Photon Phresco

Phresco Framework

Newest version of the Social Media Framework

Version 1.1

May 2012

This document applies to Phresco Framework v1.1 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 who aim to use the Phresco Framework user interface for maintaining and controlling the critical phases in the entire life cycle of the project.

1.1 Document Conventions

Convention	Description
Blue	Identifies elements on a screen
Italic	Quotations

2 Introduction

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

2.1 What Is Phresco Framework?

Phresco is a next-generation development framework of frameworks. It is a platform for creating next generation web, mobile and multi channel presences leveraging existing investments combined with accepted industry best practices. Phresco 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.

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

3 Phresco – Powered With Facets

3.1 Lifecycle Management:

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

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

3.2 Pre - Built Architecture:

Phresco provides a pre built architecture model with certified template architectures like multi channel widget, SharePoint web parts, NodeJS Service gateway, etc… It contains standard build structures with supported versions.

3.3 Quality Assurance:

Phresco has a series of powerful automated testing instruments designed to analyze and test the work at the code level through various QA schemes.

The testing methods deployed in Phresco 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, Phresco makes sure that your project is of quality standards. Phresco 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:

Web development supports for ASP.NET, SharePoint, PHP (raw), PHP (Drupal), Word press, HTML Mobile Widget, Java standalone, and YUI & jQuery widget archetypes. It also supports web applications using Responsive web design and Java / NodeJS based web services. (Stacks)

3.5 Mobile Web/ Application Development:

Mobile web application can be developed using HTML Mobile, YUI and jQuery widgets as Responsive web design.

3.6 Multi Channel 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, Phresco has enabled Widgetized Responsive web design to be accessed across various channels.

3.7 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 such a wide variety of mobile devices, screen sizes, and hardware features, designing a separate version of a site for each quickly becomes impractical.

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

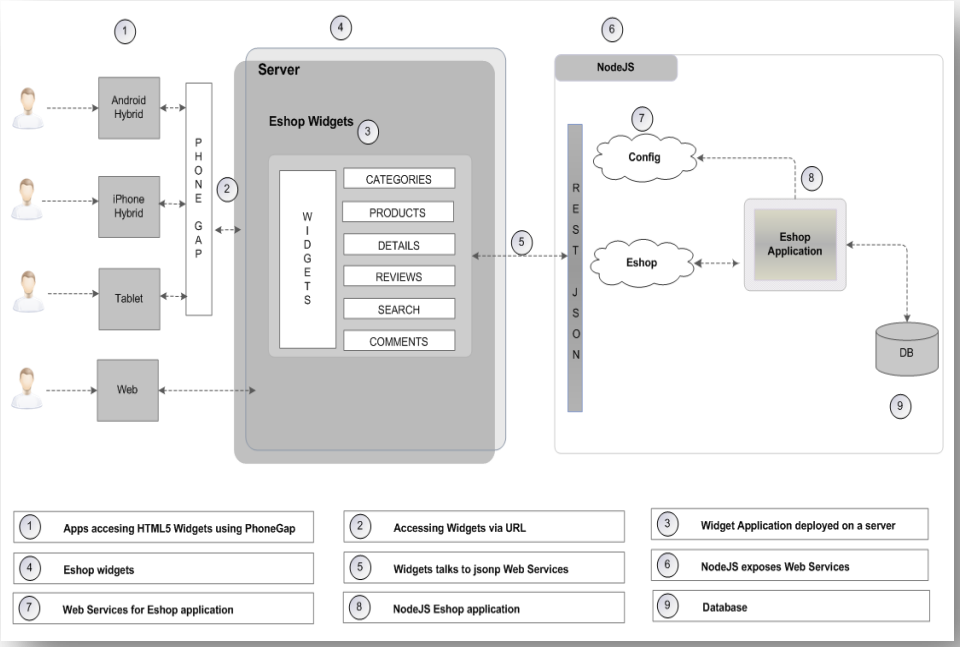


Figure 1: A representative view of RWD deployed in Phresco

3.8 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.

3.9 Continuous Integration:

One of the cherished aspects of Phresco is the continuous integration feature made available as part of the Framework which generates the build automatically 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.

4 Getting Started With Phresco 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 Phresco.

4.1 Supported Platform

- **Windows:** xp, 2003, 2008, windows 7. Both 32 and 64 bit supported.
- **Mac:** Mountain lion, Snow Leopard
- **Linux:** RedHat, SUSE Linux

4.2 Supported Browsers

- **FireFox:** Above Version 6
- **Google Chrome:** latest version
- **Internet Explorer:** Above Version 9
- **Safari:** Above Version 5
- **Opera:**

4.3 Steps To Extract Phresco Framework And Execute:

- Download the Phresco Framework-<version>.zip.
- Run Start framework server.bat for windows/ Run Start framework server.bat.sh for Mac and Linux
- Specify the URL: **http://localhost:2468/phresco** in any browser.

4.4 Logging On To Phresco Framework

- In the Web Browser, enter the URL where Phresco Framework is hosted.
- At the log on screen, enter valid Insight (Photon) credentials or guest credentials can be used as demouser as username and phresco as password
- Now click Login

4.5 Using the Navigation Controls in User Interface

Phresco's 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, Applications, and Settings or Help menus.

- Click on **Go** to Phresco Home in order to reach the Phresco HOME page.
- Click on Applications, Settings or Help in order to reach the respective pages.

Figure 2: Screenshot for Go to Phresco Home page

4.6 Phresco Videos

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

The Phresco video page is readily obtainable when go to Phresco home is clicked. The playlists mostly has the complex working experience of any project.

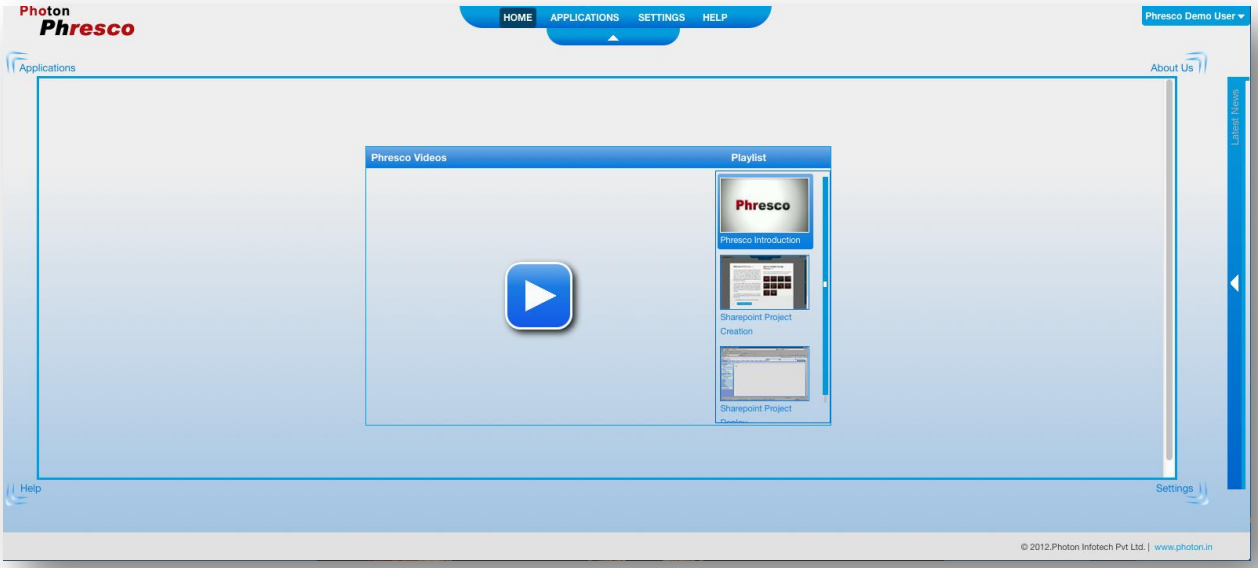


Figure 3: Screenshot for Home page

4.7 Logging Out of Phresco Framework

- To log out of Phresco Framework, click Sign Out (located in the drop down list on the right top corner of the page).
- If Phresco is left inactive for more than 3 hours, the session will automatically expire

