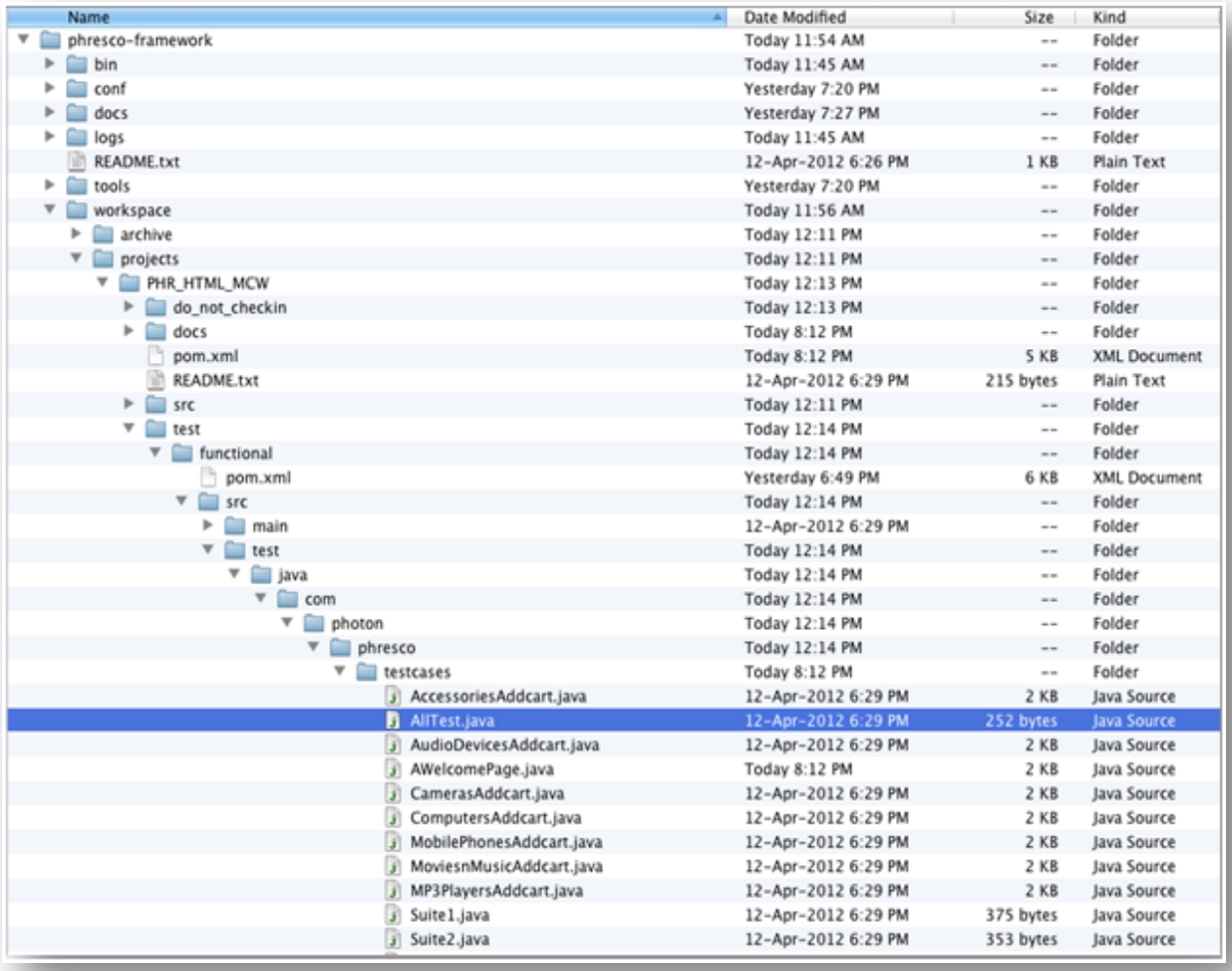

Figure 8-46: Android Hybrid functional tests structure

d. AllTest: This is a root mts file that can either call a suite of Test cases or an individual Test case.

e. Test Cases: These perform unique testing scenarios against the application.This is an mt file.

8.7.1.1 Existing Test Cases and Suites Out Of the Box in HeliOS

AllTest

In HeliOS testing Framework a mts suite file is named as “AllTest”.

The AllTest contains a bunch of test cases, each of which performs a unique scenario on the application. Test developers can start writing new suites and test cases by following the syntax and structure of HeliOS frameworks out of the box structure. Following is the AllTest file which shows up how to add / include the Test cases inside it.

```
Test Register Run %thinktime=5000 %timeout=2000
Test Login Run %thinktime=5000 %timeout=2000
Test Browse Run %thinktime=8000 %timeout=2000
```

Test cases

A Test case is a set of conditions under which a tester will determine whether an application is meeting the requirement. This helps the user to verify a particular functionality during the testing process. Elements can be identified and screen shots can be captured when the test fails.

```
Label Register Tap
Input regfirstname EnterText Mobile %thinktime=2000 %timeout=1000
Input reglastname EnterText mobiletablet %thinktime=2000 %timeout=1000
Input regemail EnterText mt@photoninfotech.net %thinktime=2000 %timeout=1000
Input #4 EnterText mypass %thinktime=2000 %timeout=1000
Input #5 EnterText 04425354565 %thinktime=2000 %timeout=1000
WebView * drag 308 394 295 397 2 -2
WebView * swipe Left %thinktime=10000
WebView * drag 75 21 57 26 2 1
WebView * swipe Left %thinktime=5000
```

8.7.1.2 To Add New Test Case in All Test

You can add a new test case to the Test suite using the following method. Here is an example for creating a new test case.

```
Test Register Run %thinktime=5000 %timeout=2000
Test Login Run %thinktime=5000 %timeout=2000
Test Browse Run %thinktime=8000 %timeout=2000
Test Billing Run %thinktime=10000 %timeout=2000
```

8.7.1.3 Report generated after execution

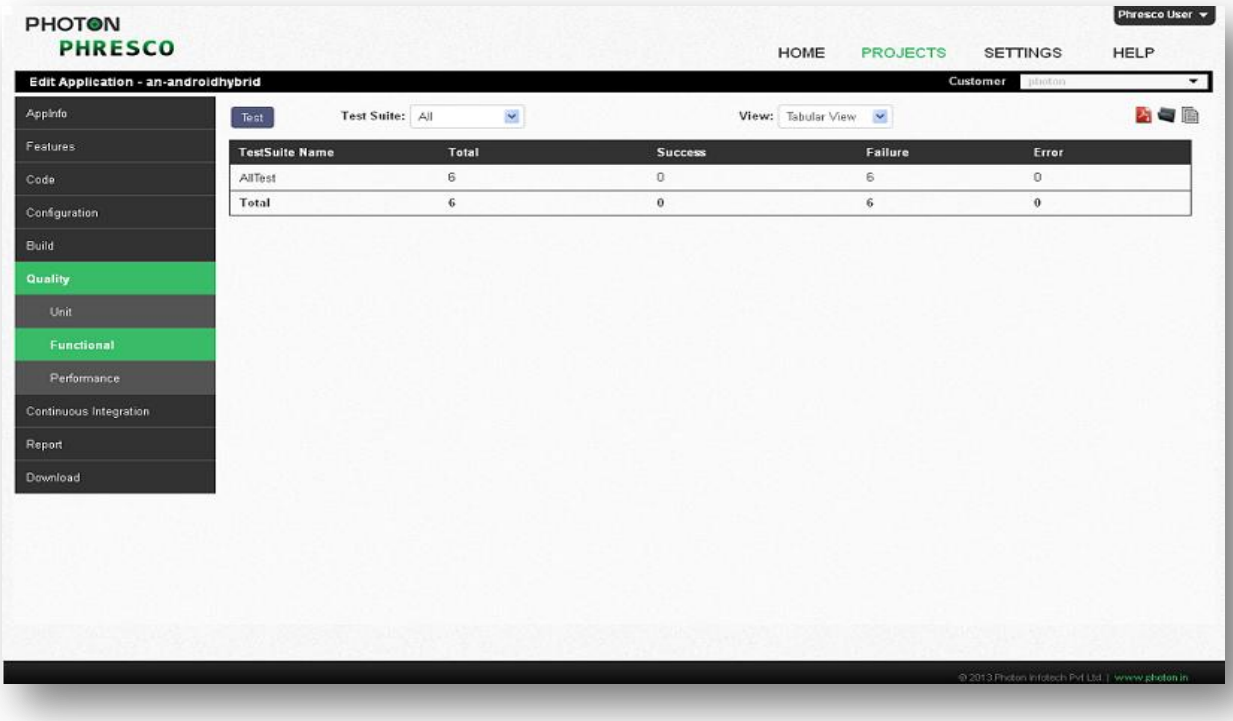

Figure 8-187: Android Hybrid functional test report for all test cases in tabular view

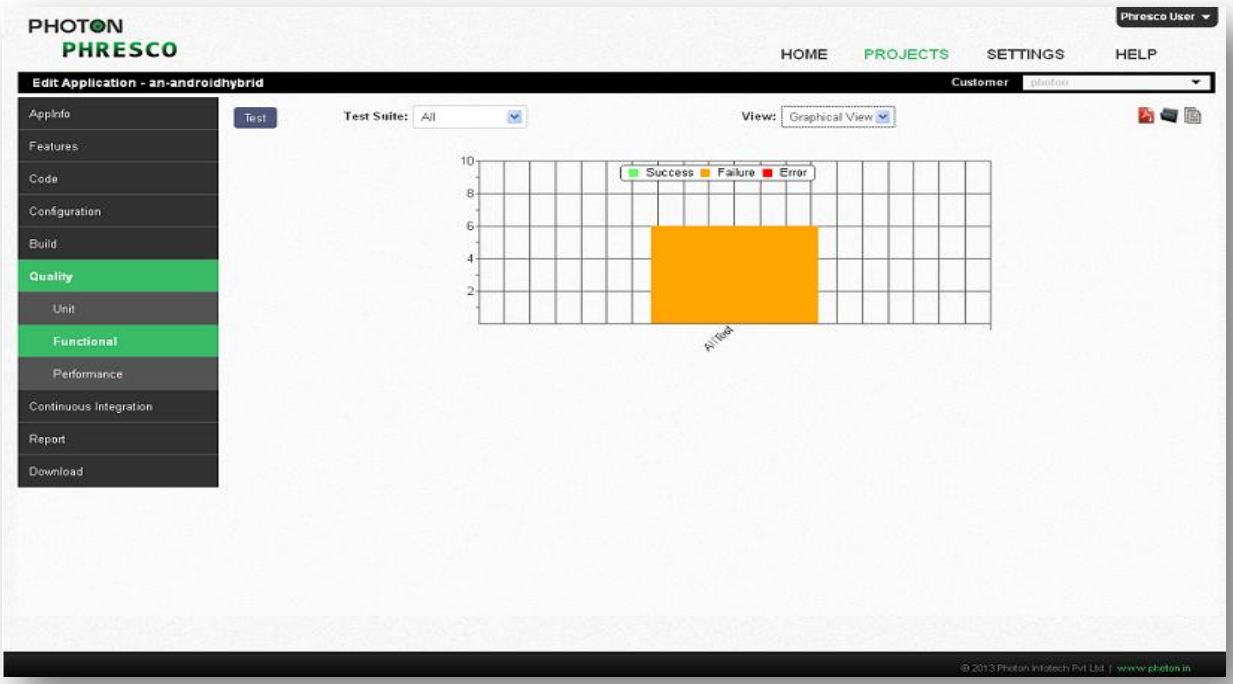

Figure 8-198: ***Android Hybrid functional test report for all test cases in graphical view***

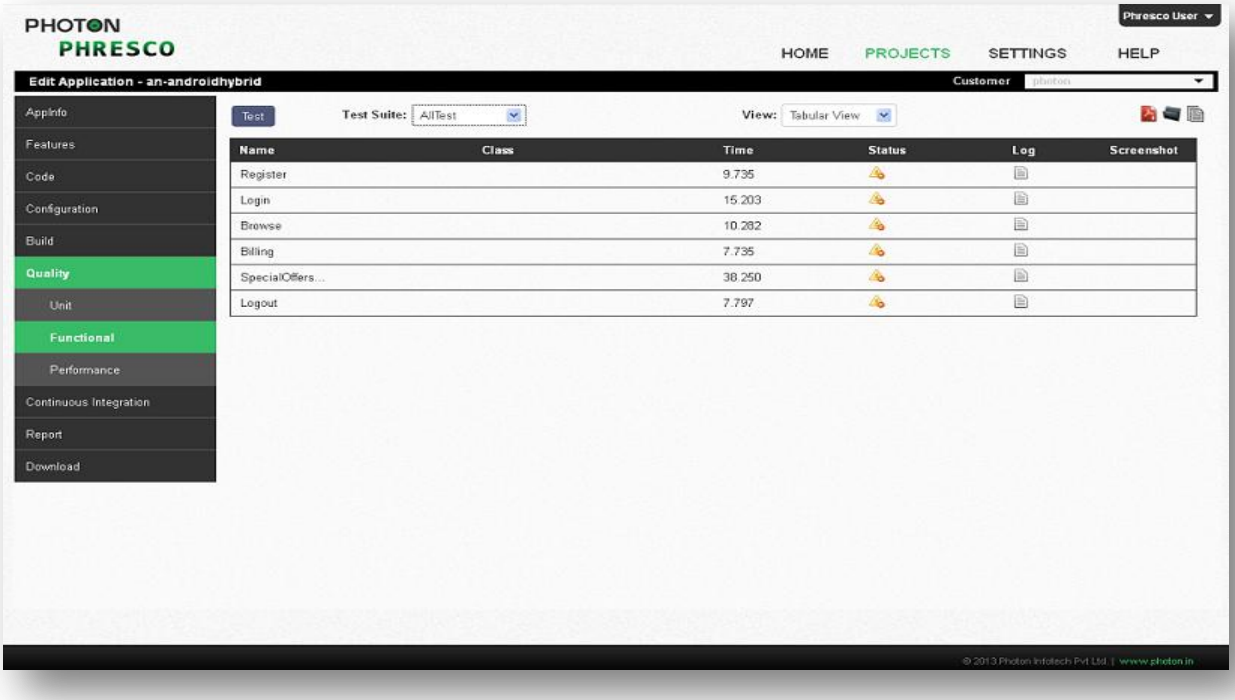

Figure 8-209: ***Android Hybrid functional test report for single test case in tabular view***

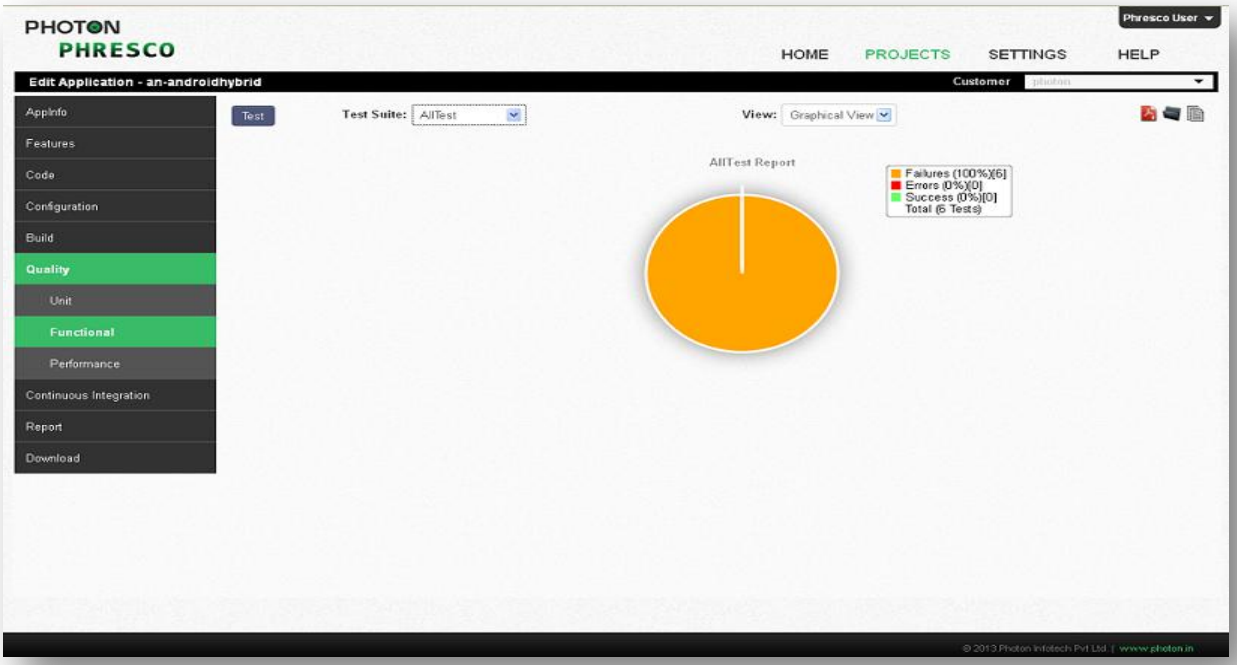

Figure 8-50: ***Android Hybrid functional test report for single test case in graphical view***

Limitations in Monkey Talk

- The Test case execution faces issues due to the element ID change when the auto popup key board pops up
- The Test case execution faces few problems on executing the same scripts in different resolutions

---

Note:

- In addition to this HeliOS also supports Calabash for Android application.

.

8.7.2 Performance Test for Android

HeliOS enables the users to test the performance for their android project. It triggers the test cases simultaneously on all the selected devices and once the tests are completed the results will be available on screen in tabular and graphical formats.

Steps to be followed to trigger performance test

1. Click the Applications-> Project created->Quality->Performance test->Test
2. A Performance Test pop up box appears and the number of android devices can be selected for testing.
3. When three devices are connected the performance testing occurs simultaneously as shown in figure:

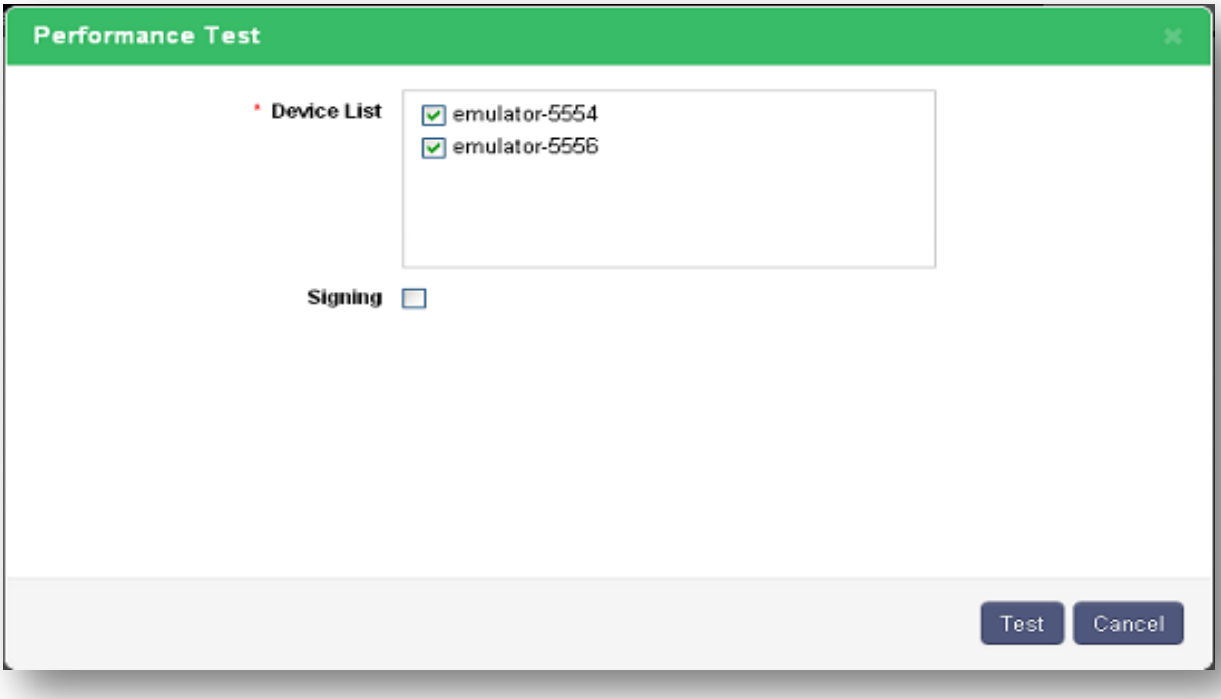

Figure 8-51: Choosing multiple devices for performance test.

Figure 8-52: Performance test in multiple devices

8.7.2.1 Report generated after execution

Report for the performance testing is generated in both graphical and tabular form. The response time and the throughput are given as the report after the performance testing.

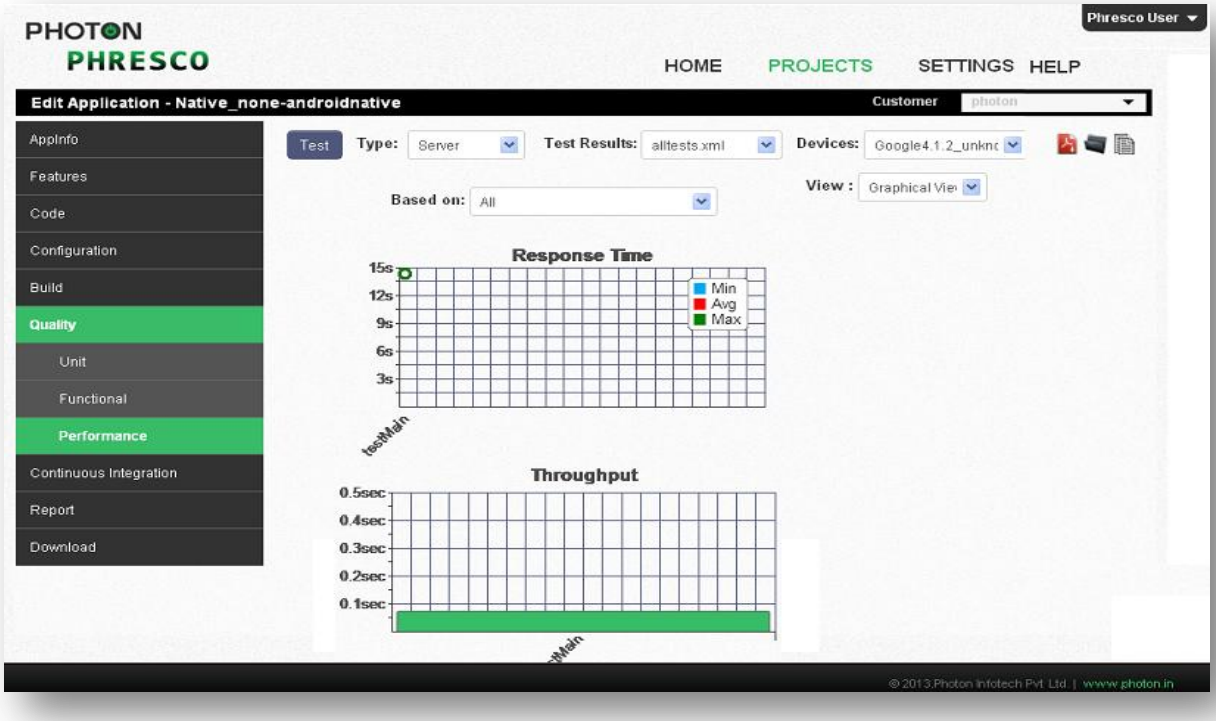

Figure 8-53: Android graphical report for performance test

8.8 iPhone Applications

8.8.1 Unit Test Case

Xcode offers two types of unit tests: logic tests and application tests.

Logic tests

This checks the exact functionality of a unit of code by itself without the help of application. Specific test cases can be put together to exercise a unit of code. Stress testing of the code can also be performed using logic tests. Logic tests run only on the simulators.

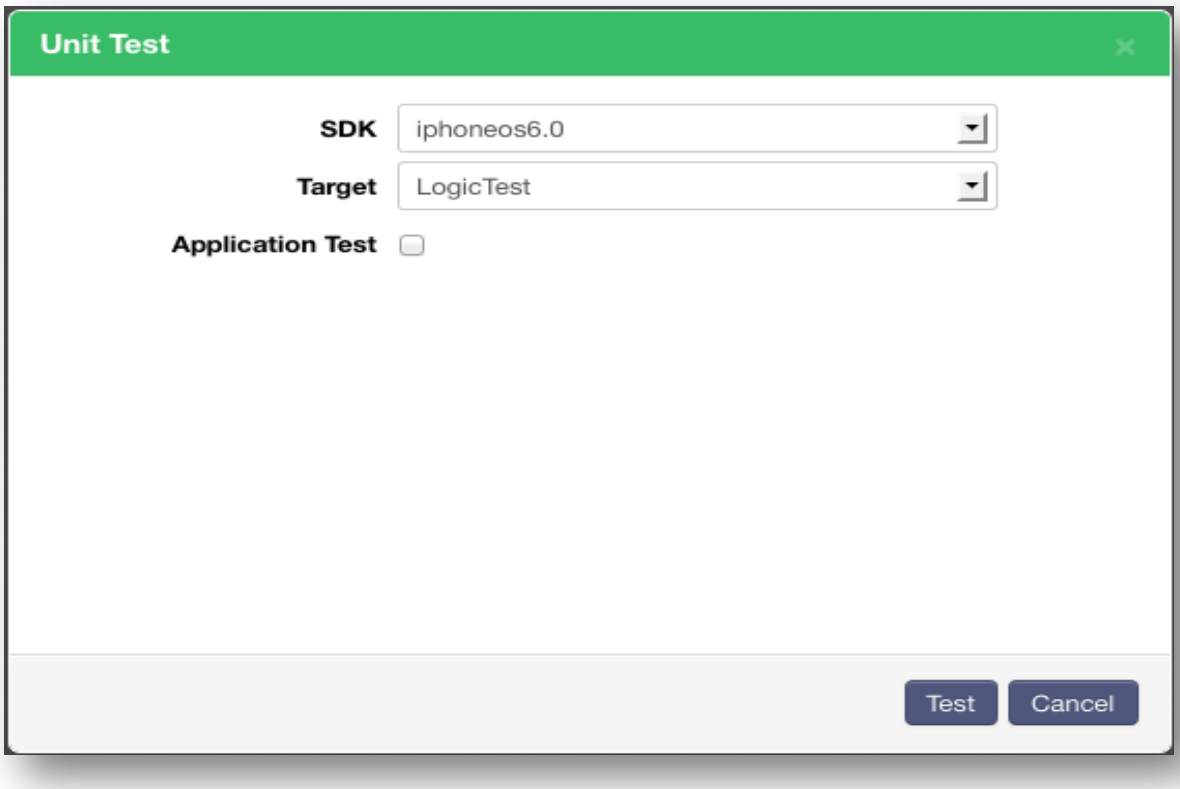

Figure 8-54: iPhone Unit test case

Application tests

This check the units of code in the context of the app. Application tests are used to test the connections of user interface controls and also to test the work of the controls and controller objects.

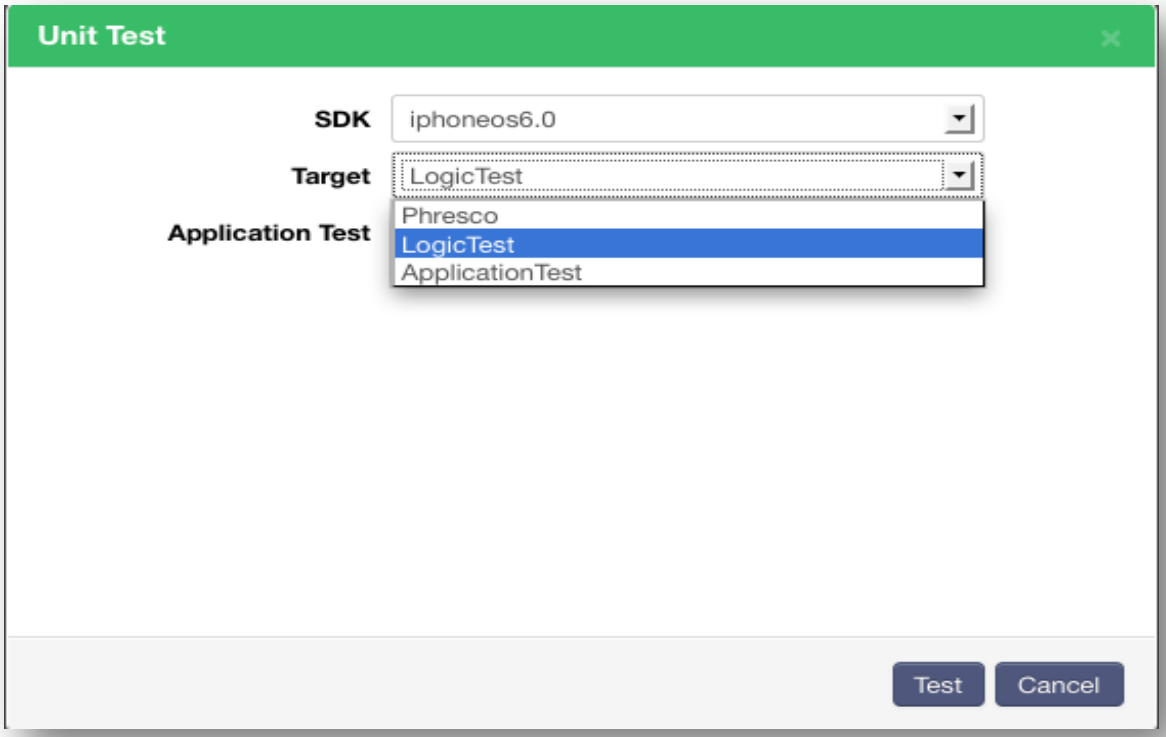

Figure 8-55: iPhone Unit test case

In order to run application test from Xcode 4.2 or 4.3 follow the below steps.

Step – 1

Following configurations are to be done for Unit test target Build settings

- In Other test flag field add -RegisterForSystemEvents
- In BundleLoader field add $(BUILT_PRODUCTS_DIR)/Phresco.app/Phresco
- In TestHost field add $(BUNDLE_LOADER)
- In Test After Build field add Yes
- In Header search path field add usr/include/libxml2

In Target Dependencies

- Add Phresco.app in target dependencies of unit test target.

Step -2

In Xcode 4.2

Modify the file “RunPlatformUnitTests” which is available in the path "/Developer/Platforms/iPhoneSimulator.platform/Developer/Tools".

If loop in main method file should be replaced with the below text

```
mkdir -p "${BUILT_PRODUCTS_DIR}/Documents"
mkdir -p "${BUILT_PRODUCTS_DIR}/Library/Caches"
mkdir -p "${BUILT_PRODUCTS_DIR}/Library/Preferences"
mkdir -p "${BUILT_PRODUCTS_DIR}/tmp"

export CFFIXED_USER_HOME="${BUILT_PRODUCTS_DIR}/"

RunTestsForApplication "${TEST_HOST}" "${TEST_BUNDLE_PATH}"
```

In Xcode4.3,

Modify the file “RunPlatformUnitTests” which is available in the path "/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/Tools".

Note:

- For logical test, Step 2 should not be considered.
- In Xcode4.5, the Other test flag field should be kept empty.

8.8.1.1 Structure of Alltest and Test Cases

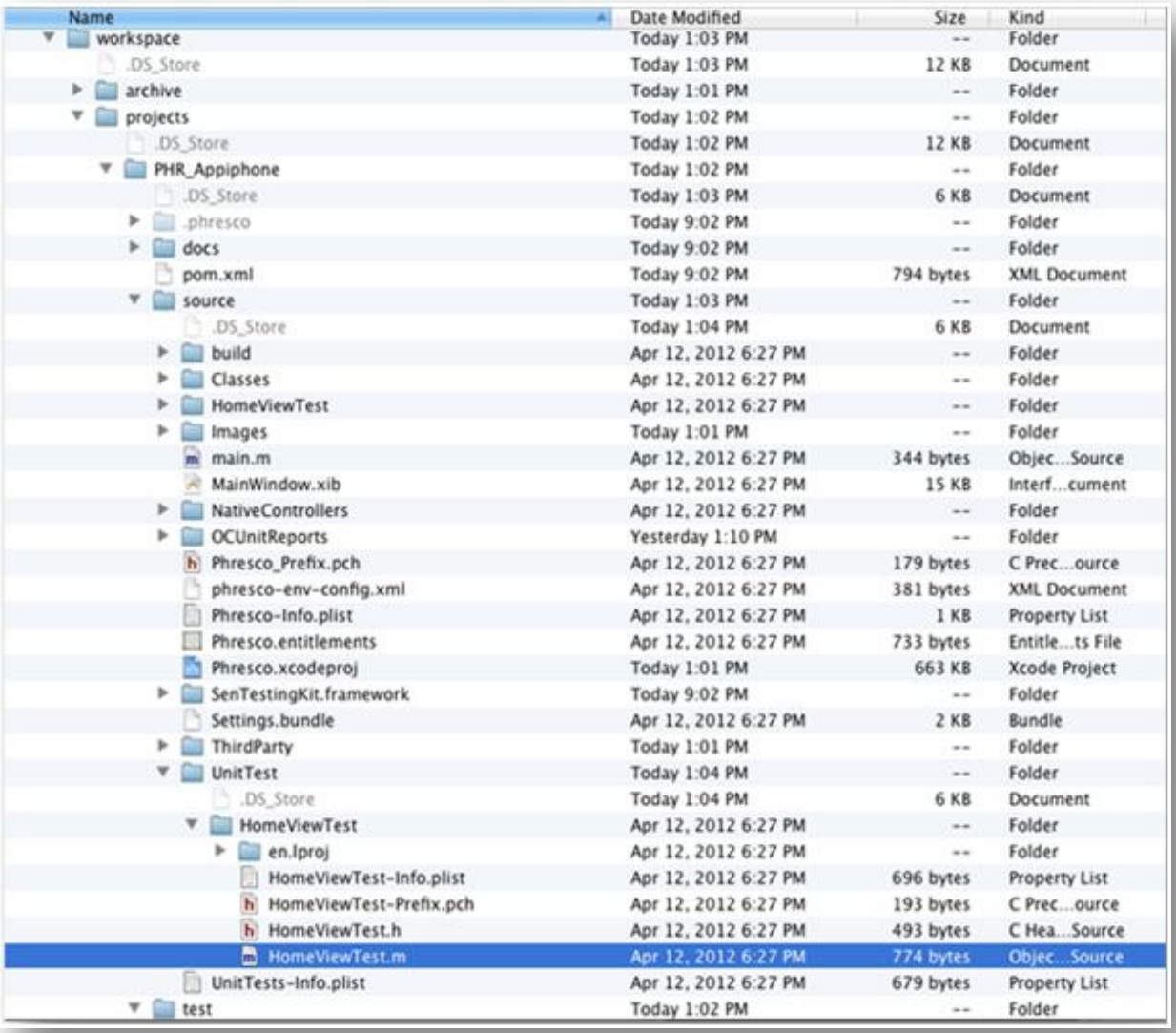

Figure 8-56: iPhone unit tests structure

a. **Home view test:** This is a test suite in which all other test cases are called.

b. **Test cases:** Test cases are present inside the test suites

Homeviewtest for iPhone Application

The testSuite contains a bunch of test cases, each of which performs a unique scenario on the application. Test developers can start writing new suite classes and test cases by following the syntax and structure of HeliOS framework's out of the box class structure. HomeViewTest class calls all the suite classes and the test cases within it. Once suite classes and test cases are written developers can execute those from HeliOS Framework and can see the report. Following is the HomeViewTest files which shows up how to add / include the class inside it.

```
#import <SenTestingKit/SenTestingKit.h>
#import <UIKit/UIkit.h>

#import "PhrescoAppDelegate.h"
#import "RootViewController.h"
#import "HomeViewController.h"
#import "BrowseViewController.h"
#import "ResultViewController.h"
#import "ProductDetailsViewController.h"
#import "AddToBagViewController.h"
#import "ViewCartController.h"
#import "CheckOutViewController.h"
#import "CheckOutOVerallViewController.h"
#import "ReviewViewController.h"
#import "ReviewCommentsViewController.h"
#import "LoginViewController.h"
#import "RegistrationViewController.h"
HomeViewTest.h

@interface HomeViewTest : SenTestCase{

@private
iShopAppDelegate *appDelegate;
RootViewController *rootController;
HomeViewController *homeController;
BrowseViewController *browseController;
ResultViewController *resultController;
ProductDetailsViewController *pdtDetailController;
AddToBagViewController *addCartController;
ViewCartController *viewCartController;
CheckOutViewController *checkViewController;
CheckOutOVerallViewController *checkOverallController;
ReviewViewController *reviewController;
```

```
ReviewCommentsViewController *reviewCommentsController;
LoginViewController *loginController;
RegistrationViewController *registerController;


    UITableView *tblView;
}
-(void) testAction;
@end

//////

- (void)testExample{

    [rootController tabBarButtonAction:@"0"];
  // STFail(@"Unit tests are not implemented yet in HomeViewTest");
}
-(void) testAction{

    [homeController buttonAction:@"0"];

}
-(void) testBrowse{


    //[browseController  tableView:tblView didSelectRowAtIndexPath:0];

    STAssertThrows([browseController tableView:tblView
didSelectRowAtIndexPath:-1] , @"Invalid index");


}
-(void) testProductResult{

    [resultController  tableView:tblView didSelectRowAtIndexPath:0];
    [resultController reviewButtonSelected:@"0"];

}
-(void) testProductDetail{

    [pdtDetailController addToCart:@"0"];

}
```

```
-(void) testAddToCart{

  [addCartController removeIndex:@"0"];

}
-(void) testViewCart{

   [viewCartController browseButtonSelected:@"0"];

}
-(void) testCheckCart{

   [checkViewController cancelAction:@"0"];

}
-(void) testCheckOverall{

   [checkViewController reviewAction:@"0"];

}
-(void) testReview{

   [reviewController tableView:tblView didSelectRowAtIndexPath:0];

}
-(void) testComments{

   [reviewCommentsController goBack:0];

}

-(void) testLogin{

   [loginController registerButtonSelected:0];

}

-(void) testRegister{

   [registerController registerButtonSelected:0];

}
@end
```

Test Register Example for a Test Case

A test case is a set of conditions under which a tester will determine whether an application is meeting the requirement. This helps the user to verify a particular functionality during the testing process.