4.8 To change the skin color of your account:

There are two different skins (Red and Blue) available in Phresco Framework. In Phresco User Interface, click skins (this link resides in the right side at the very top of the page)

4.9 To update the available version

Phresco releases periodic updates for the Framework and these new versions enables the "Update Available" button in the About Phresco tab. Users can update to latest available releases by clicking the "Update Available" button.

Screenshot should be inserted with changed content of about phresco

5 Phresco Project Life Cycle

Phresco is used to perform the following tasks effectively.

5.1 Project Creation

Phresco primarily helps the developers, testers, release engineers, managers and others to create projects with flexibility and affordability. Projects that are created using Phresco, invariably have high quality standards and best practiced structure. Once the project is created, Phresco also manages the entire lifecycle of the project and replenish the best productivity.

Creating an Application

Projects can be created with any type of technology and features(create hyperlink). Features and technologies are provided out of the box and can be used for creating the projects. Phresco is user friendly and helps the software engineers to create projects with more ease. The applications can be created from the Applications → Add Application. Applications tab has the application info and features from which the libraries can be selected for project creation in the wizard.

Application Info

This encompasses the information about an application of a project. Also this holds the information about the Server, Database, Web Service and Email components which the project consumes.

Fields present in the application info are as follows

Name

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

Description

It denotes the description of the project created. Upto 150 characters of the field can be entered and is a mandatory field.

Version

Version of the project can be entered in this field. 20 numeric characters with a special character "." can be entered in this field.

Applications

Application type can be selected from the radio button. There are three types of application administered by the Phresco, they are

- Web Application
- Mobile Application
- Web Services

Technology (Archetype)

Type of the technology should be selected in this field and it is a mandatory field. Mentioned below are the dependent technologies for each application type.

The technologies under web application are

PHP (Versions: 5.0.x - 5.4.x), Drupal6 (Versions: 6.3, 6.25, 6.19), Drupal7 (Version: 7.8), SharePoint (Versions: 3.5, 3.0, 2.0), HTML5 Multichannel YUI Widget (Versions: 1.6, 1.5), HTML5 Multichannel JQuery Widget (Versions: 1.6, 1.5), HTML Mobile Widget (Versions: 1.6, 1.5), ASP.NET (Versions: 3.5, 3.0, 2.0), WordPress (Version: 3.3.1) and Java stand alone (Versions: 1.6, 1.5)

The technologies under mobile application are

iPhone Native, iPhone Hybrid, Android Native and Android Hybrid.

The technologies under web services are

Node JS Web Service (Versions: 6.14, 6.11, 6.8, 6.7, 61) and Java Web service (Versions: 1.6, 1.5)

Pilot Project

Phresco has included Pilot projects for every technology it supports. 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.

The pilot project selected from the drop down list incorporates some libraries, components and validated code structures to the newly created project and henceforth can be modified based on the requirement. If none project is selected from the drop down list, all the components and libraries should be configured.

Phresco by publishing pilot projects in the repository server,

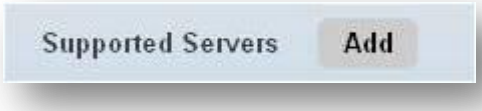
- Allows components/libraries deployed to be re-used in multiple other projects.
- Authorizes re-branding of the pilot projects.

Advantages

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

Supported Servers

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



The desired server can be selected using the "Add" option against Supported Servers in the "Add Application" window. For the same project, Phresco allows more than one server to be shortlisted. This has been implemented to answer the adaptability concerns that afflict projects. In the implementation page, the developers can choose their preferred server from the list.

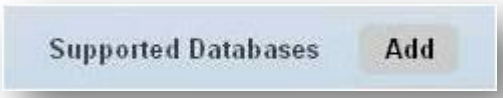
The supported servers in the Phresco framework include

- Apache Tomcat
- JBoss
- IIS
- Web logic
- Apache
- NodeJS
- Jetty.

These servers are available with different versions.

Supported Databases

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



The desired database can be selected using the "Add" option against Supported Databases in the "Add Application" window. For the same project, Phresco allows more than one database to be shortlisted. In the implementation page, the developers can choose their preferred database from the list.

The supported databases in the Phresco framework include

- MySQL
- Oracle
- MongoDB
- DB2
- MSSQl

These databases are available with different versions.

Web Services

Some of the established Web Services are supported in the Phresco Framework. As like Servers and the databases, Phresco allows more than one Web Service to be selected from the "App Info" window. From the shortlisted Web Services displayed in the implementation page, the desired option can be chosen.

The supported Web Services in the Phresco Framework include,

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

Features

Phresco also renders exiguous features to enhance a project. This include External features, JS libraries and custom features.

External features:

External features are those features provided out of the box by Phresco for the users to enhance their project. These features can be chosen in multiple based on the requirement of the project.

JS Libraries:

JS Libraries are the features provided out of the box by Phresco for the users in implementing the code. These libraries can also be chosen in multiple based on the project's requirement.

Custom Features:

These features are the customer specific features which can be incorporated in Phresco on a request by the user based on the project's requirement.

Features considered in Phresco are

Libraries: jar, dll, etc

Modules: Drupal module, NodeJS module, etc

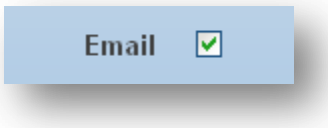
Features: Web parts.

Features are the libraries of a project and the developers make use of these features to have superlative creation of project. For example Facebook application is a feature which can be added to the project. This allows the user to access the Facebook application by contacting the web browser.

Email

Email checkbox can be selected if a project requires email configuration. There are some projects which uses the option of sending email notification to the mail id furnished by the user.

Only if the email checkbox is selected during the creation of project, email type will be displayed in the drop down shown in the configuration page.



When the application info and the features are selected the project creation is completed.

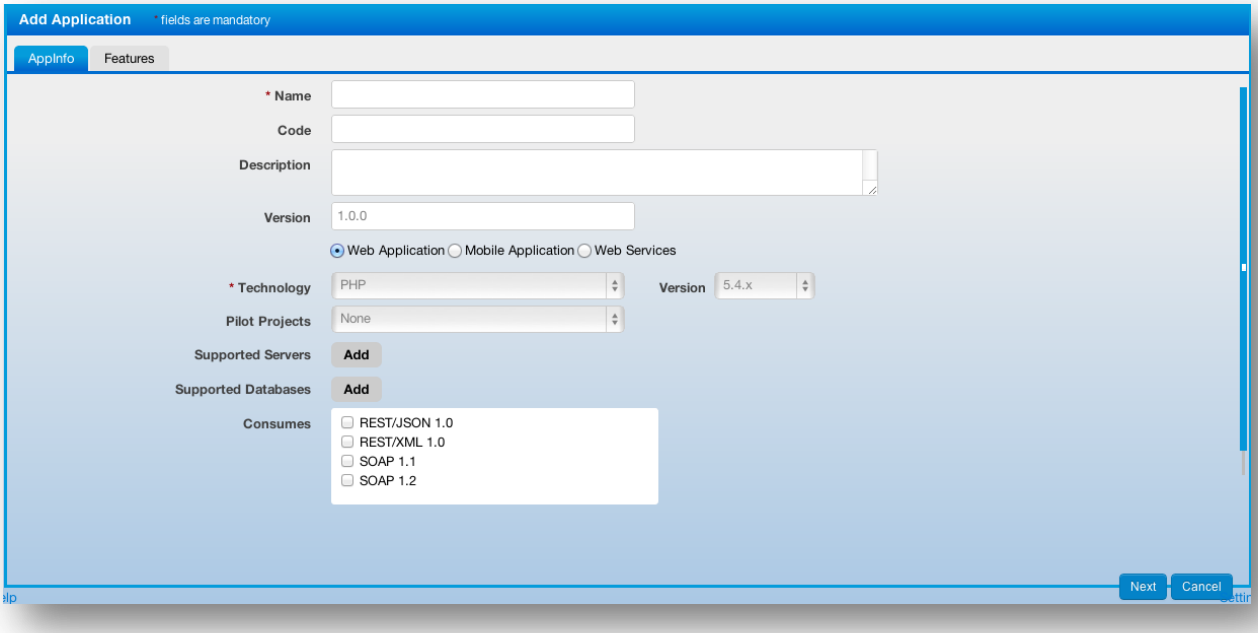


Figure 4: Screenshot for Application information page

5.2 Code Validation

Code validation is a method in which the source code of the project is tested in compliance with the standards. Phresco renders code validation by manipulating sonar tool which picks up standards for contra distinct technologies and examines in contrast with codes of a project. When Phresco starts, the code validation tool also starts automatically. Code validation process can be triggered by clicking on code validation only when the sonar tool is in active status.

How does Code Validation work in Phresco?

Code Validation is the tool which aggregates the standards to verify 100% validation of the code and code coverage for all the technologies. The errors are extricated depending on how the errors affect the project. Result is furnished in the form of graph.

Violations and Rules Compliance

- Blocker
- Critical
- Major
- Minor
- info

Code Validation

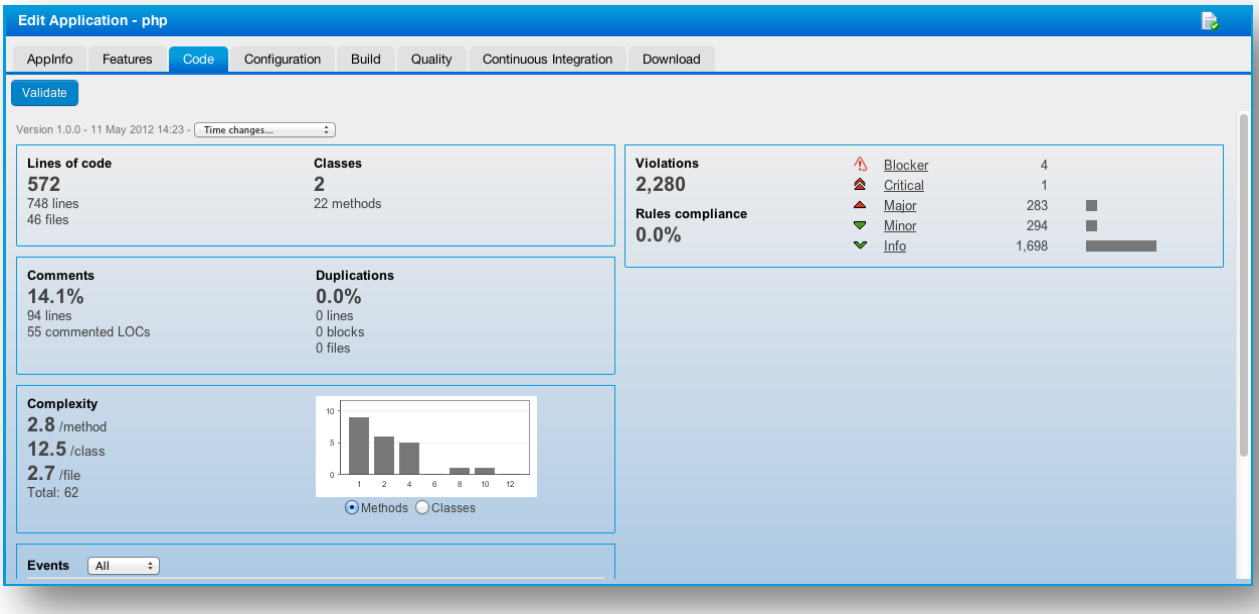


Figure 5: Screenshot for code validation

Prerequisites for code validation

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

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

How to install PEAR in Phresco?

1. Set PHP in environment variable
2. To set Maven path in the environment:
   - Open a command promt
   - Navigate to Phresco Framework> bin
   - Run the env.bat (windows) this will set Maven temporarily in the command prompt
   - 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.
3. Download <Php_Drupal_code_validation_setup> from Phresco

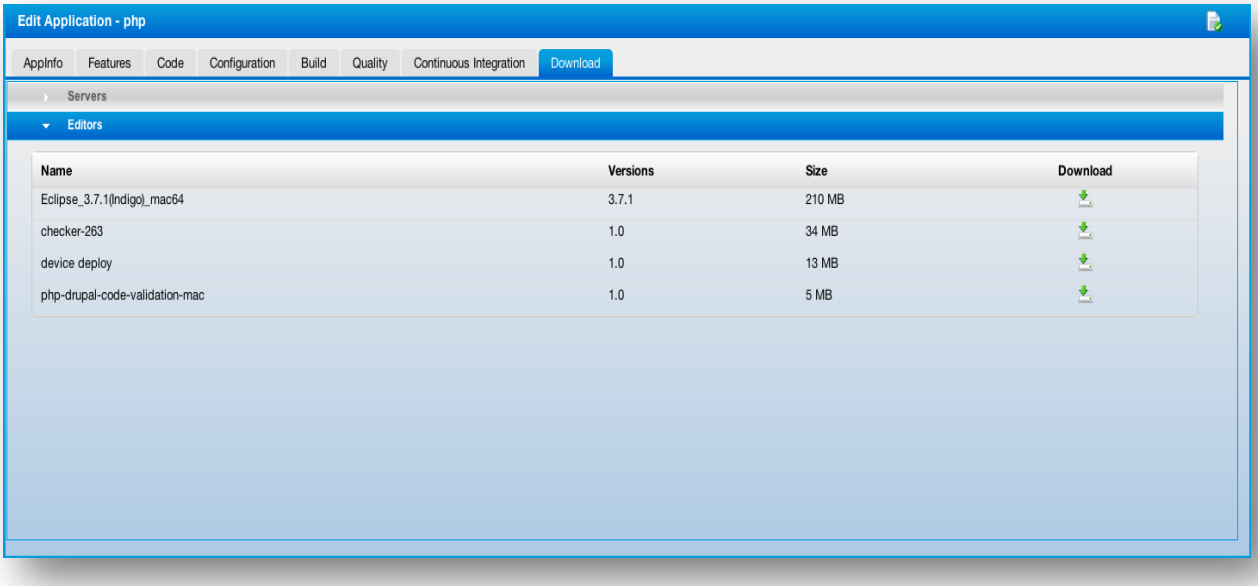


Figure 6: Screenshot for downloading <<Php_Drupal_code_validation_setup

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

Steps which requires user input

a. *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 button.