The following is the example of a test register

```
-(void) testRegister{

   [registerController registerButtonSelected:o];
}
```

8.8.1.2 To Add New Test Case in Homeviewtest

You can add a new test case to the Homeviewtest using the following method. Here is an example for creating a new test case 'testsploffers'.

```
-(void) testSplOffers{

   [splOfferController tableView:tblView
didSelectRowAtIndexPath:o];

}
```

8.8.1.3 Report generated after execution

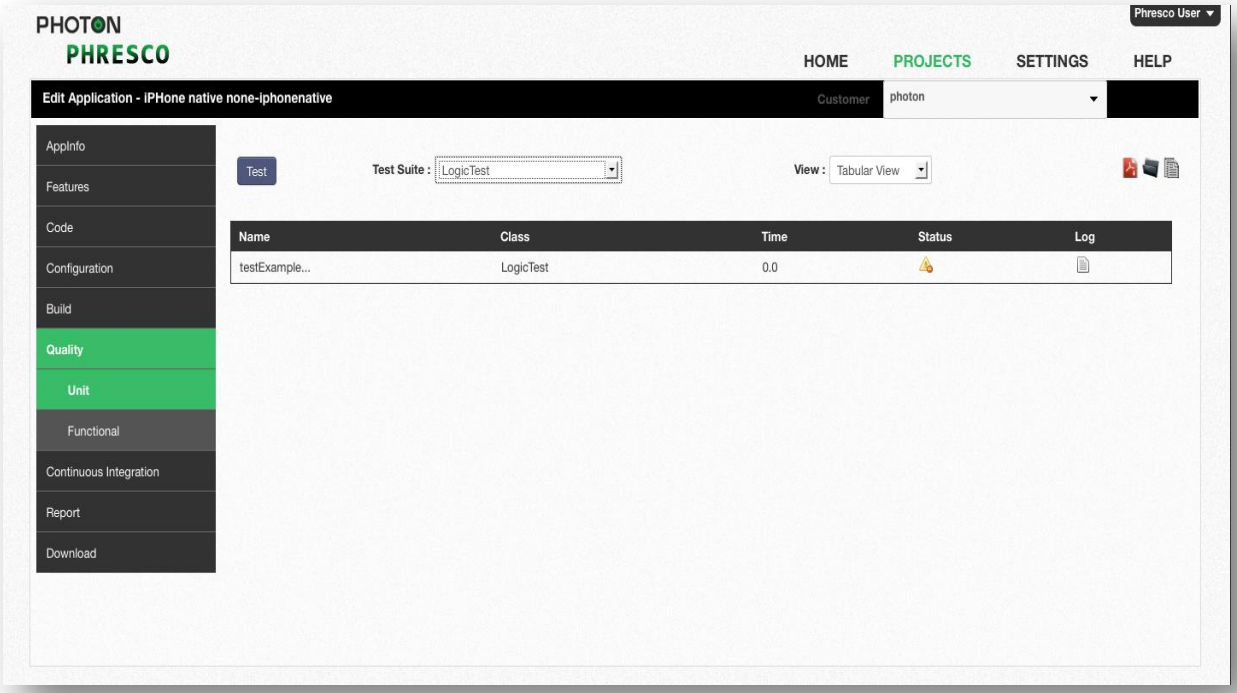

Figure 8-57: ***iPhone unit test report for single test case in tabular view***

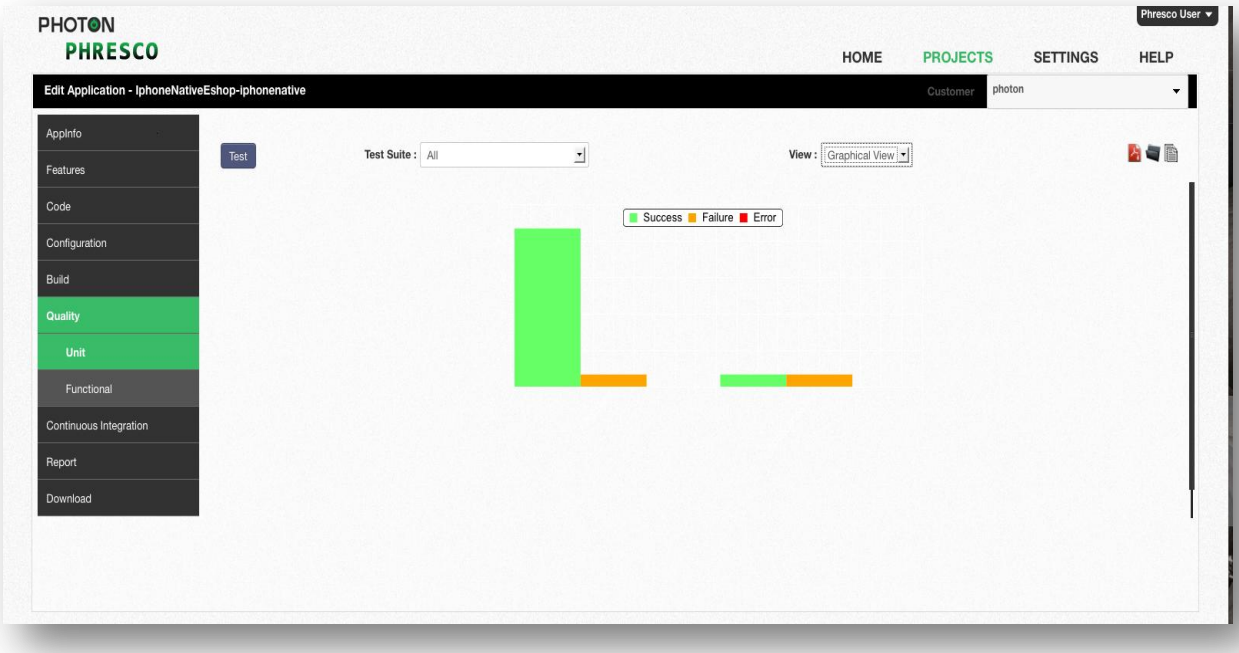

Figure 8-58: iPhone unit test report for all test cases in graphical view

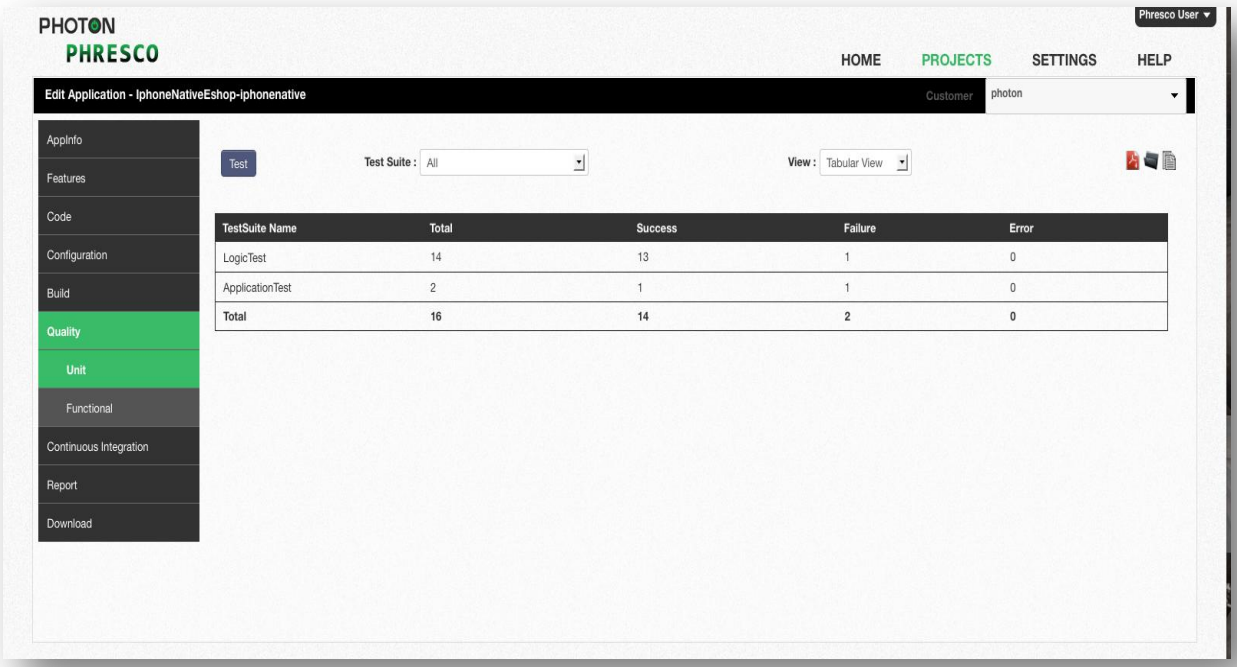

Figure 8-219: ***iPhone unit test report for all test cases in tabular view***

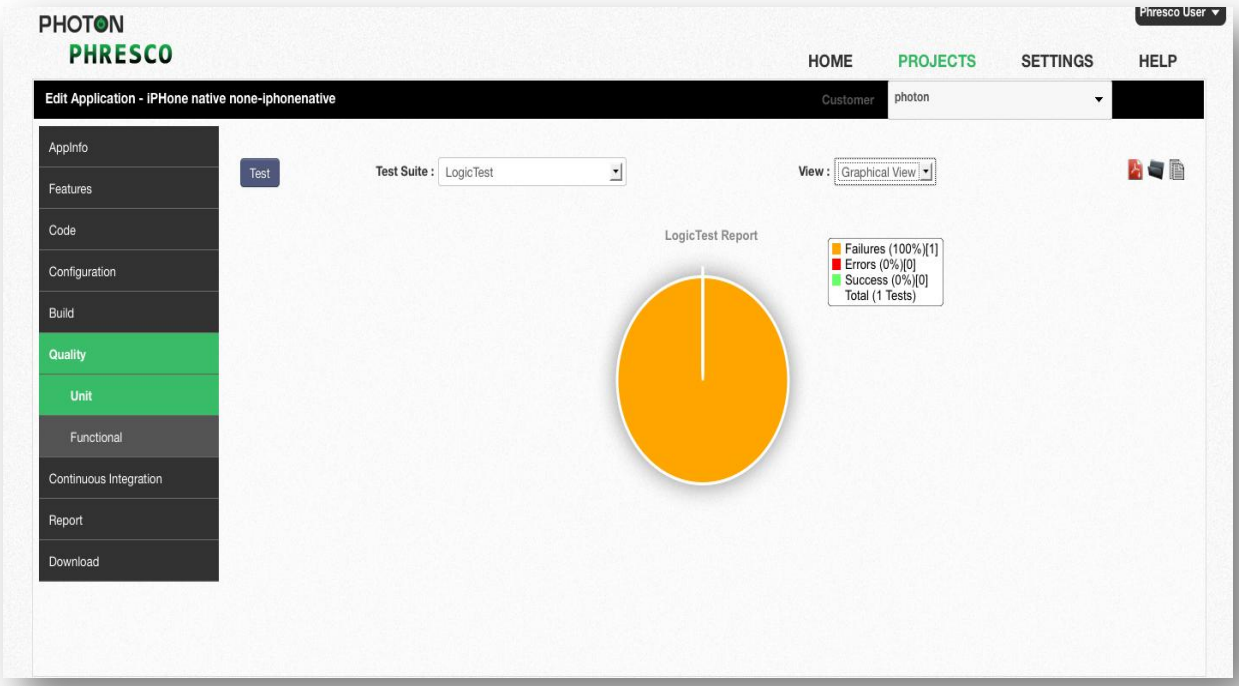

Figure 8-60: ***iPhone unit test report for single test case in graphical view***

8.8.2 Functional Test Case

8.8.2.1 Structure of Alltest and Test Case

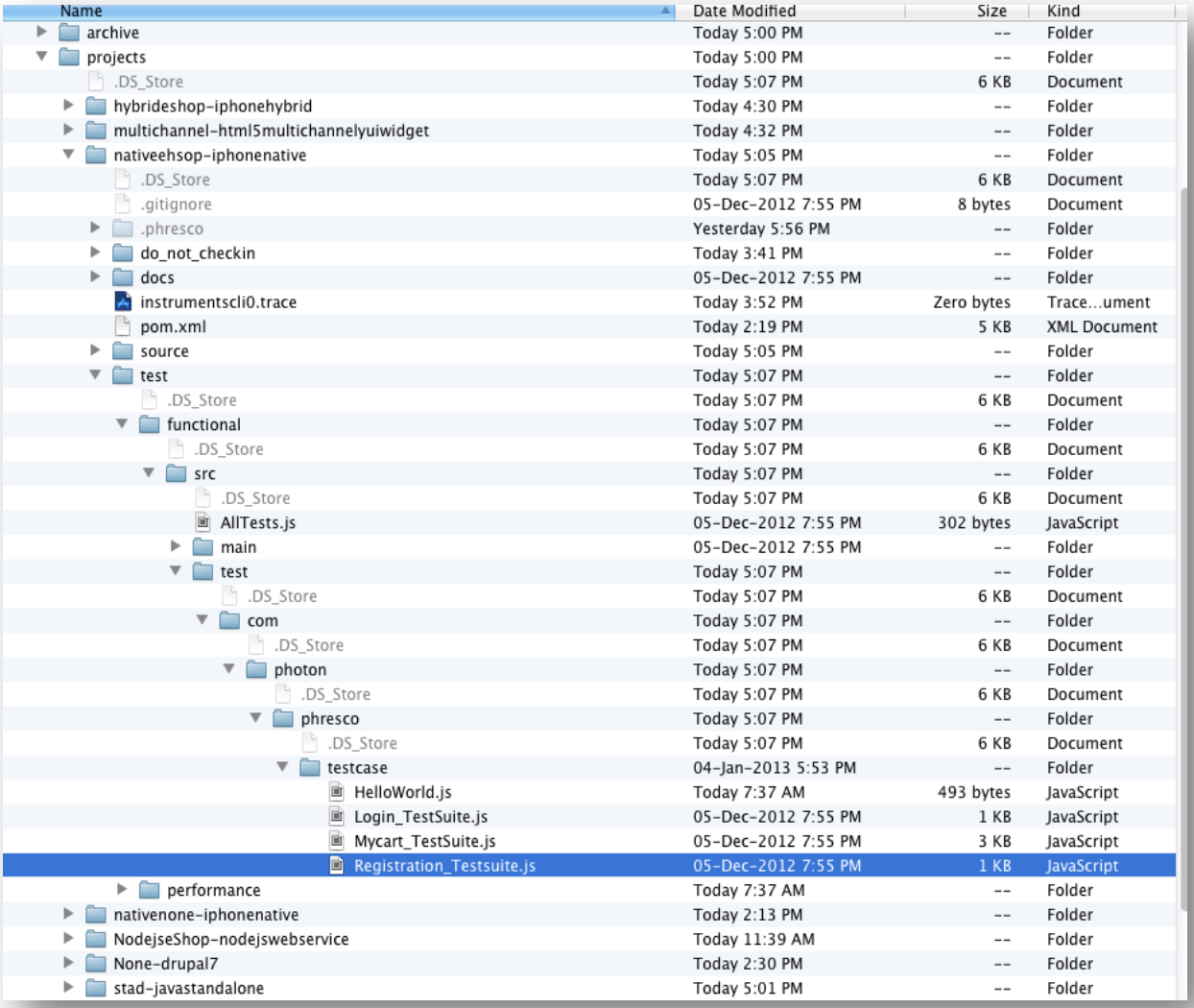

Figure 8-61: iPhone functional tests structure

a. **AllTest**: Alltest is the root file that carries the Test suites.

b. **Test Suite:** This is the js file which contains multiple test cases.

c. **Test cases:** These perform unique testing scenarios against the application.