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

c. *Would you like to alter php.ini <D:\wamp\bin\php\php5.3.5\php.ini>? [Y/n] :*

Type y and press enter.

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

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

f. *D:\wamp\bin\php\php 5.3.5>call pear upgrade pear*
Wait till the system downloads the file.

g. After downloading system pops up with a message box.
Are you sure you want to add the information in C:\wamp\bin\php\php5.3.5\PEAR_ENV.reg to the registry?
Click yes

h. Click ok in the message box stating "Information in C:\wamp\bin\php\php5.3.5\PEAR_ENV.reg has been successfully entered into the registry"

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

5.3 Environment Configuration

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

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

Following steps can be followed to create an environment.

Step1: Multiple environments can be created from the Edit Application page by clicking the Application created→configuration→ environment

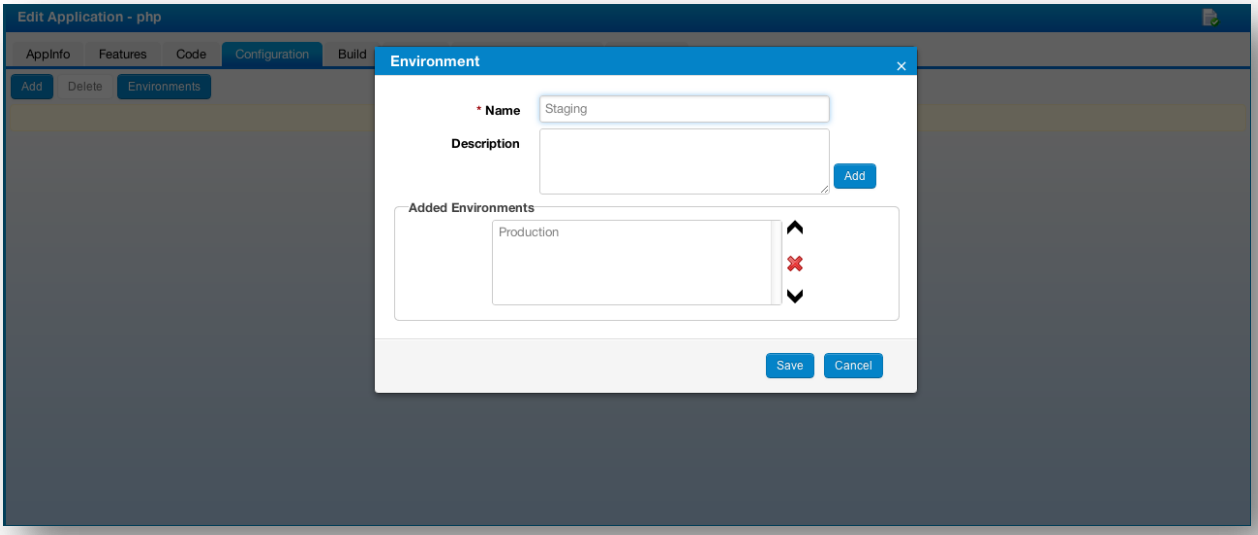


Figure 7: Screenshot for creating environment

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

Name

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

Added environment

The added environment lists the environments created. The deletion and ordering of environments can be done using the arrow and cross symbol present in this field.

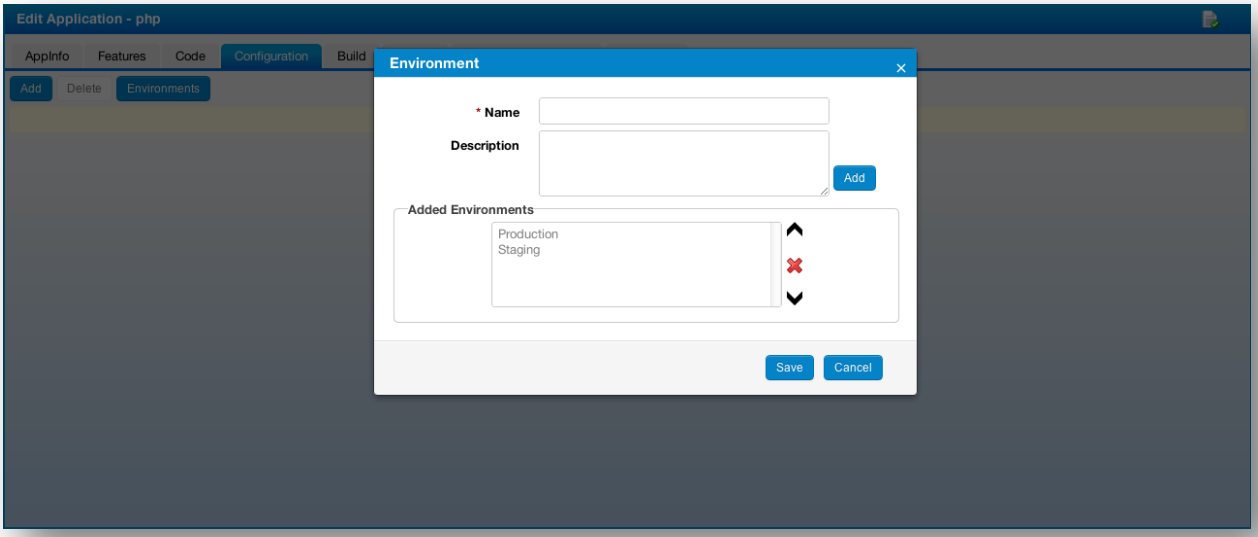


Figure 8: Screenshot for adding environment

Step3: Server, Database, Web Service and Email 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.

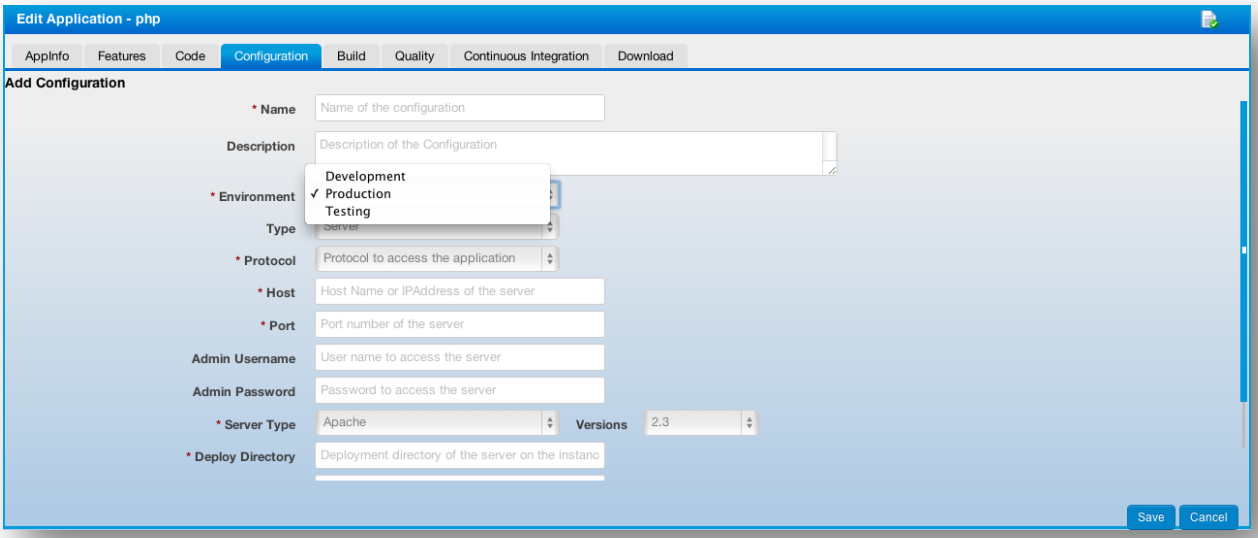


***Figure 9: Screenshot for* Selecting environment**

Note:

I. Multiple environments can be associated when building a project.
II. Projects can be deployed only in one environment
III. For functional, performance and load testing any one environment has to be selected.

Advantages:

Apart from these configurations, Phresco also allows user defined configuration template. New configuration templates can be published in the Phresco server which would be immediately seen in the drop down list of configuration page. For example log4j template can be published in the Phresco server which can be used across the projects.

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

There are four types of configuration which can be set for an environment.

5.3.1 Server Configuration

The server configuration can be added from Application created→configuration→ Add. The configuration type varies based on the configurations selected in the appinfo page.

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

Name

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

Server Type

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

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.

Root 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.

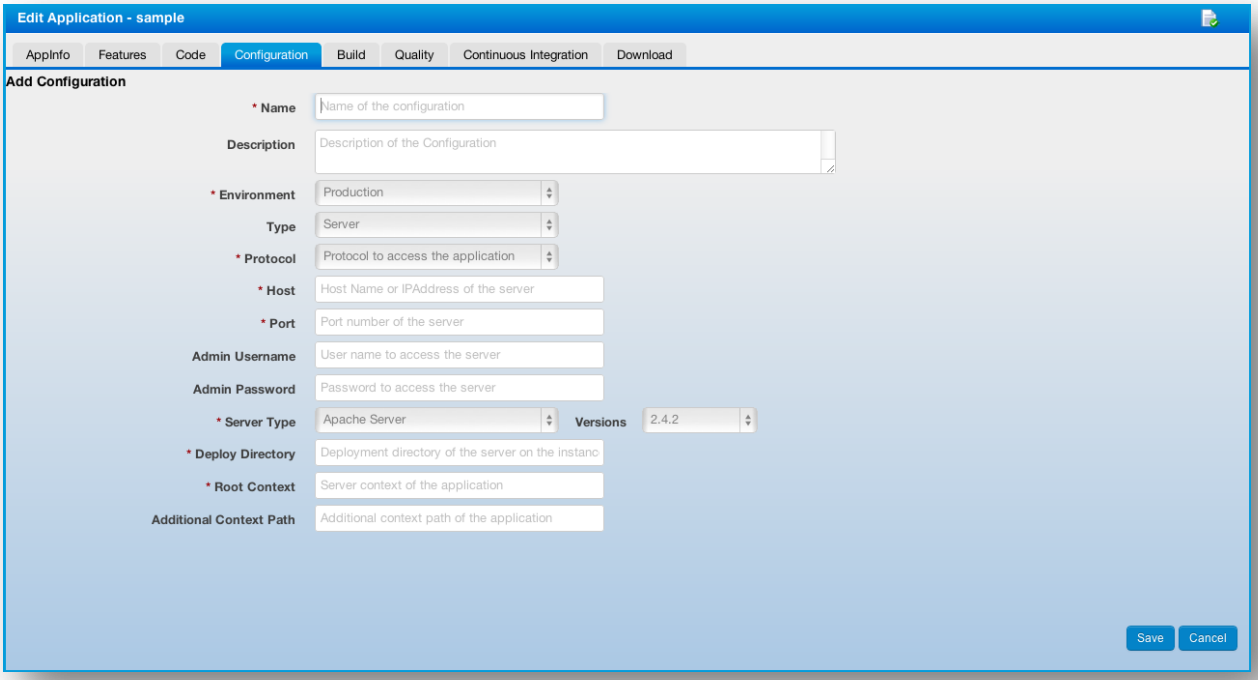


Figure 10: Screenshot for server configuration

5.3.2 Database Configuration

Name

It denotes the Name in which the 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 the 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 the 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

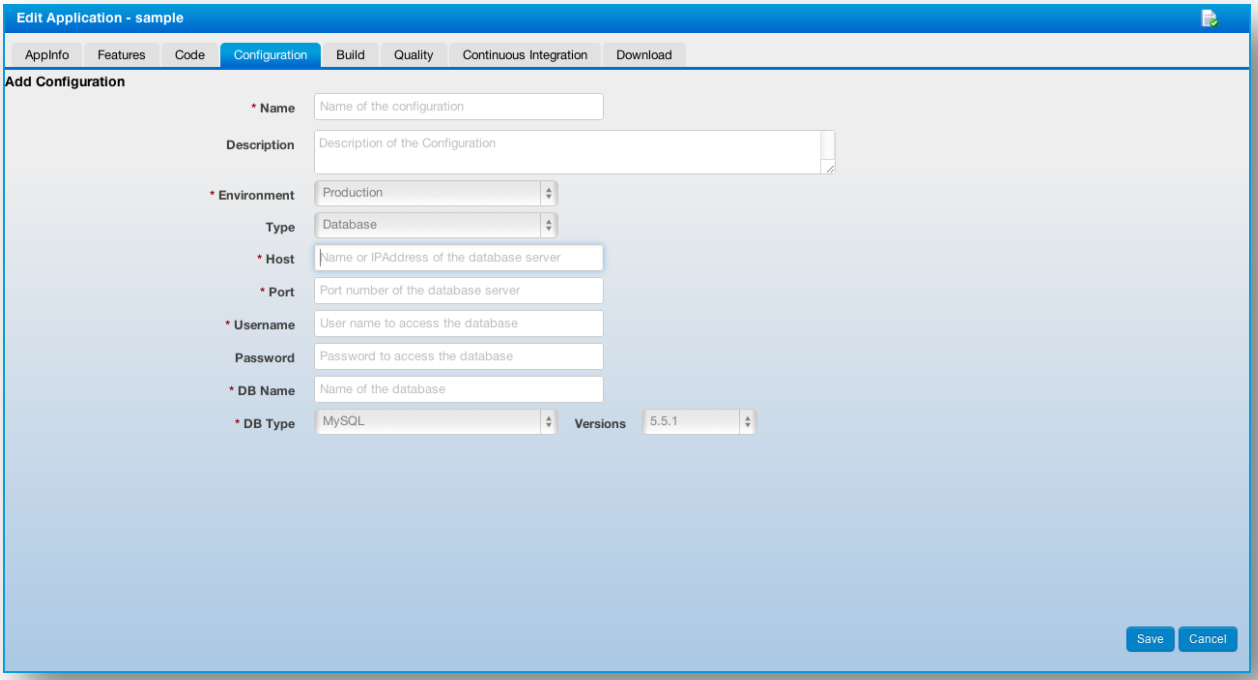


Figure 11: Screenshot for database configuration

5.3.3 WebService Configuration

Name

It denotes the Name in which the 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 the 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. Web Service option should be selected for Web Service configuration.