8.8.2.2 Existing Test Cases and Suites Out Of the Box in HeliOS

All Test for iPhone

AllTest calls all the suite files and the test cases within it. Once suite files and test cases are written developers can execute those from HeliOS Framework and can see the report. Following is the AllTest file which shows up how to add / include the suite file inside it.

```
#import "main/com/photon/phresco/util/MainActivity.js"
#import "main/com/photon/phresco/util/UIElements.js"
#import "test/com/photon/phresco/testcase/AudioDevice_suite.js
#import "test/com/photon/phresco/testcase/Television_suite.js"
```

Test suite for iPhone

Testsuite is a js file which holds all the other test cases. Functional test runs in the order by which the test suites are created. The Testsuite contains a bunch of test cases, each of which performs a unique scenario on the application. Test developers can start writing new suite file and test cases by following the syntax and structure of HeliOS framework's out of the box structure. Test suite contains the test cases within it. Once suite files and test cases are written testers can execute those from HeliOS Framework and can see the report. Following is the Test suite file which shows up how to add / include the Test cases inside it.

```
#import "../../../../../main/com/photon/phresco/util/MainActivity.js"
#import "../../../../../main/com/photon/phresco/util/UIElements.js"
function testAudioDevice(testname){
    try{
       clickOnScroll(Browse_id);
       clickOnScroll(AudioDevice_id);
       clickOnScreen(110,127);
       clickOnScreen(184,211);
       clickOnScreen(259,223);
       clickOnScroll(UpdateCart_id);
       waitForFewSeconds(1);
       clickOnScroll(Checkout_id);
       clickOnScreen(259,223);
       waitForFewSeconds(1);
       clickOnScroll(MyCart_id);
       clickOnScreen(259,223);
       waitForFewSeconds(1);
       clickOnScroll(Remove_id);
       clickOnScroll(Back_id);
       UIALogger.logPass(testname);
      }
        catch(error){
        captureScreenshot(testname);
        //UIALogger.logFail("Fail");
        UIALogger.logError(testname);
      }
}
testAudioDevice("AudioDeviceTest");
```

To Add New Test Suite in All Test

You can add a new test suite to the Main Test Suite using the following method.

```
#import "main/com/photon/phresco/util/MainActivity.js"
#import "main/com/photon/phresco/util/UIElements.js"
#import "test/com/photon/phresco/testcase/AudioDevice_suite.js
#import "test/com/photon/phresco/testcase/Television_suite.js"
#import "test/com/photon/phresco/testcase/Computer_suite.js"
```

To Add New Test Case in Test Suite

You can add a new test case to the Test suite using the following method.

```
#import "../../../../../main/com/photon/phresco/util/MainActivity.js"
#import "../../../../../main/com/photon/phresco/util/UIElements.js"
function testAudioDevice(testname){
    try{
        clickOnScroll(Browse_id);
        clickOnScroll(AudioDevice_id);
        clickOnScreen(110,127);
        clickOnScreen(184,211);
        clickOnScreen(259,223);
        clickOnScroll(UpdateCart_id);
        waitForFewSeconds(1);
        clickOnScroll(Checkout_id);
        clickOnScreen(259,223);
        waitForFewSeconds(1);
        clickOnScroll(MyCart_id);
        clickOnScreen(259,223);
        waitForFewSeconds(1);
        clickOnScroll(Remove_id);
        clickOnScroll(Back_id);
        UIALogger.logPass(testname);
      }
function testTelevision(testname){
     try{
        clickOnScroll(Browse_id);
        clickOnScroll(Television_id);
        waitForFewSeconds(1);
        clickOnScreen(110,127);
        clickOnScreen(184,211);
        clickOnScreen(259,223);
        waitForFewSeconds(1);
        clickOnScroll(UpdateCart_id);
        clickOnScroll(Checkout_id);
        waitForFewSeconds(1);
        clickOnScreen(259,223);
        clickOnScroll(MyCart_id);
         clickOnScreen(259,223);
```

```
            clickOnScroll(Remove_id);
             waitForFewSeconds(1);
             clickOnScroll(Back_id);
            UIALogger.logPass(testname);
            }
          catch(error){
                    captureScreenshot(testname);
                    //UIALogger.logFail("Fail");
                  UIALogger.logError(testname);
          }
}
testAudioDevice("AudioDeviceTest");
testTelevision("TelvisionTest");
```

8.8.2.3 Report generated after execution

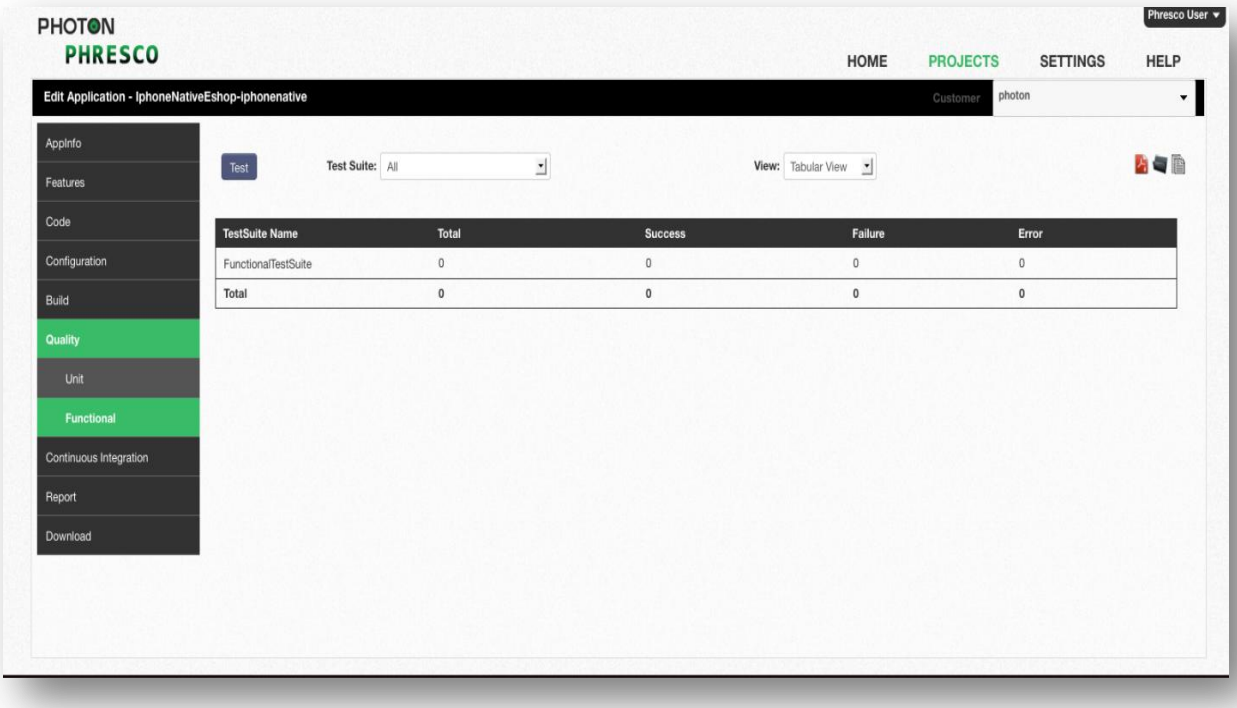

Figure 8-62: ***iPhone functional test report for single test case in tabular view***

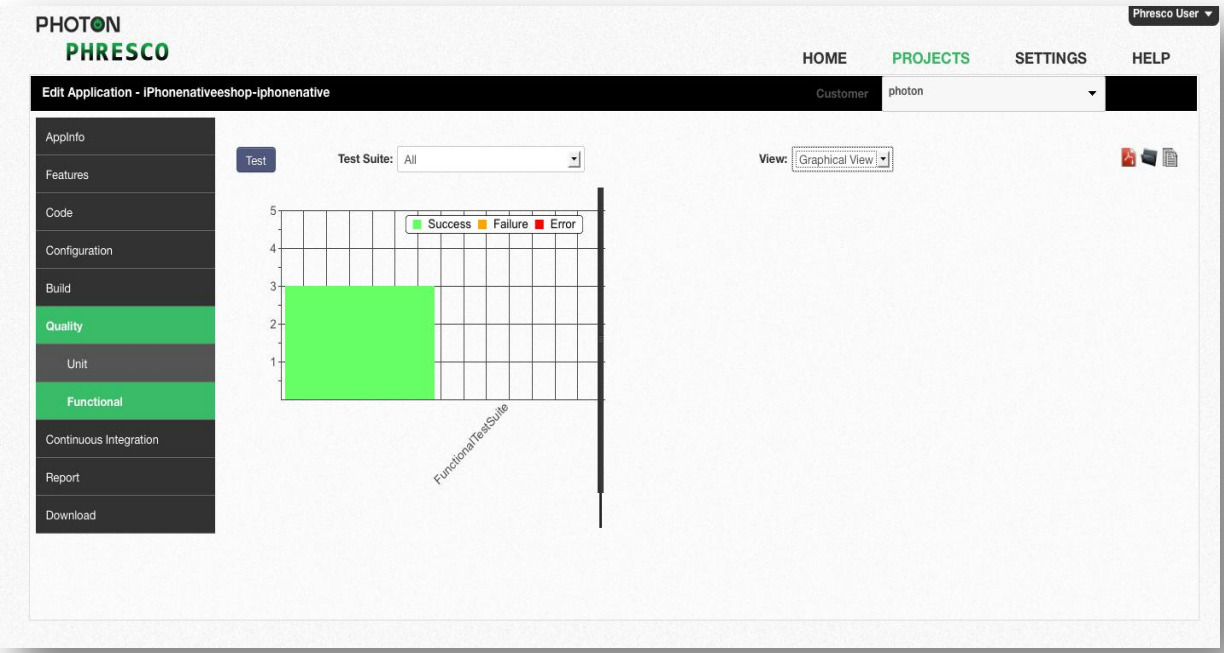

Figure 8-63: ***iPhone functional test report for all test cases in graphical view***

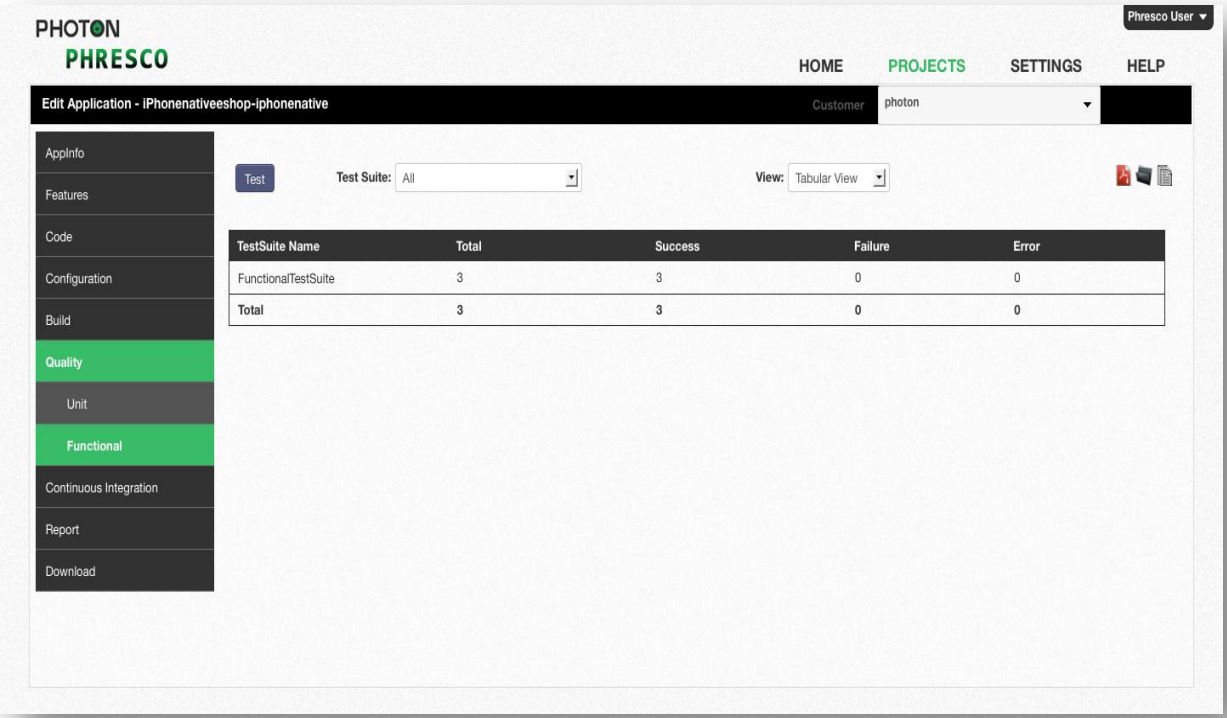

Figure 8-64: ***iPhone functional test report for all test cases in tabular view***

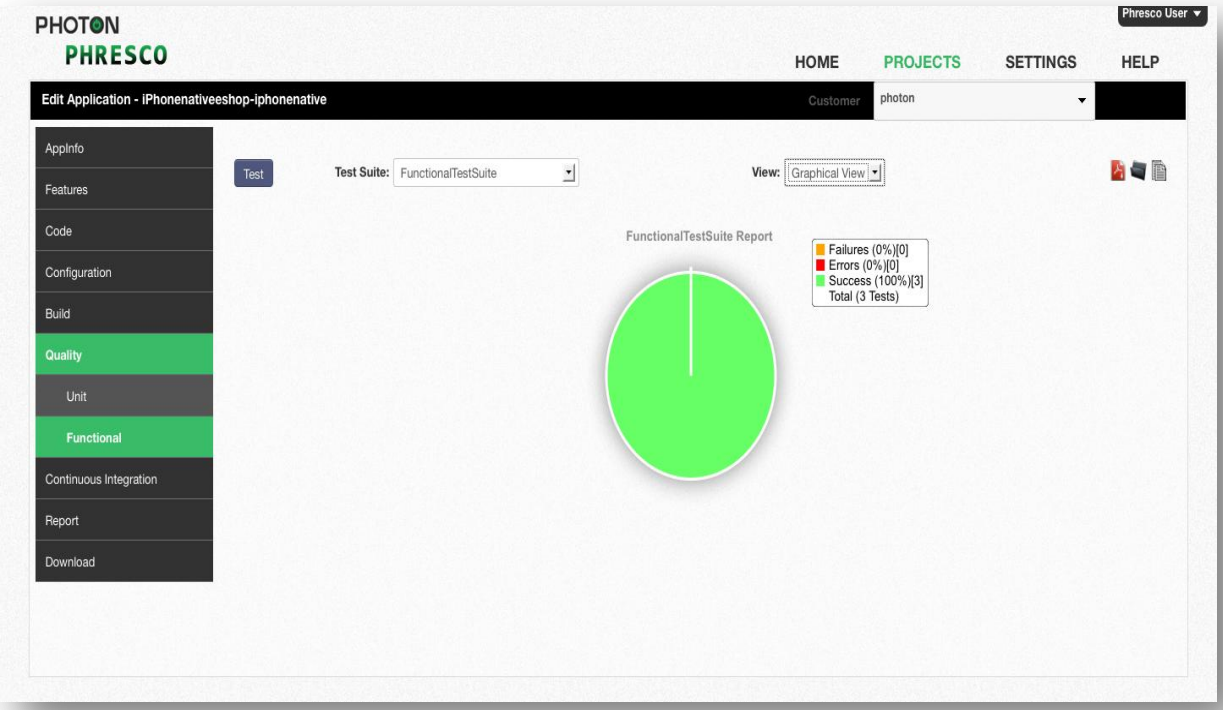

Figure 8-65: ***iPhone functional test report for single test case in graphical view***

Note:

- In addition to this,HeliOS supports Calabash for iPhone application.

8.9 Node js Test Cases

8.9.1 Unit Test Case

8.9.1.1 Structure of Test Suites and Test Case

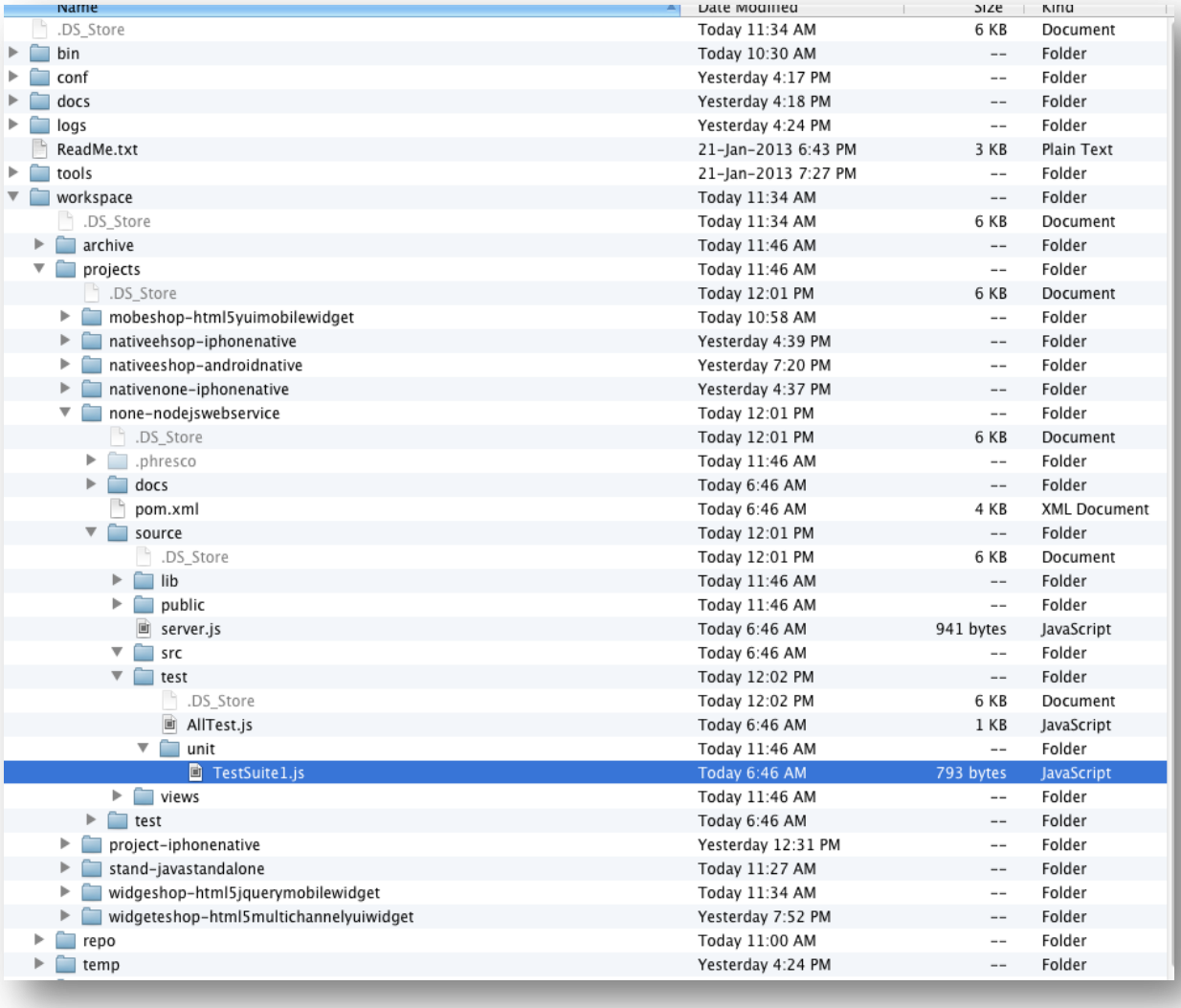

Figure 8-66: Unit test folder structure for NodeJs

b. **AllTest**: Alltest is the root file that carries the Test suite.

c. **Test suite**: Test suite is made to run through AllTest

d. **Test case**: All the Test cases should be added within the test suite.

8.9.1.2 Existing Test Cases and Suites Out Of the Box in HeliOS

Alltest for Node js

AllTest is the root category which holds the test suite. AllTest contains a bunch of test suites, each of which performs a unique scenario on the application. Test developers can start writing new suite classes by following the syntax and structure of HeliOS framework's out of the box class structure. Once test suite classes and test cases are written developers can execute those from HeliOS Framework and can see the report. Following is the AllTest file which shows up how to add / include the class inside it.

```
var nodeunit = require('nodeunit');
var reporter = nodeunit.reporters.junit;

var opts = {
   "error_prefix": "\u001B[31m",
   "error_suffix": "\u001B[39m",
   "ok_prefix": "\u001B[32m",
   "ok_suffix": "\u001B[39m",
   "bold_prefix": "\u001B[1m",
   "bold_suffix": "\u001B[22m",
   "assertion_prefix": "\u001B[35m",
   "assertion_suffix": "\u001B[39m"
}

opts.output = "target/surefire";
reporter.run(['source/test/eshop'], opts);
```

Test Suites example

Test suite is a collection of test cases that are intended to be used to test a project code to show that it has some specified set of behaviors. A test suite contains detailed instructions or goals for each collection of test cases. New test cases can also be added by using the following syntax.