Host

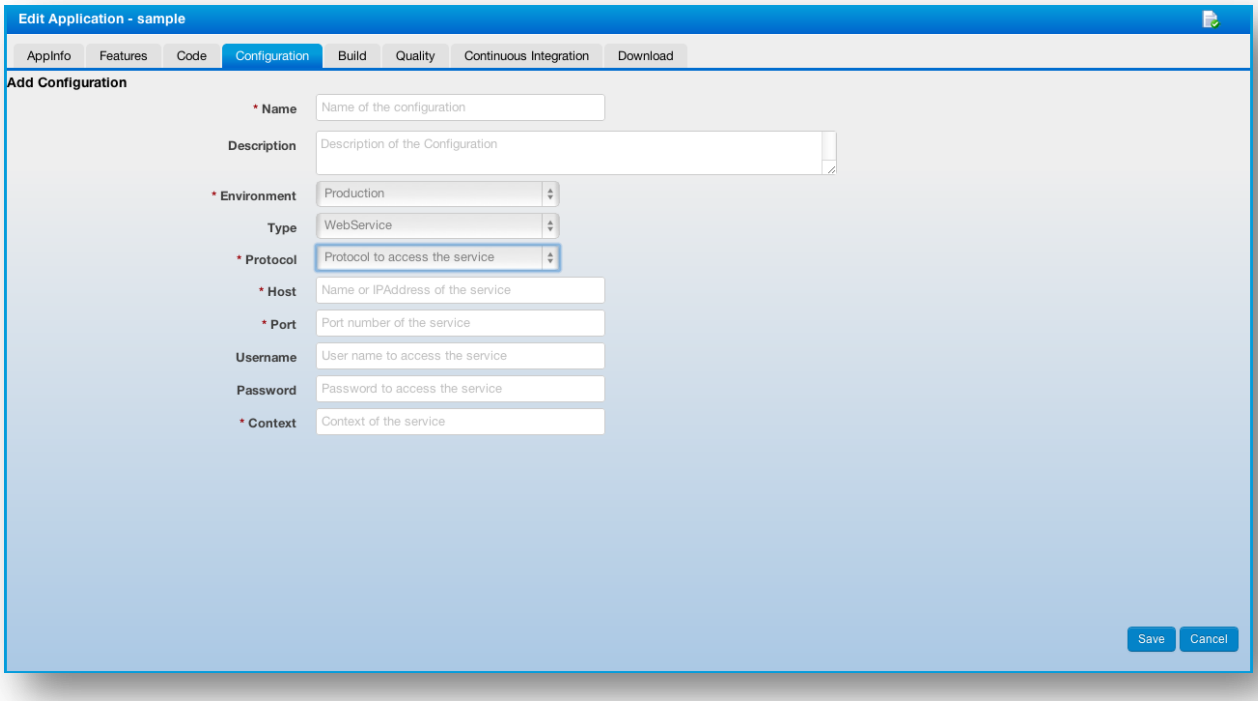
It denotes the Host name of the web service in which the project is created and is mandatory.

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.

Context

Context of the web service can be entered in this field. The symbol "*" notifies that this field is mandatory.



5·4

Figure 12: Screenshot for web service configuration

5.4.1 E-Mail Configuration

Name

It denotes the Name in which the 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 the 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.

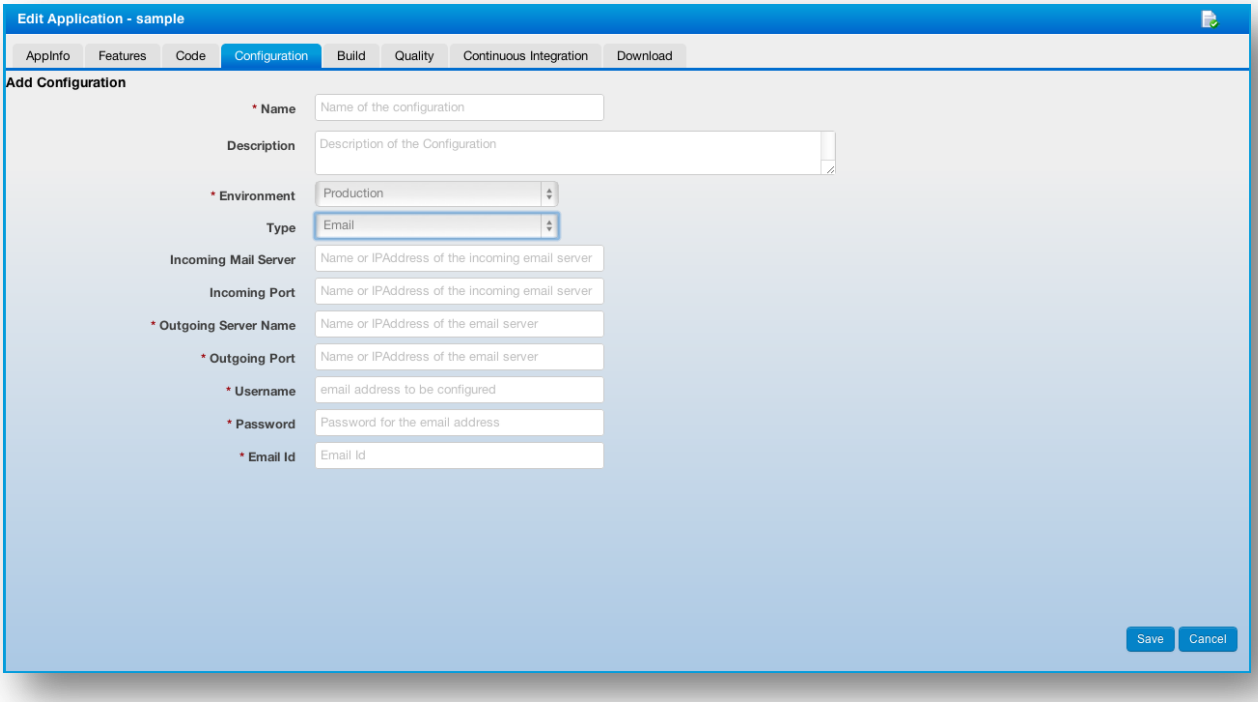


Figure 13: Screenshot for email configuration

5.5 Build generation

Phresco enables building a project with ease by single click. It builds ????.(source code). Build is to make the created project available for use. Build should be interpreted as a general process that should be customized according to requirement and characteristic of a particular project.

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

Environment Selection

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

Show settings

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

Hide Log:

5.6 Project deployment

Build and deploy is to make the created project available for use. Build and deploy should be interpreted as a general process that should be customized according to requirement and characteristic of a particular project.

By creating different environments projects can be built or deployed in various environments even in the remote system.

Deployment: Execute SQL

Execute SQL checkbox present in the deploy pop up pushes the database script in the database name created manually. The name of the database that is created manually and the DB name in the Phresco User Interface should be the same.

When the project is deployed the required database can be made available only by checking the option Execute SQL.

5.6.1 Remote Deployment

The application package can be deployed to a local macine or remote machine. Phresco enables the remote deployment concept when ever there is a requirement to deploy to a remote staging or production machine.

Enabling Remote Deployment:

Remote deployment in Phresco 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)

Servers supported in Phresco for Remote Deployment

- Apache Tomcat – Above 6.x
- JBoss (Restricted to 7.x)
- Weblogic (Restricted to 12c)

Note: Server should be up and running during remote deployment

5.7 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 Phresco framework circumvents the troubles associated with testing.

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

5.7.1 Unit Testing

Unit testing ensures that the developers writes proper project implementation.

For Unit testing, coverage and reporting Phresco framework uses many tools like JUnit, NUnit, WSUnit, PHPUnit, NodeUnit, 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.

5.7.2 Functional Testing

Functional testing involves testing on the specifications of the 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 and xcode. 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.

5.7.3 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 Phresco the result of the testing is given through static analysis and test cases.

5.7.4 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. Project should be designed such a way that when a maximum load is reached, the project should not crash. Instead a message saying the load has exceeded should appear. Load and performance testing is usually conducted in a test environment identical to the production environment before the 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.

5.8 Continuous Integration

Continuous integration process is used for generating the builds automatically. The builds that are generated manually using build option can be made automatic by scheduling the build time. The build automation process is done using Jenkins.

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

5.9 Knowledge Repository:

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

5.10 Applications:

Using Phresco, web, mobile and web services combination applications can be created based on the project requirement. Created project contains archetype (Project structure), features, testing structure and documents with in it. Features which are provided out of the box can be adapted to the project based on the need. If, the expected features not part of the standard fare can be availed on demand from photon after validation of the license.

Phresco conjointly allows the project leads to import any project into Phresco framework by using the import SVN. Apart from creating projects using Phresco, developers can also create their project with Phresco standards and import them into Phresco Framework. After importing the project, the project will be ready to

proceed with building, deploying, testing etc. Import SVN will automatically checks out the project in the desired path of the folder {phresco-framework-1.0\phresco-framework\workspace\projects}

Also projects to be imported in the Phresco framework can be checked out in the desired folder manually. This can be done by copying the project in the path {phresco-framework-1.0\phresco-framework\workspace\projects}

Import SVN

Version controlling system is possible through Phresco where the developers can view and make changes in the project created. Import from svn tab is used for importing a replica of the project and changes can be made in the project. Many developers can check in the project and make changes which in turn reflect in main project. Conflictions may occur if two developers checks in.

5.11 Download

Downloads in Phresco 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. Downloads are available for the ease of developers who can use it on any requirement for their project.

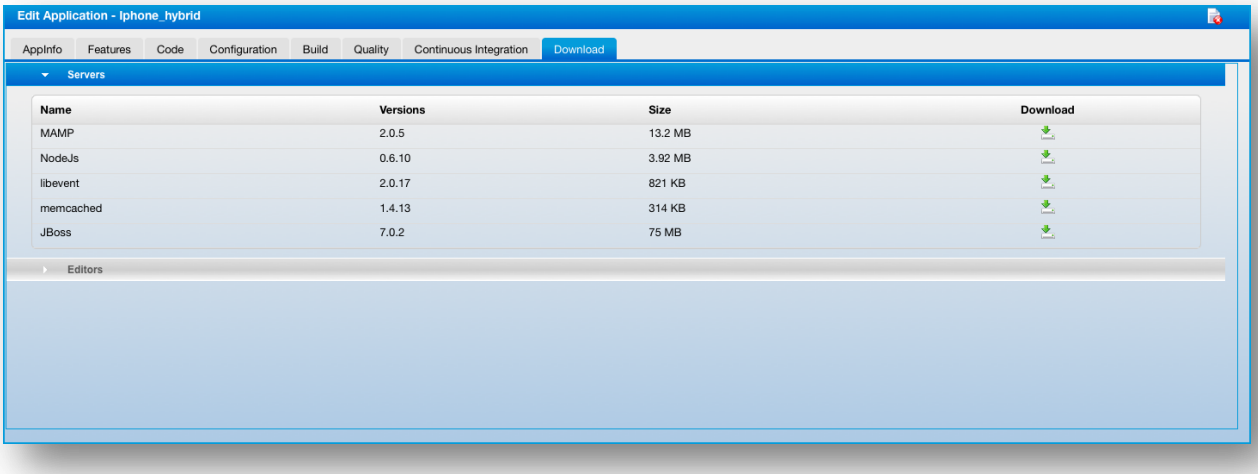


Figure 14: Screenshot for downloading servers

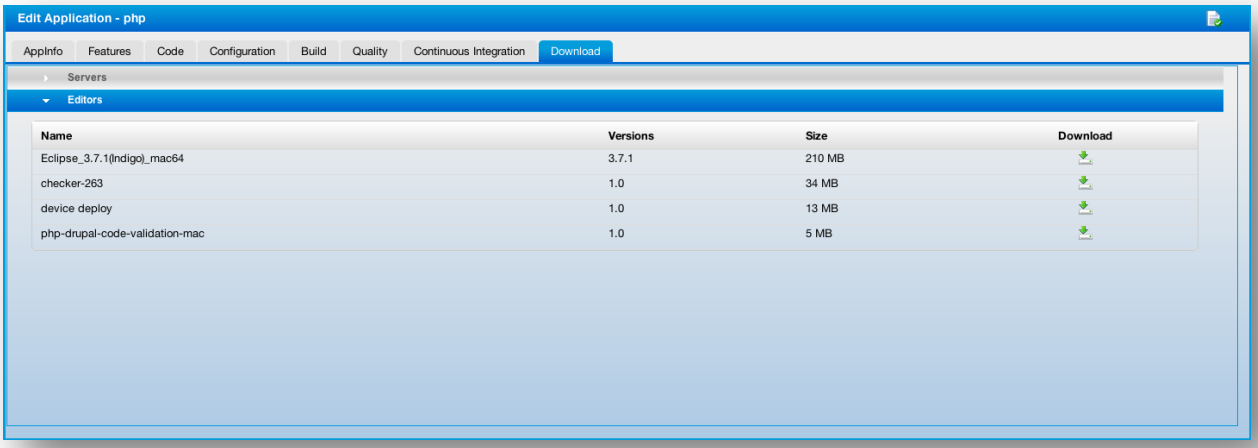


Figure 15: Screenshot for downloading editors

6 Archetypes

Archetype, provided in Phresco, is an ideal project from which similar projects can be created. Phresco 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 three different applications - web, mobile and stand alone applications. Under each applications there are list of Archetypes available as given below.

List of Archetypes for Web Applications

- PHP
- Drupal6
- Drupal7
- Sharepoint
- ASP .Net
- Wordpress
- Java WebService
- NodeJS
- HTML5 Multi Widget
- HTML5 Mobile Widget

List of Archetypes for Mobile Applications

- iPhone Native
- iPhone Hybrid
- Android Native
- Android Hybrid

List of Archetypes for Stand Alone Applications

- Java StandAlone
- NodeJS

7 Reusable Features

Phresco promotes reusability of components. As the word prescribes reusable features are the features that can be reused in any project whenever required. Phresco has added many open source pre built components. Users also have the facility to upload reusable features required for their project using the Admin console.

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

Technology	Content	Selecting a feature	Selecting a JS library
Drupal	Drupal module in zip format	The module will be added into <project_home/source/sites/all/modules	The js file will be added into <project_home>/source/sites/all/themes/jslib

8 Testing

Phresco provides standardized structure for testing in each technology.