Test case example

A test case is a set of conditions under which a tester will determine whether an application is meeting the requirement. This helps the user to verify a particular functionality during the testing process.

```
  exports.testSomething = function(test){
   test.expect(1);
   test.ok(true, "this assertion should pass");
   test.done();
};

  exports.testSomethingElse = function(test){
   test.ok(true, "this assertion should fail");
   test.done();
};
```

8.9.1.3 To Add New Test Case in Test Suite

You can add a new test case to the Test suite using the following method.

```
  exports.testSomething = function(test){
    test.expect(1);
    test.ok(true, "this assertion should pass");
    test.done();
 };

  exports.testSomethingElse = function(test){
    test.ok(true, "this assertion should fail");
    test.done();
 };
```

8.9.1.4 Report generated after execution

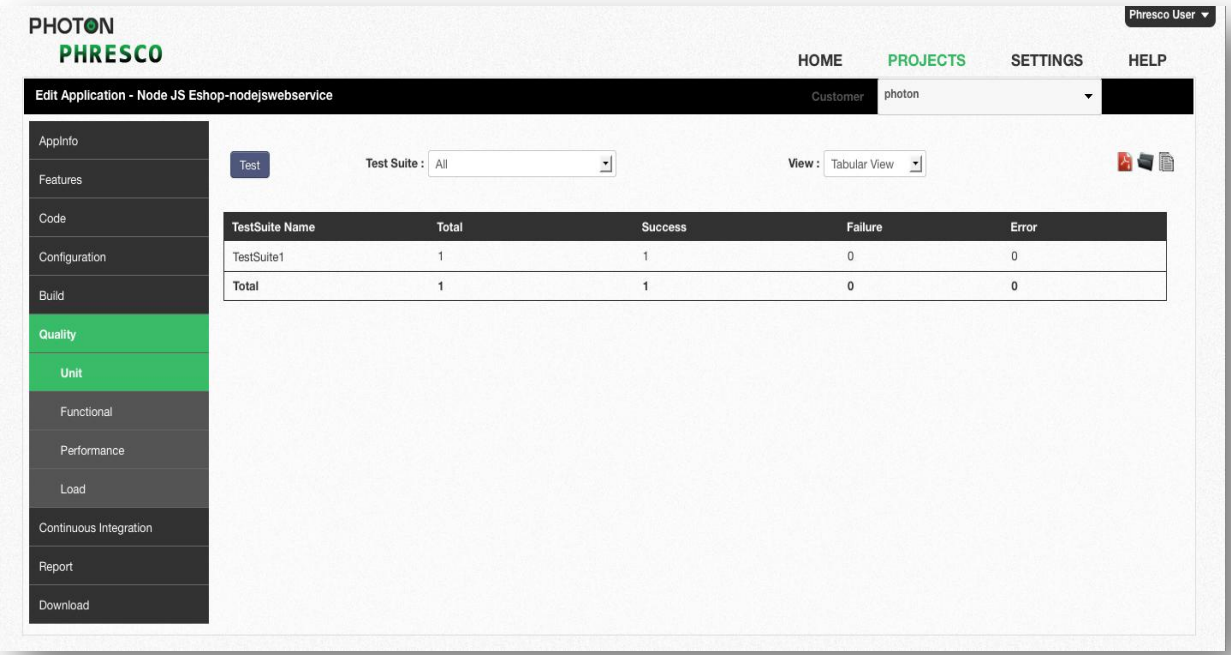

Figure 8-22: NodeJS unit test report for all test cases in tabular view

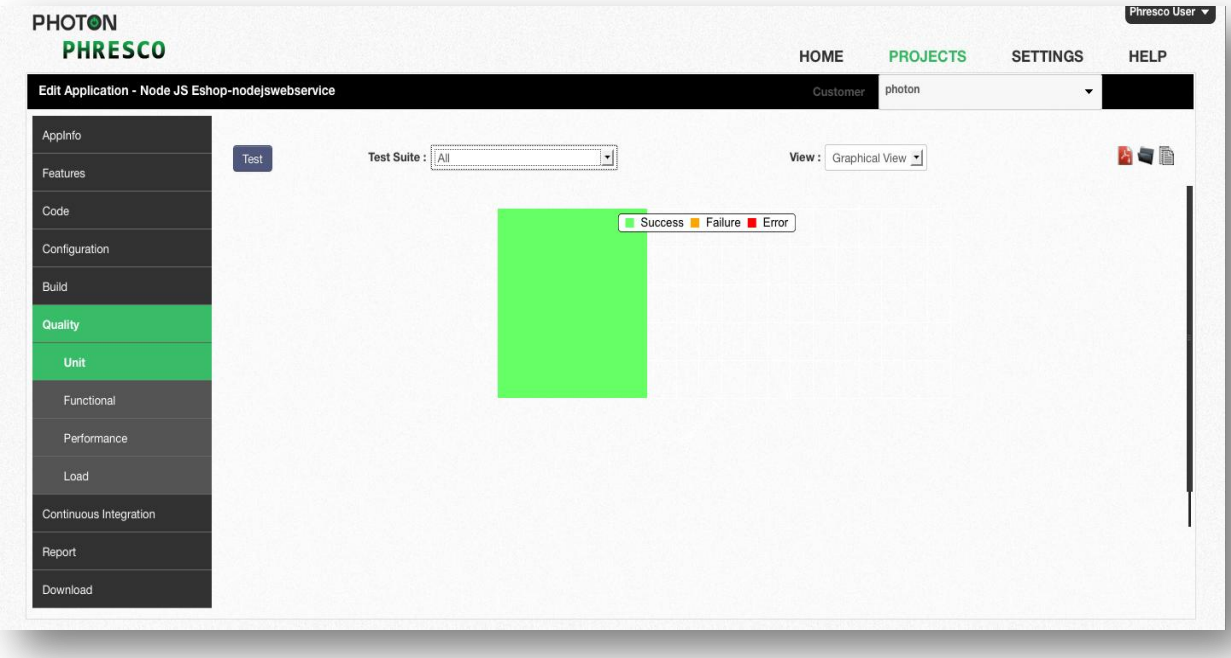

Figure 8-23: NodeJS unit test report for all test cases in graphical view

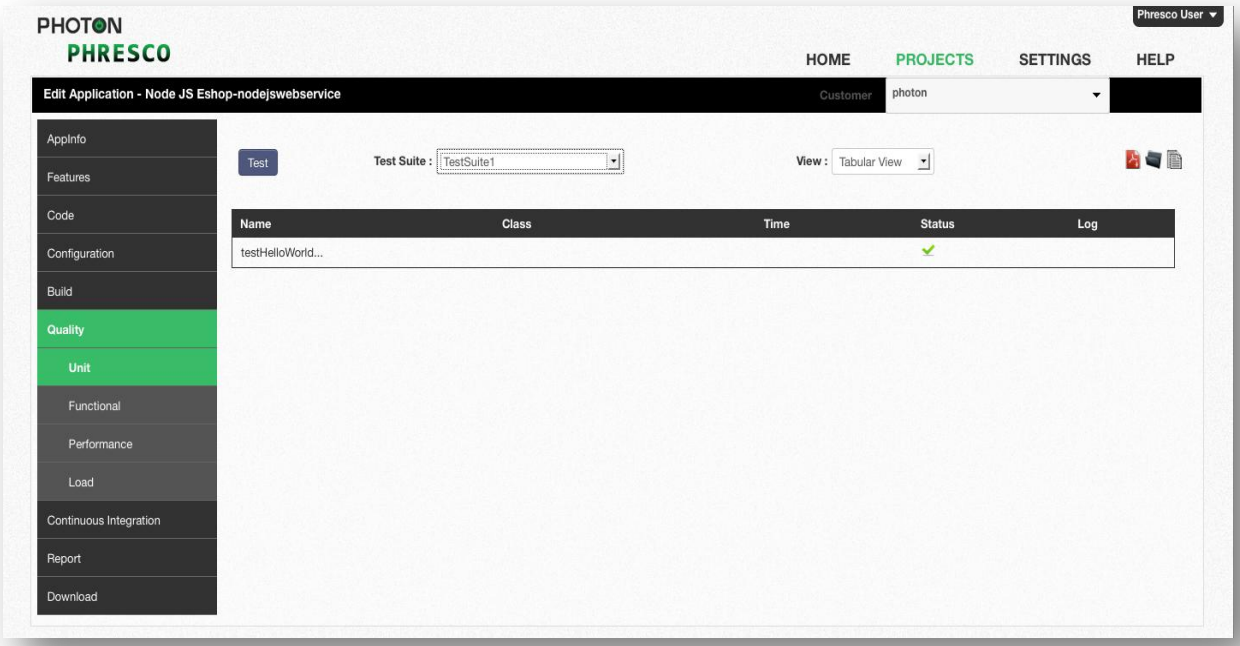

Figure 8-24: ***NodeJS unit test report for single test case in tabular view***

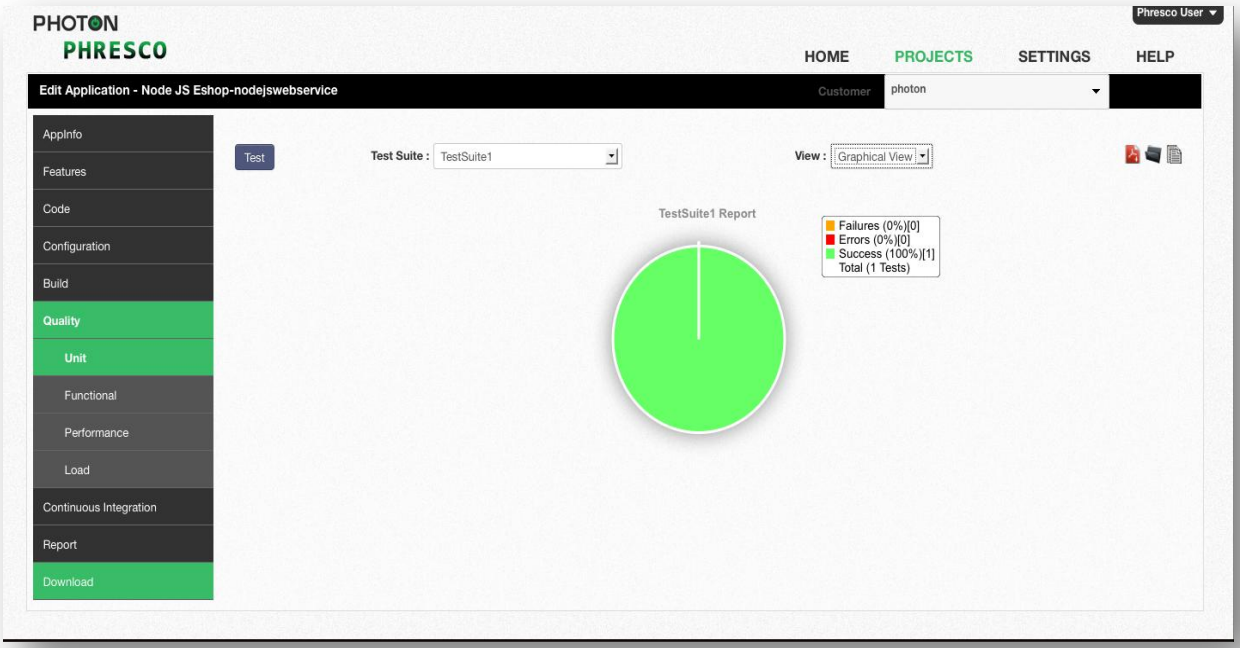

Figure 8-25: ***NodeJS unit test report for single test case in graphical view***

Note

- Refer section 8.1.2 for the Functional Test case structure which is similar to Selenium Grid Functional Test structure.

8.10 JQuery Test Cases

8.10.1 Unit Test Cases

8.10.1.1 Structure of Test Case

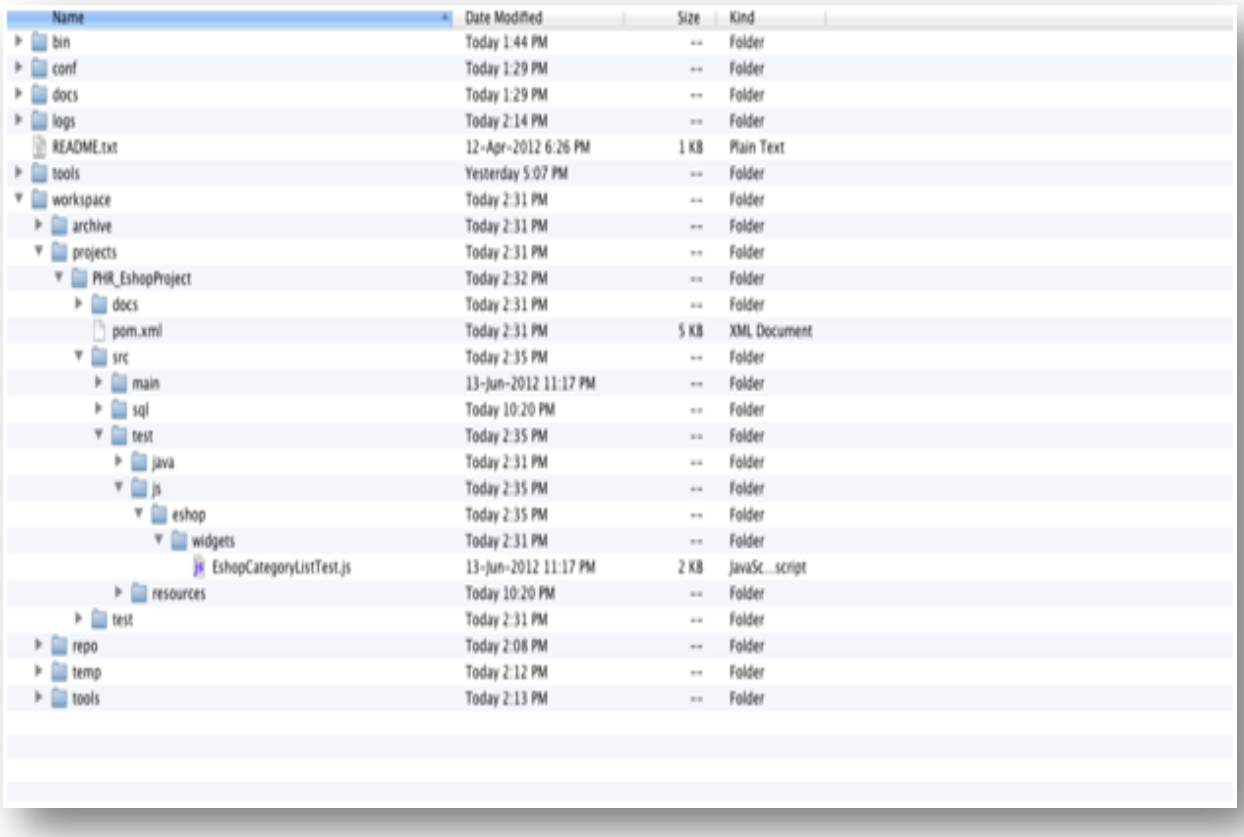

Figure 8-71: jQuery unit test case structure

a. **Test cases:** Test cases can be added directly in the js folder.

```
/*global require */

require([ "jQuery", "./Category", "./EShopAPI", "qunit" ], function($,
Category, EShopAPI, QUnit) {

    var equal = QUnit.equal, expect = QUnit.expect, test = QUnit.test;

    /**
     * Test that the setMainContent method sets the text in the category-
widget
     */
    test("category-widget unite test case passed.", function() {

        var category, initeObj, listener, api, Phresco, mainContent,
currentName, currentID, navUL, output1, output2;

        // Setup view and call method under test
        category = new Category();
        api = new EShopAPI();

  //api.wsURL = "http://172.16.25.75:2020/eshop";

        api.getWsConfig();

  category.api = api;
        category.listener = undefined;
        category.Phrescoapi = undefined;

        output1 = category.renderUI();

        mainContent = $('<section id="submenu">');
  currentName = 'name';
  currentID = 'id';
  navUL = $('<ul></ul>');

        api.getCategories(function(jsonObject){

            var productList = jsonObject,
            totalCategories = productList.category.length,
      i, category, categoryId, lis;
```

```
                        for (i = 0; i < totalCategories; i++) {
                                category = productList.category[i];
                    categoryId = category.id;
                    lis = $('<li><span><a href="javascript:void(0);">' +
category[currentName]
 + '</a></span></li>');
                                navUL.append(lis);
                        }
                    });

        output2 = mainContent.append(navUL);

                    // Expect that the text was set on the expected element
                    equal(output1.html(), output2.html(),      "Expected text not
            set in                  category-widget");
            });
});
```

8.10.1.2 Report generated after execution

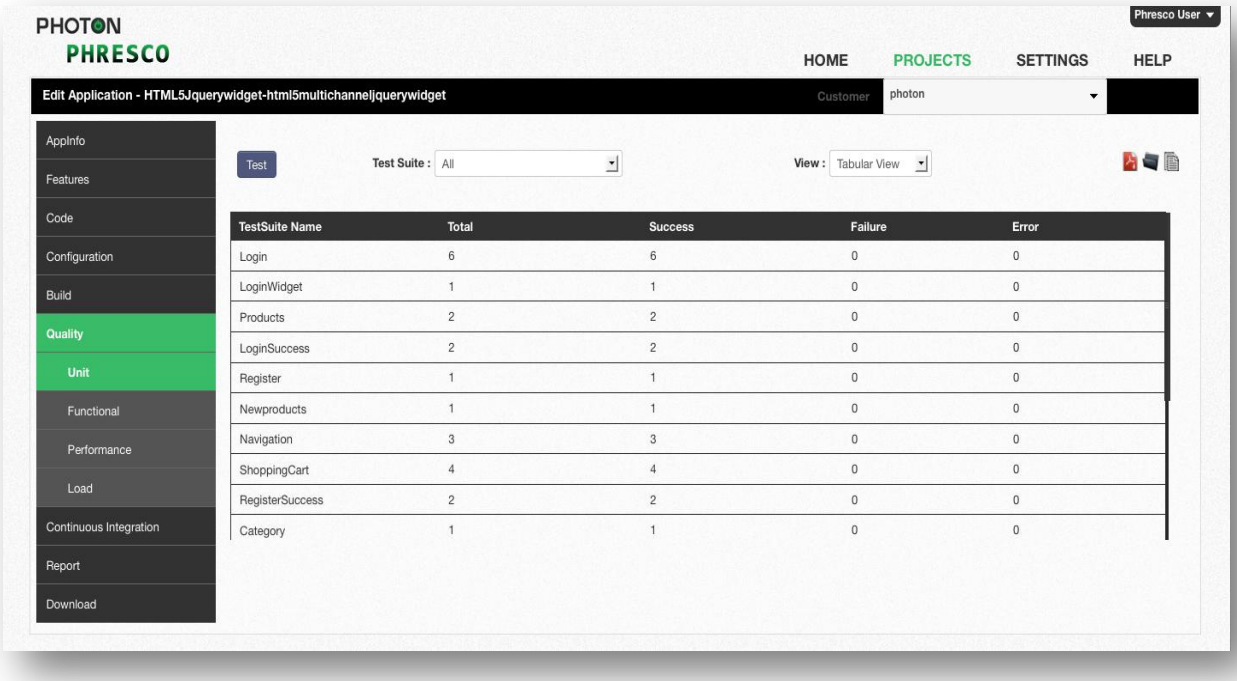

Figure 8-26: jQuery unit test report for all test cases in tabular view

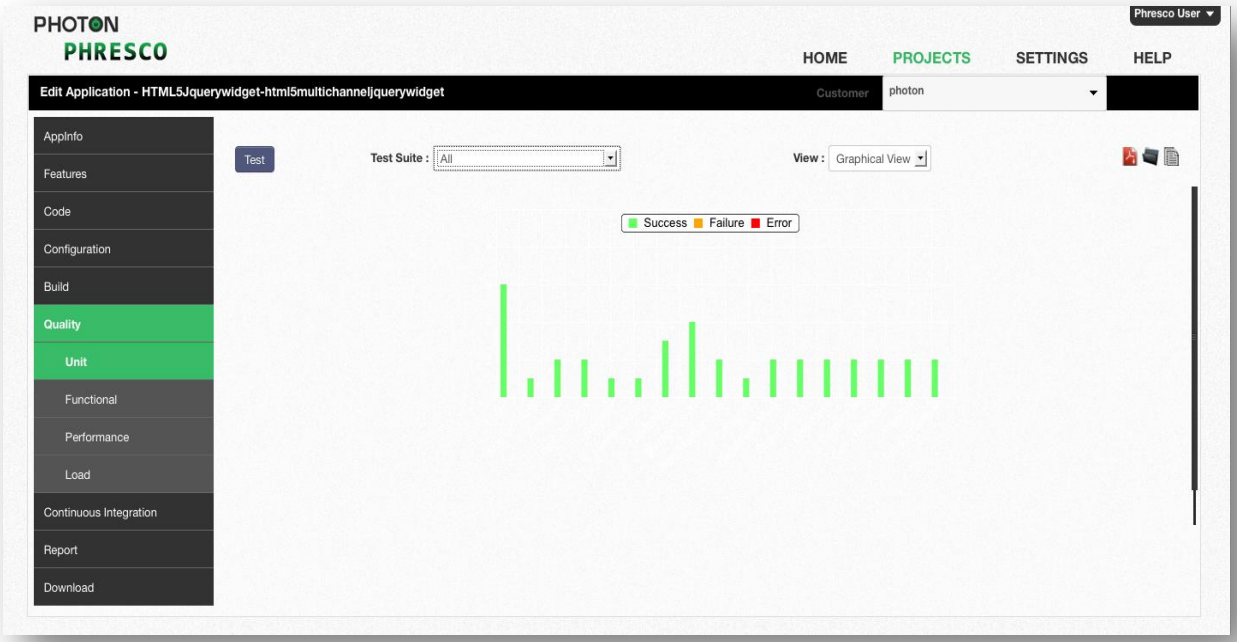

Figure 8-27: jQuery unit test report for all test cases in Graphical view

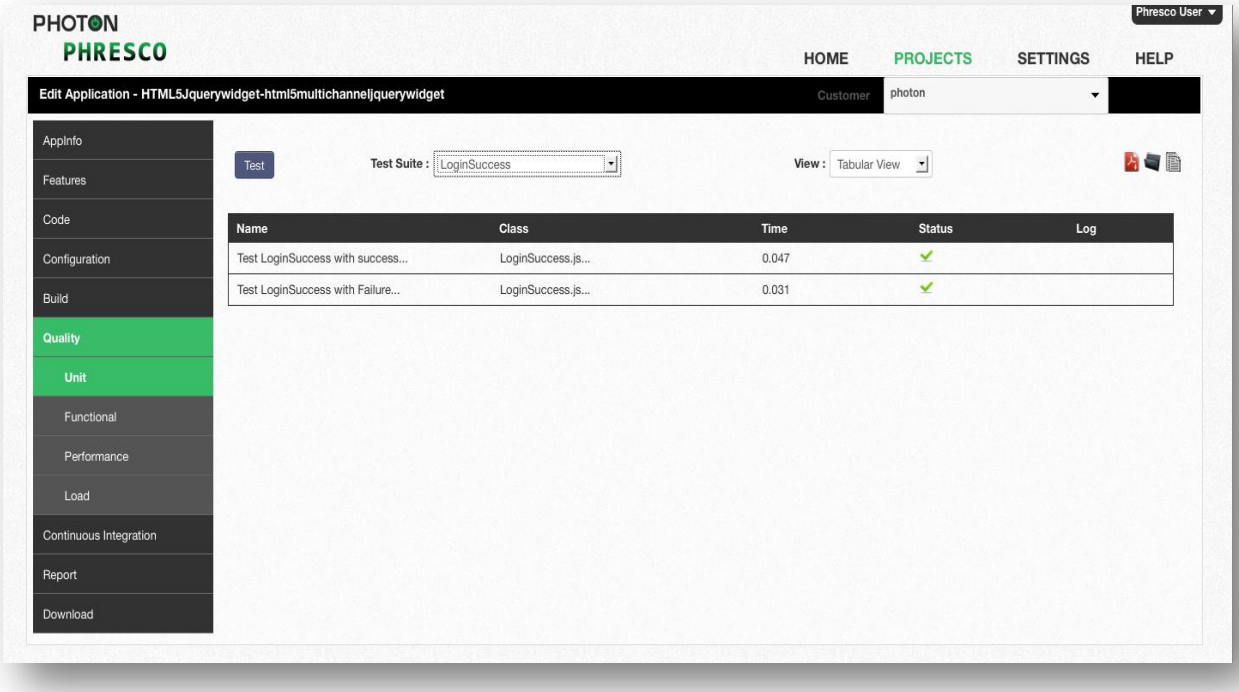

Figure 8-28: ***jQuery unit test report for single test case in tabular view***

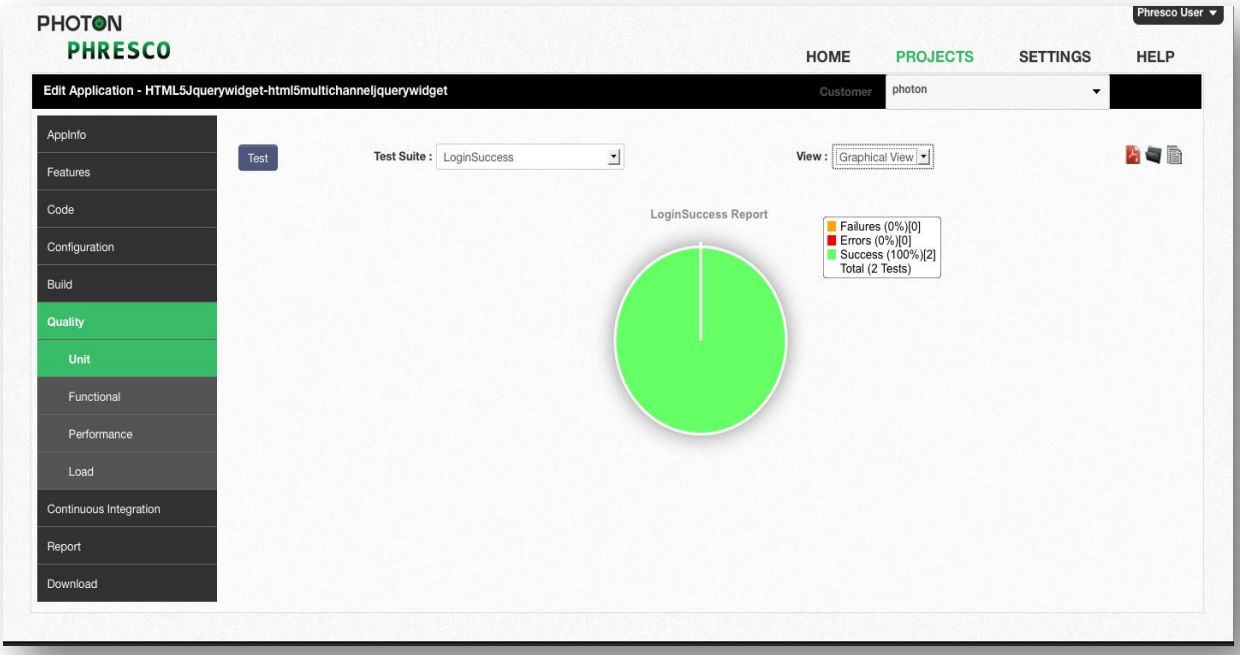

Figure 8-29: ***jQuery unit test report for single test case in graphical view***

Prerequisites for jQuery Unit Test Cases

A prior installation of Phantomjs software is essential to run Test cases for jQuery project. This software is available in the **HeliOS framework->Downloads**.

- Set environment variable for the phantomjs software.
- Extract the zip file and set the path {installation path}\phantomjs-1.5.0-win32-static in the system variables field.

8.10.2 Functional Test Cases

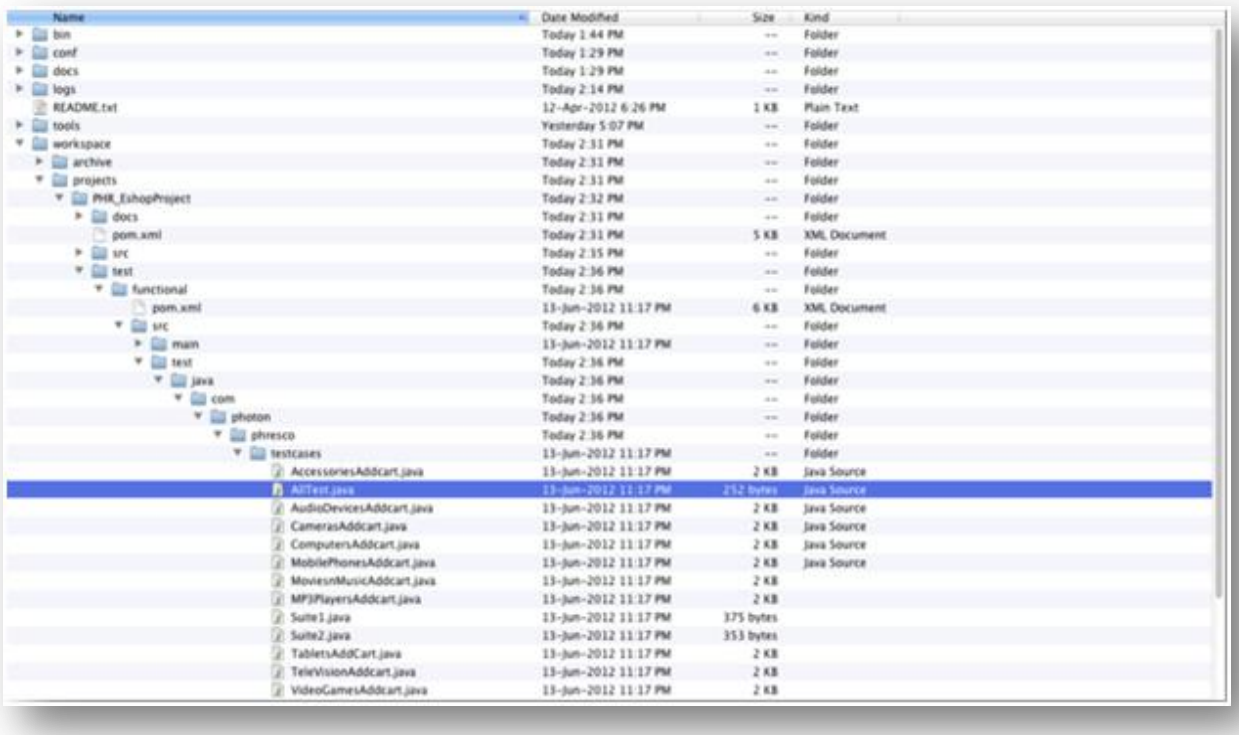

Figure 8-76: jQuery functional test case structure

a. AllTest: This is a root JUnit suite class that can either call a suite of classes or individual test cases.

b. Suite Class: This is also a JUnit suite class which calls the individual test cases.

c. Test Cases: These are individual java classes which perform unique testing scenarios against the application.

8.10.2.1 Existing Test Cases and Suites Out Of the Box in HeliOS

In HeliOS testing Framework in Junit suite class is named as “AllTest”.

The testSuite contains a bunch of test cases, each of which performs a unique scenario on the application. Test developers can start writing new suite classes and test cases by following the syntax and structure of HeliOS frameworks out of the box class structure. AllTest class calls all the suite classes and the test cases within it. Once suite classes and test cases are written developers can execute those from HeliOS Framework and can see the report. Following is the AllTest file which shows up how to add / include the class inside it.

```
Package com.photon.Phresco.testcases

import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)
@SuiteClasses({Suite1.class,Suite2.class})
public class AllTest {
}
```

Suite class

Suite class is a collection of test cases that are intended to be used to test a project code to show that it has some specified set of behaviors. A suite class contains detailed instructions or goals for each collection of test cases. New test cases can also be added by using the same syntax.

```
package com.photon.Phresco.testcases;
import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;
@RunWith(Suite.class)
@SuiteClasses({ WelcomePage.class,TeleVisionAddcart.class,
                ComputersAddcart.class,
MobilePhonesAddcart.class,AudioDevicesAddcart.class,
CamerasAddcart.class
})
public class Suite1 {
}
```

Test cases

A test case is a set of conditions under which a tester will determine whether an application is meeting the requirement. This helps the user to verify a particular functionality during the testing process. Elements can be identified and screen shots can be captured when the test fails.

```
package com.photon.Phresco.testcases;

import java.io.IOException;

import junit.framework.TestCase;

import org.junit.Test;
//import static org.testng.Assertjunit.*;
import org.openqa.selenium.server.SeleniumServer;

import com.photon.Phresco.Screens.MenuScreen;
import com.photon.Phresco.Screens.WelcomeScreen;
import com.photon.Phresco.selenium.report.Reporter;
import com.thoughtworks.selenium.Selenium;
import com.photon.Phresco.uiconstants.PhrescoUiConstants;

public class AWelcomePage extends TestCase {

 private SeleniumServer serv;
 protected Selenium selenium;
 private PhrescoUiConstants phrsc;
 private int SELENIUM_PORT;
 private String browserAppends;

 @Test
 public void testWel() throws InterruptedException, IOException, Exception
{
        try {
                phrsc = new PhrescoUiConstants();
                String serverURL = phrsc.PROTOCOL + "://"
                                + phrsc.HOST + ":"
                                + phrsc.PORT + "/";
                browserAppends = "*" + phrsc.BROWSER;
                assertNotNull("Browser name should not be
null",browserAppends);
```

```
            SELENIUM_PORT = Integer.parseInt(phrsc.SERVER_PORT);
            assertNotNull("selenium-port number should not be null",
                        SELENIUM_PORT);
            WelcomeScreen wel=new WelcomeScreen(phrsc.SERVER_HOST,
      SELENIUM_PORT,
                  browserAppends, serverURL, phrsc.SPEED,
                  phrsc.CONTEXT );
                  assertNotNull(wel);
                  MenuScreen menu = wel.menuScreen();
            assertNotNull(menu);

            } catch (Exception t) {
                  t.printStackTrace();
                  System.out.println("ScreenCaptured");
                  selenium.captureEntirePageScreenshot("\\WelPageFails.png",
                              "background=#CCFFDD");
            }
      }
      @Override
      public void setUp() throws Exception {

            serv = new SeleniumServer();
            try {
                  serv.start();
            } catch (Exception e) {
                  clean();
                  throw e;
            }
      }
      @Override
      public void tearDown() {
            clean();
      }
      private void clean() {
            if (serv != null) {
                  serv.stop();
            }
            if (selenium != null) {
                  selenium.stop();
            }
      }
}
```

8.10.2.2 To Add a New Test Suite in Alltest Class

You can add a new test suite to the AllTest class as follows.

Example

```
package com.photon.Phresco.testcases;

import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)
@SuiteClasses({Suite1.class,Suite2.class})
public class AllTest {
}
```

8.10.2.3 To Add New Test Case in Test Suite

You can add a new test case to the Test suite using the following method. Here is an example for creating a new test case in the name “CamerasAddcart”

```
package com.photon.Phresco.testcases;

import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)
@SuiteClasses({
WelcomePage.class,TeleVisionAddcart.class,ComputersAddcart.class,
                    MobilePhonesAddcart.class,AudioDevicesAddcart.class,
                    CamerasAddcart.class
                  })
public class Suite1 {

}
```

8.10.2.4 Report generated after execution

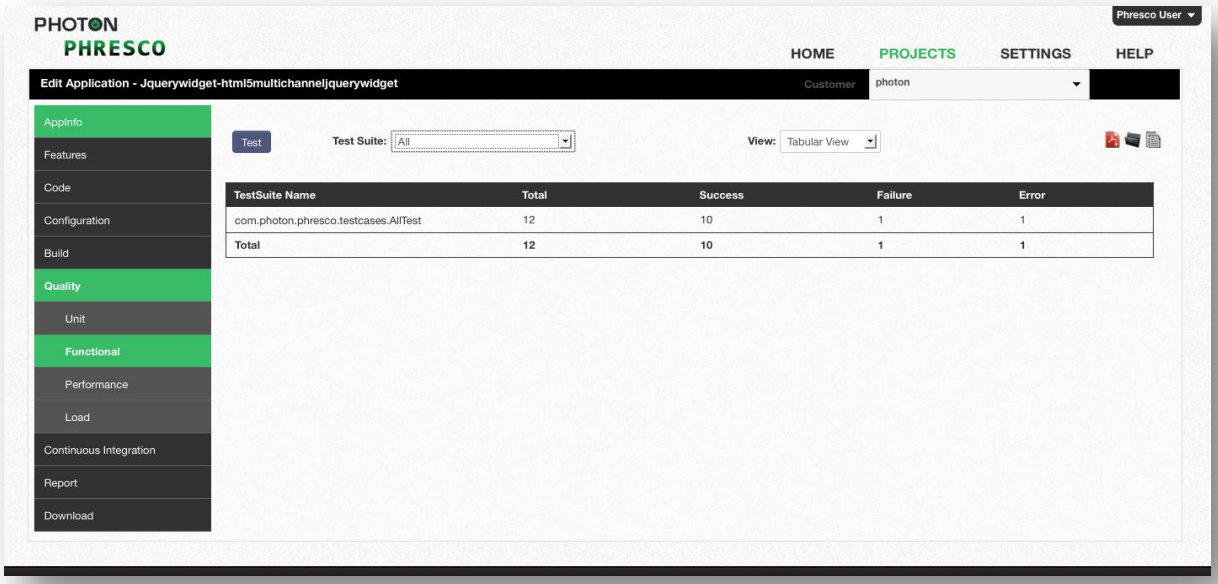

Figure 8-30: jQuery Functional test report for all test cases in tabular view

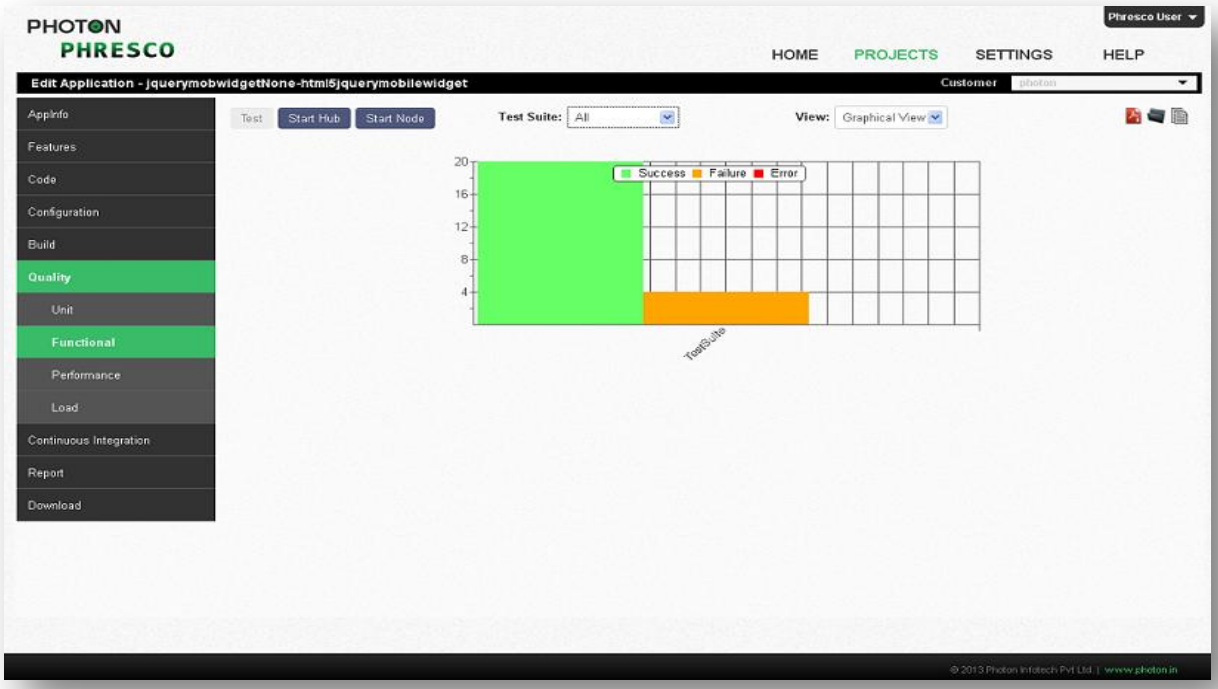

Figure 8-31: jQuery functional test report for all test cases in graphical view

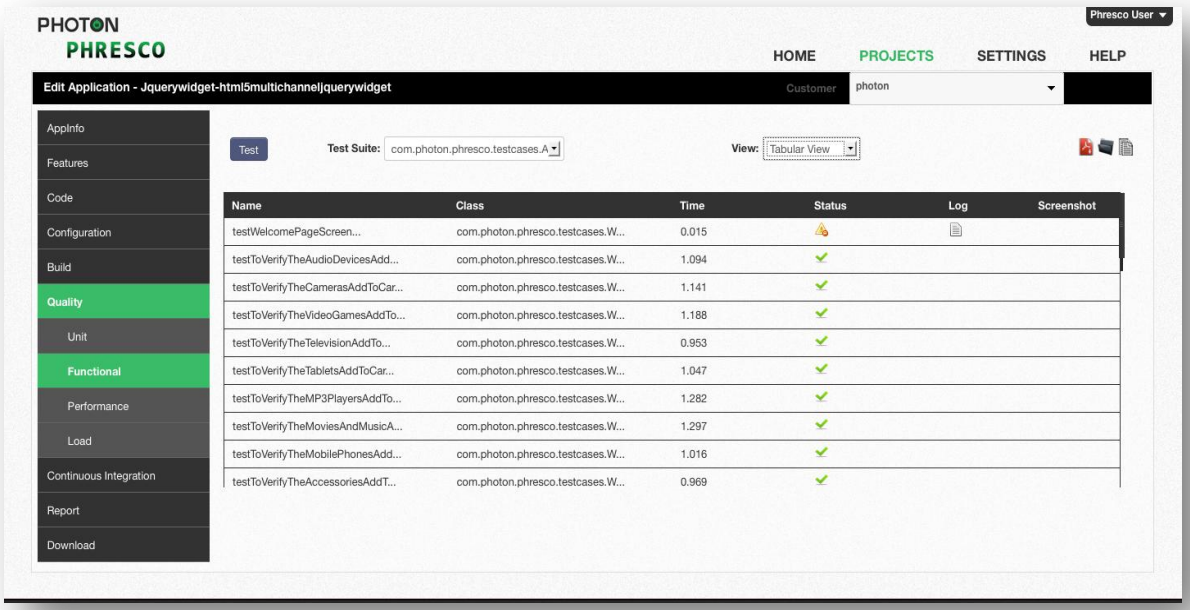

Figure 8-32: jQuery functional test case report for single test case in tabular view

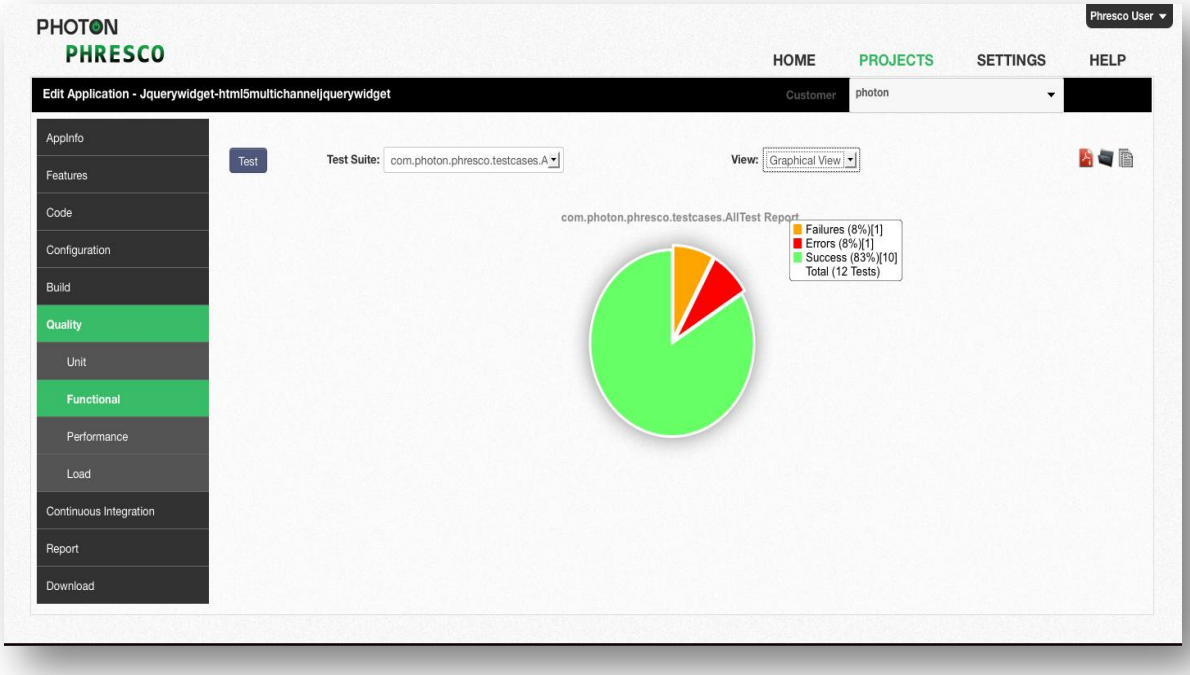

Figure 8-33: jQuery functional test case report for single test case in graphical view

8.11 Performance Testing

HeliOS enables the users to test the performance of their project in an elementary way. It integrates JMeter as a tool and as a result generates a report that can be viewed by HeliOS users. The Apache JMeter is a tool which is mainly used for testing both performance and load testing. It is fully comprised of java application and provides the result in tabular and graphical form. The inputs are entered in HeliOS user interface which in turn is assigned as an input to the JMeter. The parameters for performance testing is calculated and given as a final report in HeliOS framework.

Performance testing is testing the performance of a Server when certain number of users hits the URL at specific period of time. This helps in calculating the average response time, throughput, minimum responsive time and maximum responsive time.

Average response time:

Average response time is the Average time calculated when the server responds for any given input. This is calculated by using JMeter.

Throughput:

Throughput is the amount of capacity that a server can handle. In other words throughput is the amount of work that a server does in a specific time.

Minimum and Maximum Response time:

Minimum Response time is the minimum time calculated when the server responds for any given input.

Maximum Response time is the maximum time calculated when the server responds for any given input.

Performance testing can be done against server, web service and database.

Performance test against server by using HeliOS Framework:

To run a performance test against a server, the server has to be selected along with the environment and the Test Result Name should be filled. To test the performance of any server, access to the server should be available. Add header tab is used to enter the header parameter which authenticates the server for testing. Few mandatory fields in Context URLs like Name, context, Type, Encoding and fields

like No. of Users, Ramp-Up Period, and Loop count in Thread Group should be filled before Performance testing is done. GET or POST type can be selected in the type field. GET is the method used when user needs to get the content from the server. POST is the method used when input is fetched. One or more number of servers can be tested by clicking the add button while the minus button is used to deselect the URLs.

Performance Test against Web Service using HeliOS Framework:

To run a performance test against a Web service, the Web services has to be selected along with the environment and the Test Result Name should be filled. To test the performance of any WebService, access to the WebService should be available. Add header tab is used to enter the header parameter which authenticates the WebService for testing. Few mandatory fields in Context URLs like Name, context, Type, Encoding and fields like No. of Users, Ramp-Up Period, and Loop count in Thread Group should be filled before Performance testing is done. GET or POST type can be selected in the type field. GET is the method used when user needs to get the content from the server. POST is the method used when input is fetched. One or more number of web services/servers can be tested by clicking the add button while the minus button is used to deselect the URLs.

Performance Test against Database using HeliOS Framework:

To run a performance test against a database, the database has to be selected along with the environment and the Test Result Name should be filled. To test the performance of any database, access to the database should be available. Add header tab is used to enter the header parameter which authenticates the database for testing. Few mandatory fields in Database Query like Name, Query Type, Query and fields like No. of Users, Ramp-Up Period, and Loop count in Thread Group should be filled before Performance testing is done. One or more number of database can be tested by clicking the add button while the minus button is used to deselect the database.

Report generated after performance testing

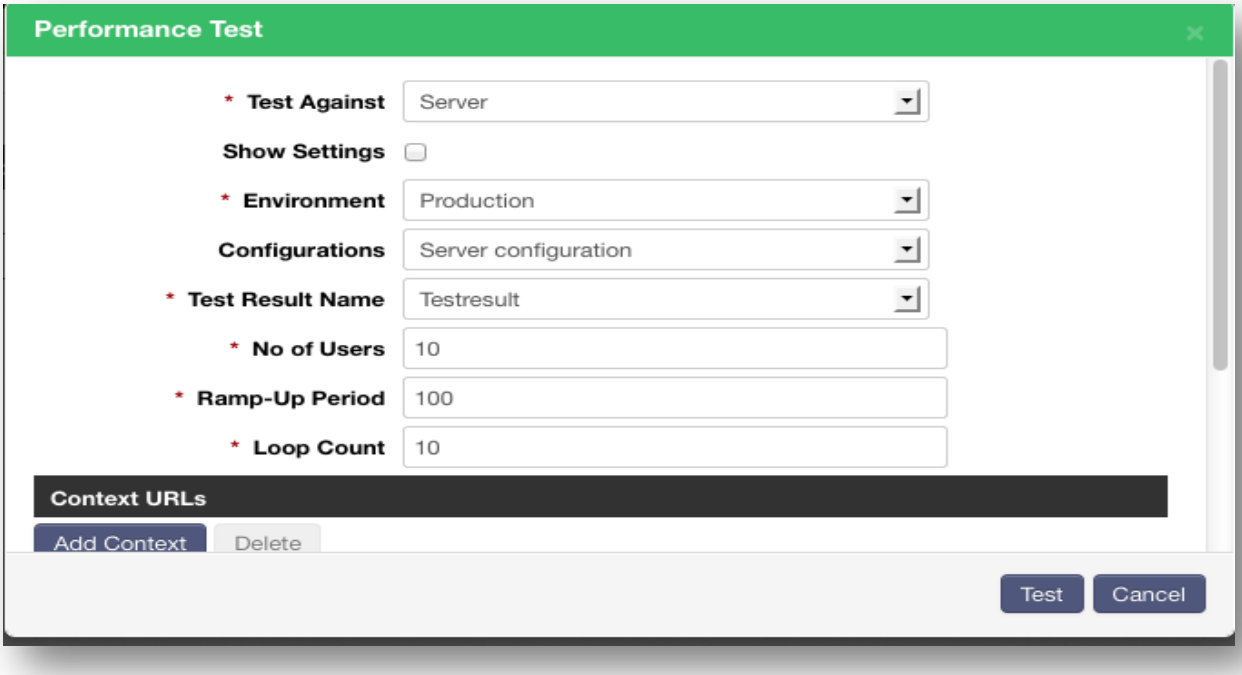

Figure 8-81: Performance test Pop up

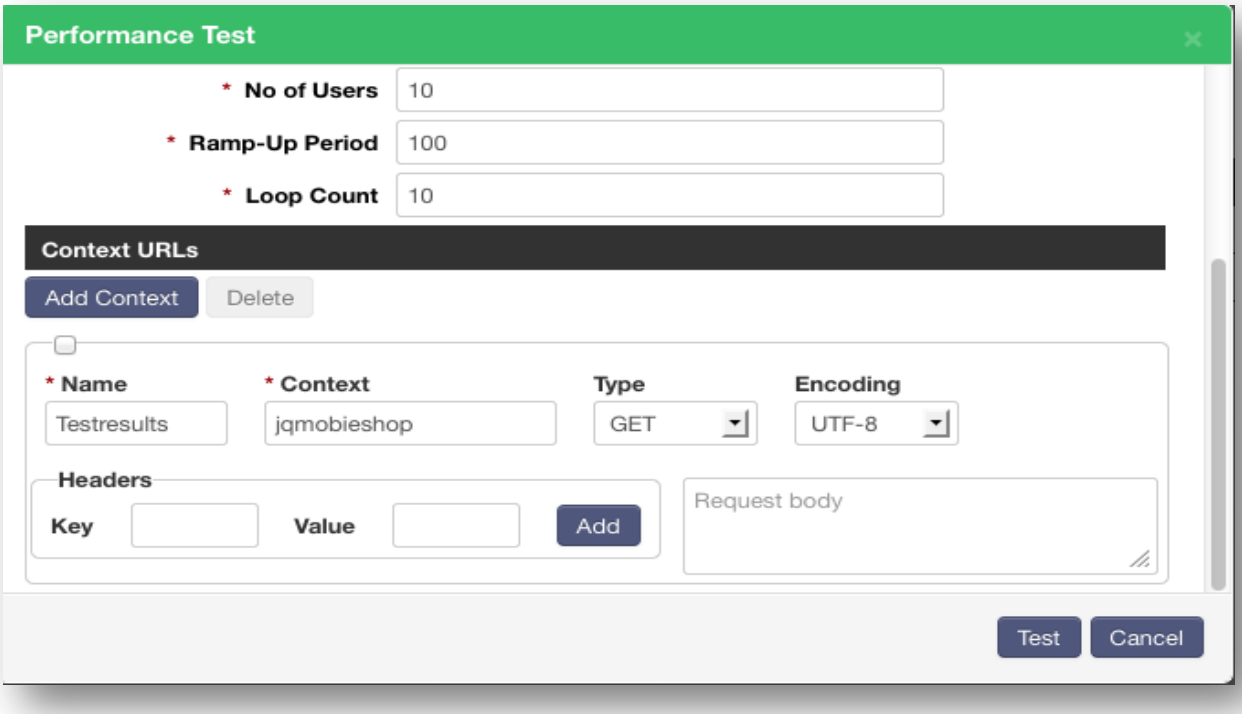

Figure 8-34: Request header

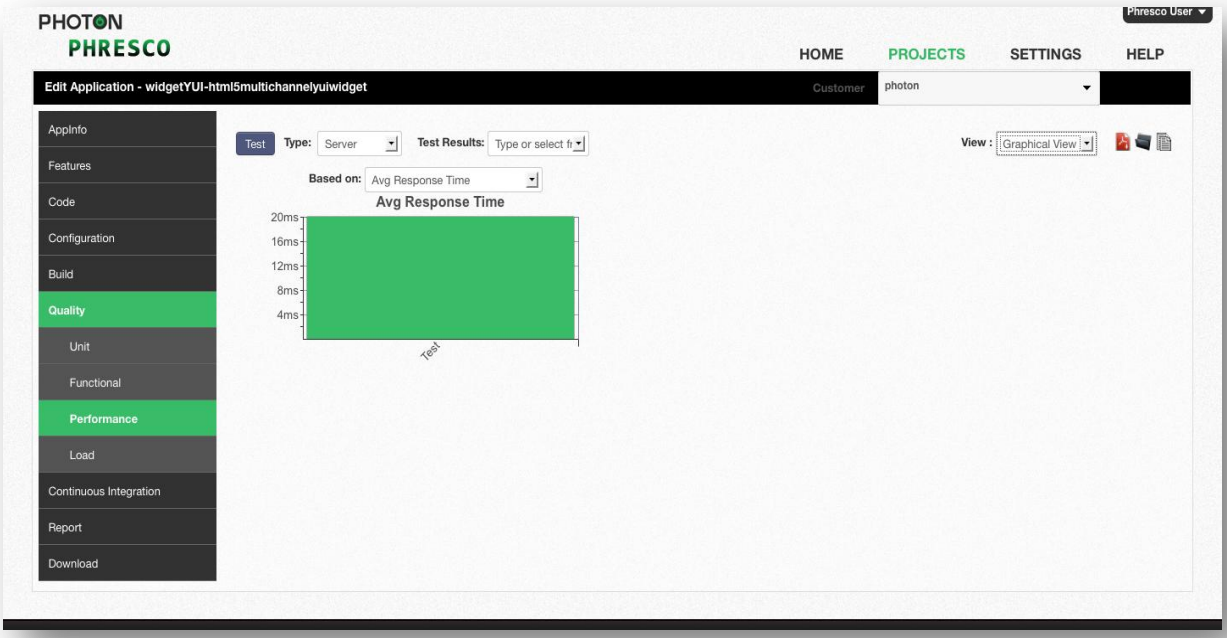

Figure 8-83: Performance test report graphical view

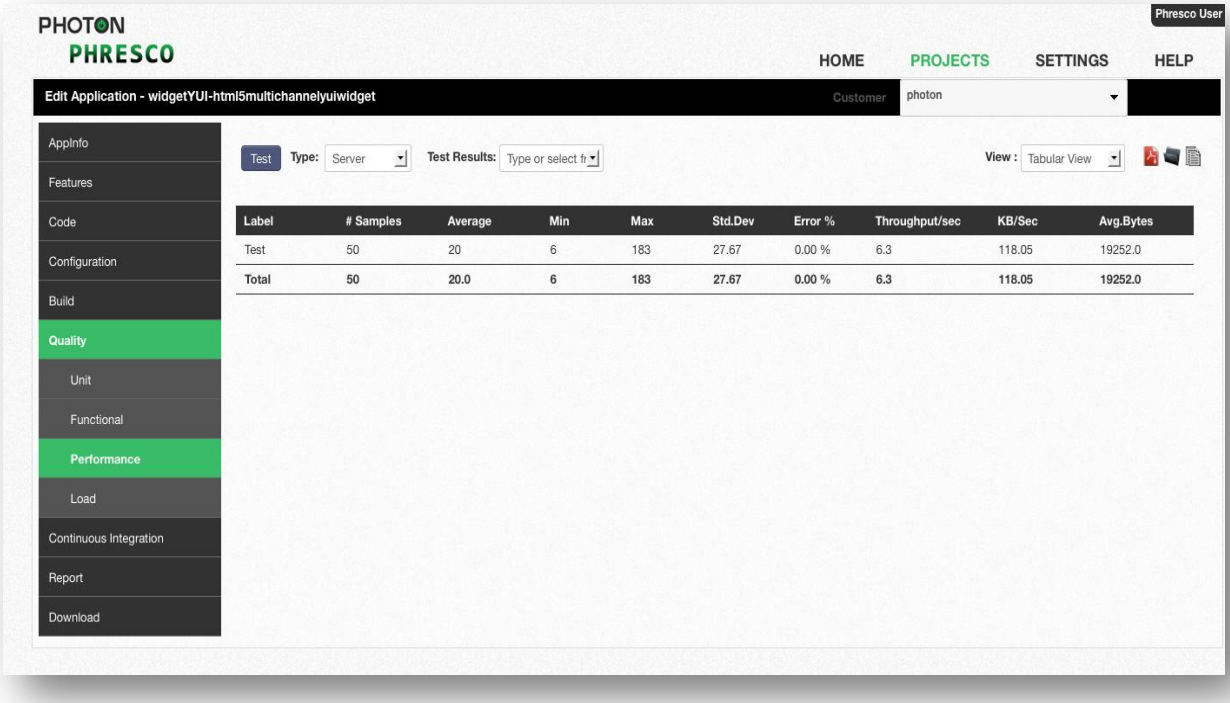

Figure 8-35: Performance test report tabular view

8.12 Load test

Load testing generally refers to the practice of modeling the expected usage of a server by simulating multiple users to access the same project concurrently. Project should be designed in such a way that when a maximum load is reached, a project should not crash and instead it should show a message saying the load has exceeded. Load and performance testing is usually conducted in a test environment identical to the production environment before a project is permitted for real time usage.

HeliOS uses JMeter for Load testing. The result is given through static analysis and test cases. The test cases will point out the error and this will be very useful for the testing team.

Elapsed Time

The amount of time that has passed since a particular process started especially compared with the amount of time that was calculated for it in a plan.

Load test against server using HeliOS framework

To run a load test against a server, the server has to be selected along with the environment and the Test Result Name should be filled. Few mandatory like No. of Users, Ramp-Up Period, and Loop count in Thread Group should be filled before load testing is done. By clicking the test button, JMeter carries the inputs and provides the end report in both tabular and graphical form. The elapsed time and the status can be viewed from HeliOS user interface.

Load test against Web service using HeliOS framework

To run a load test against a Web service, the Web service has to be selected along with the environment and the Test Result Name should be filled. Few mandatory like No. of Users, Ramp-Up Period, and Loop count in Thread Group should be filled before load testing is done. By clicking the test button, JMeter carries the inputs and provides the end report in both tabular and graphical form. The elapsed time and the status can be viewed from HeliOS user interface.

8.12.1 Report Generated After Load Testing

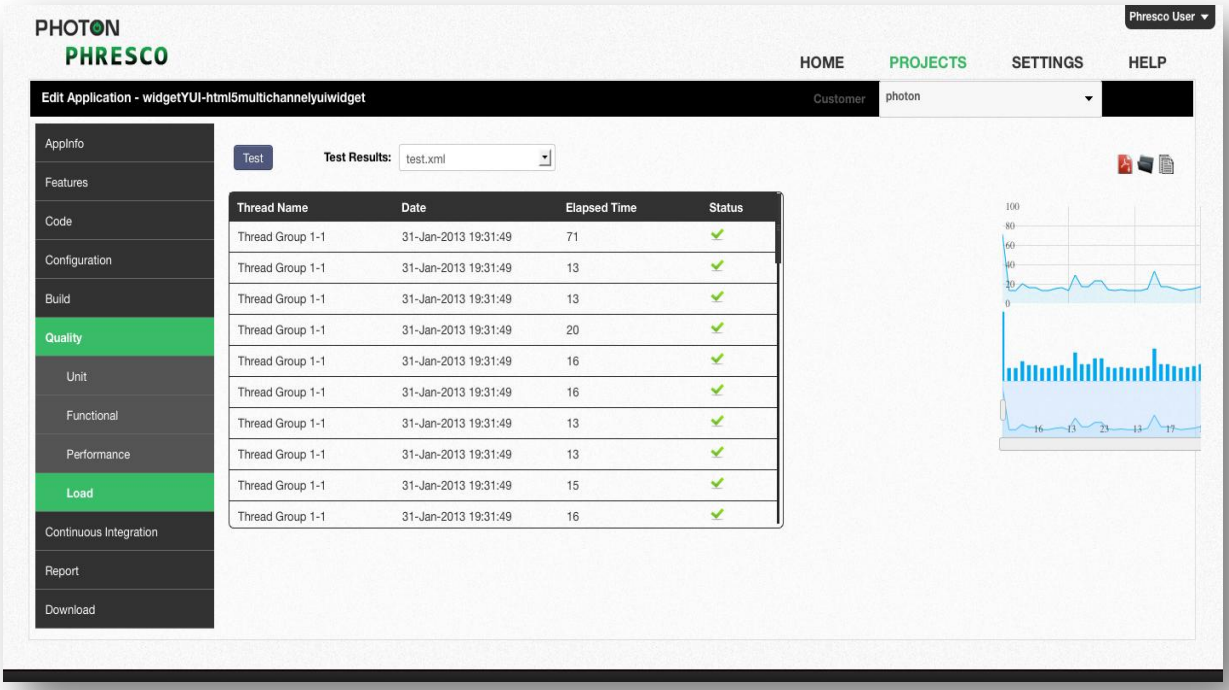

Figure 8-36: Load test report

9 Continuous Integration

Continuous Integration is the process where the builds are automatically generated and deployed by scheduling it to a particular time. Also testing process can be automated using continuous integration. This process reduces the integration problem and it gives the development team to produce a full quality project. This deducts the error that occurs during build, deployment and testing. It automatically gives the report through provided mail id. Continuous Integration uses the tool called Jenkins which is integrated within HeliOS Framework. Jenkins server starts separately in the local host and the port number is 3579 and the context is ci. When the codes are changed by the developers in the version controlling system, builds are automatically developed and the report is generated inside the Jenkins which also intimates about the build success or failure through email. If there is no change in the code, build will not start in the server.

HeliOS also allow users to create multiple jobs by scheduling different project in different timings. This multiple job in continuous integration can be achieved by creating jobs after the completion of one job.

9.1 Performance of Continuous Integration

Setup button present in HeliOS user interface loads the Jenkins and Jenkins can be started when the start button is clicked.

Note:

Once the Jenkins is started for one technology, it is not necessary to start the Jenkins again for other technology. When the stop button is clicked, the Jenkins stops running and the starting procedure should be initiated from setup and start.

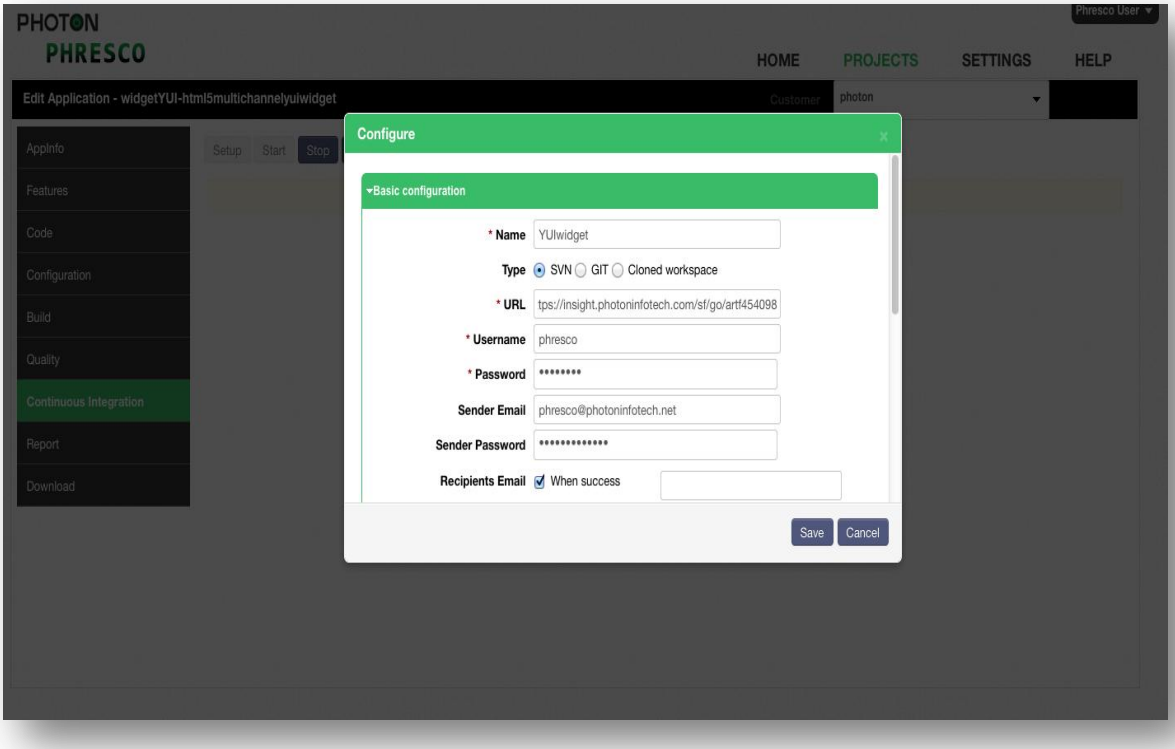

Figure 9-1: Continuous Integration

Fields in the Continuous Integration are as follows

Name

It denotes the name which identifies the Continuous Integration file and it is automatically filled with the code PHR_, followed by a project name.

URL

It denotes the SVN/GIT URL from which the projects are checked. Continuous Integration process is carried on to the copied link of a project.

Cloned Workspace

Select Clone Workspace in the configuration to re-use/clone the downloaded source code in another configuration's workspace.

Username

It denotes the SCM username.

Password

It denotes the SCM password.

Downstream Project

It creates link within the multiple jobs created so that when one operation of the job is completed, the next operation starts automatically depending on the order selected

Sender Email

It denotes the email id to which status of the build can be sent.

Sender Password

It denotes the password of the email id to which status of the build can be sent.

Recipients Email

The checkboxes in this field denotes, if the build failure or build success report should be sent through Email. When success checkbox sends the email when the build succeeds and when failure checkbox sends the email when the build fails.

Build triggers

- Build periodically:
  The option build periodically is to build the project in specific time.

- Poll scm:
  This option builds the project only when there is a commit in the source code.

Schedule

The date and time intervals can be selected which builds a project automatically in the given period of time and sends the report to the mail.

Cron expressions

A cron expression is a string consisting of few sub expressions that describe individual details of the schedule.

9.2 Continuous integration with Build configurations

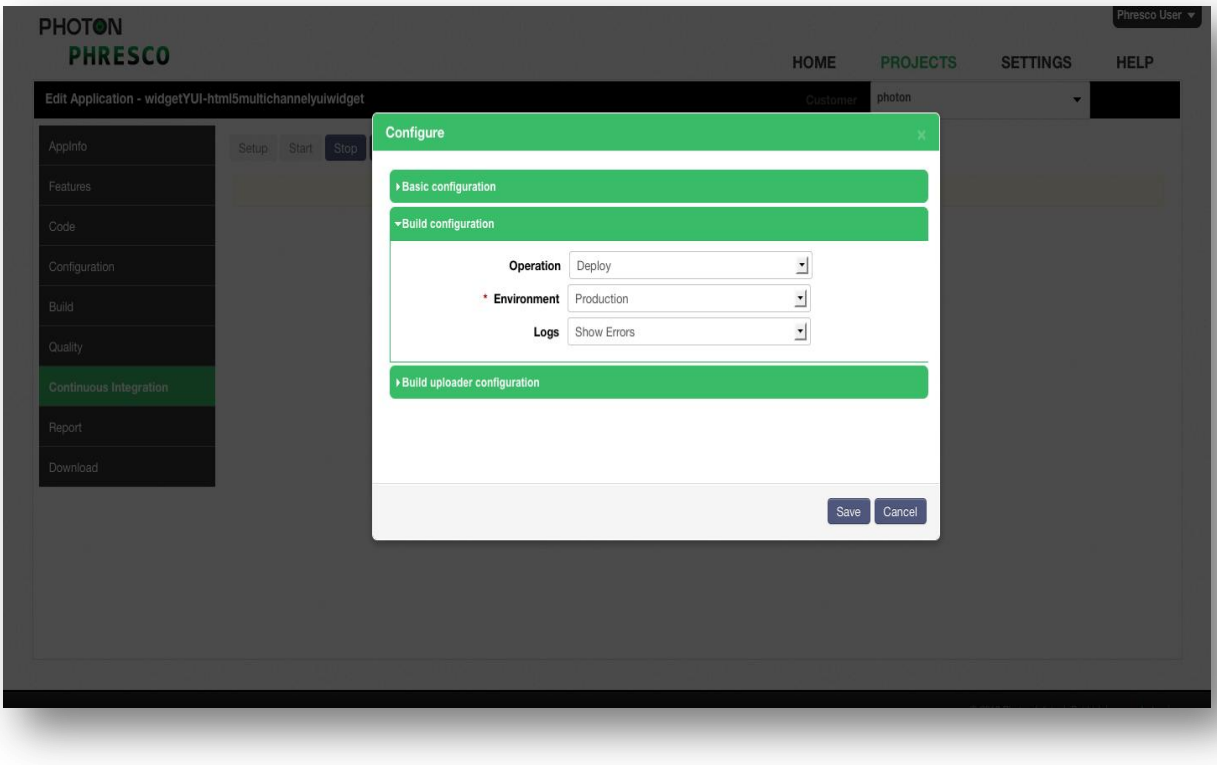

Figure 9-2: Build configuration

Fields in the Build Configuration are as follows

Operation

Build, deploy and test operation can be selected from the dropdown field to perform the particular operation

Environment

Created environments can be selected from the drop down box. If the environment is not created Production environment is selected as default

Clone the workspace

This can be enabled to use the same source code across all the operations and can be selected as 'no' to check out the source code for each and every operation

Show settings

The global configurations for server, database, web service and email can be selected using the show settings option.

Skip unit tests

HeliOS enables skip unit tests where the unit tests will be skipped on building a project. This consumes less time on building a project.

9.3 Continuous integration with collabnet plugin

Continuous integration process builds the project automatically and also uploads the build in insight using the collabnet plugin.

Fields in the Build Uploader Configuration are as follows

Enable collabnet file release

When the build is to be uploaded in insight automatically, collabnet file releases can be selected as yes otherwise no option should be selected.

URL

It denotes the collabnet WebService URL to which the build should be uploaded.

Username

It denotes the user name of the insight credentials for accessing the insight.

Password

It denotes the password of the insight credentials.

Project

It denotes project name in the collabnet to which the project is to be uploaded.

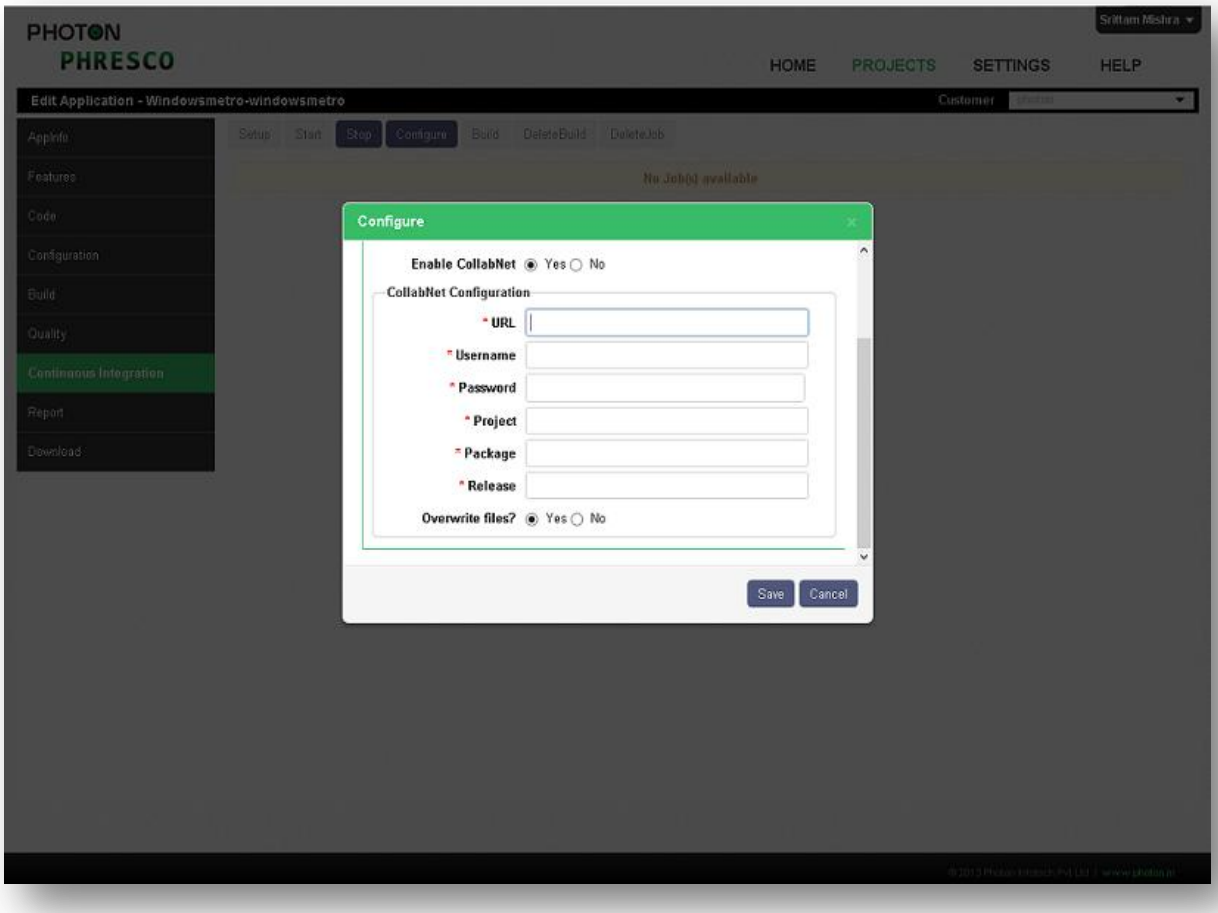

Figure 9-3: Build uploader configuration

Package

It denotes the Package name of a project where the build is to be uploaded.

Release

It denotes the folder releases name of a project where the build should is to be uploaded.

Overwrite existing files

When the existing build in insight is to be overwritten, yes option can be selected otherwise no option should be selected.