8.1 Test cases for PHP technology

8.1.1 Unit test cases

8.1.1.1 Structure of alltest and test cases

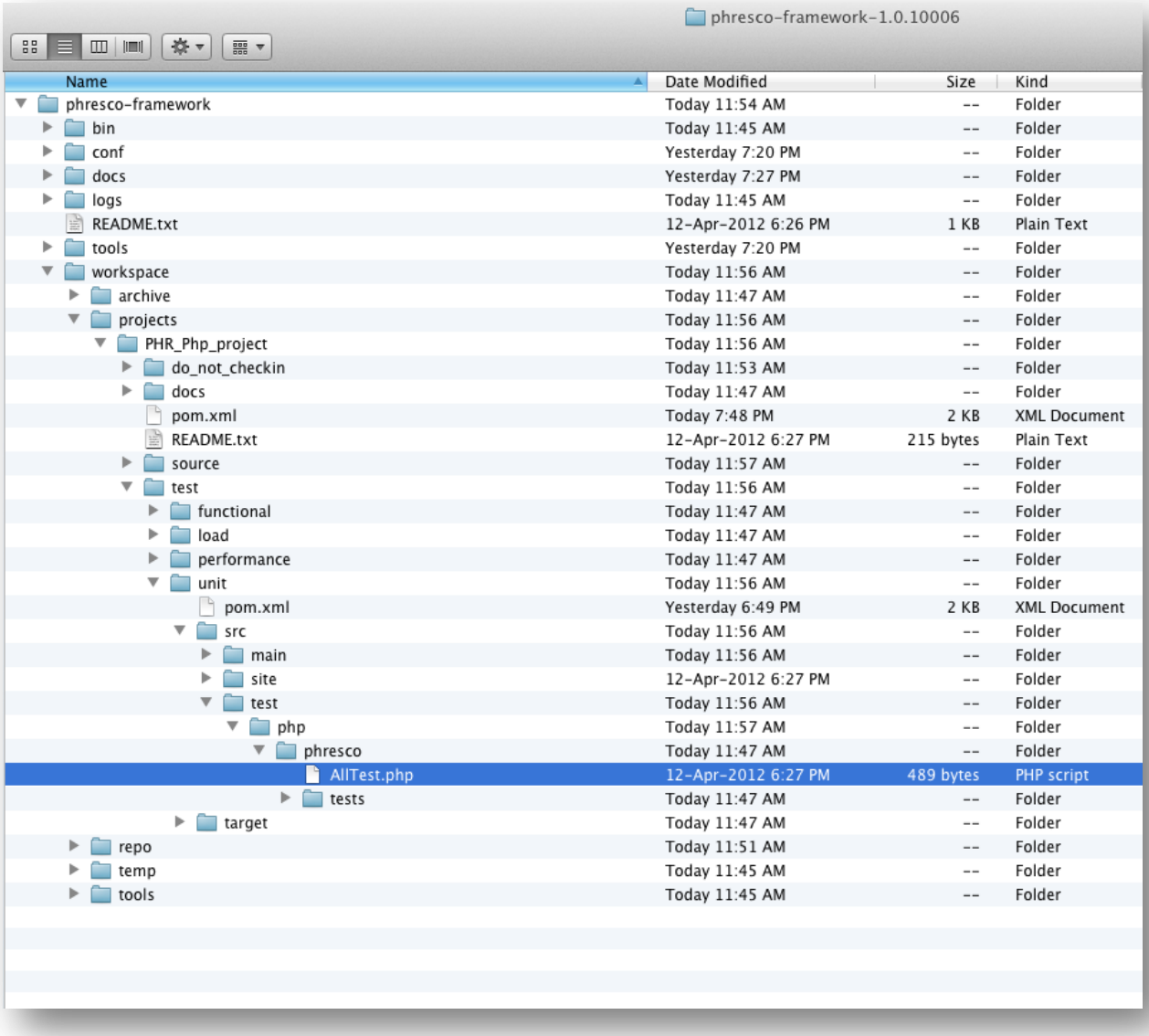


Figure 16: Screenshot for the structure of all test and test cases

a. All test : 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: Test cases which are called can be tested manually by entering the data which are defined in the base screen

8.1.1.2 Existing Test Cases and Suites Out of the Box in Phresco

AllTest for PHP

AllTest is the root category which holds all the test cases. AllTest can call any number of test cases depending on the requirements. Developers / testers can start writing new test cases by following the syntax and structure of the phresco framework's out of the box test case structure. Once test cases are written, developers can execute them using the Phresco Framework and analyse the report generated.

The following code structure is a test case inclusion/addition illustration of a sample AllTest file for username validation.

```
<?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(){

}

}
```

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 build will succeed only if the characters are correct. Testing the username is one unit test case and the test cases can be many in number.

```
<?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'));
      }
}
?>
```

8.1.1.3 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.

```
$testSuite = new AllTest('Phspunittest');
            $testSuite->addTest(new UsernameValidation("testValidation"));
            $testSuite->addTest(new DateValidation("testDate"));
            $testSuite->addTest(new EmailVerification("testEmail"));  newly added test case
```

Provided the required files should be called.

8.1.1.4 Report generated after testing

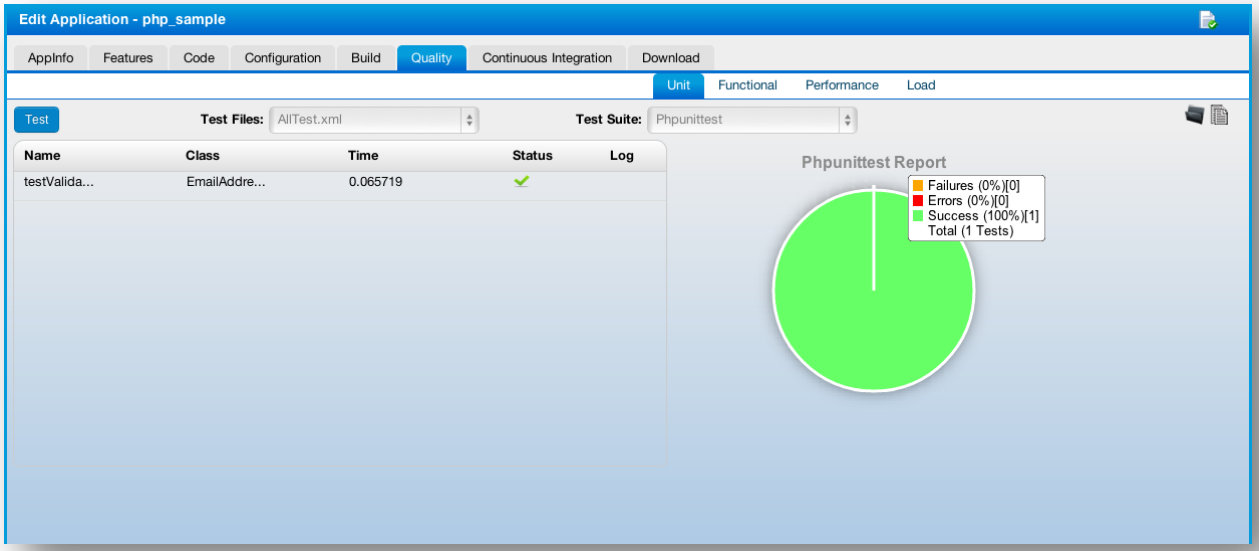


Figure 17: Screenshot for Report generated after testing

8.1.2 Functional test case

8.1.2.1 Structure of Test Suites and Test Case

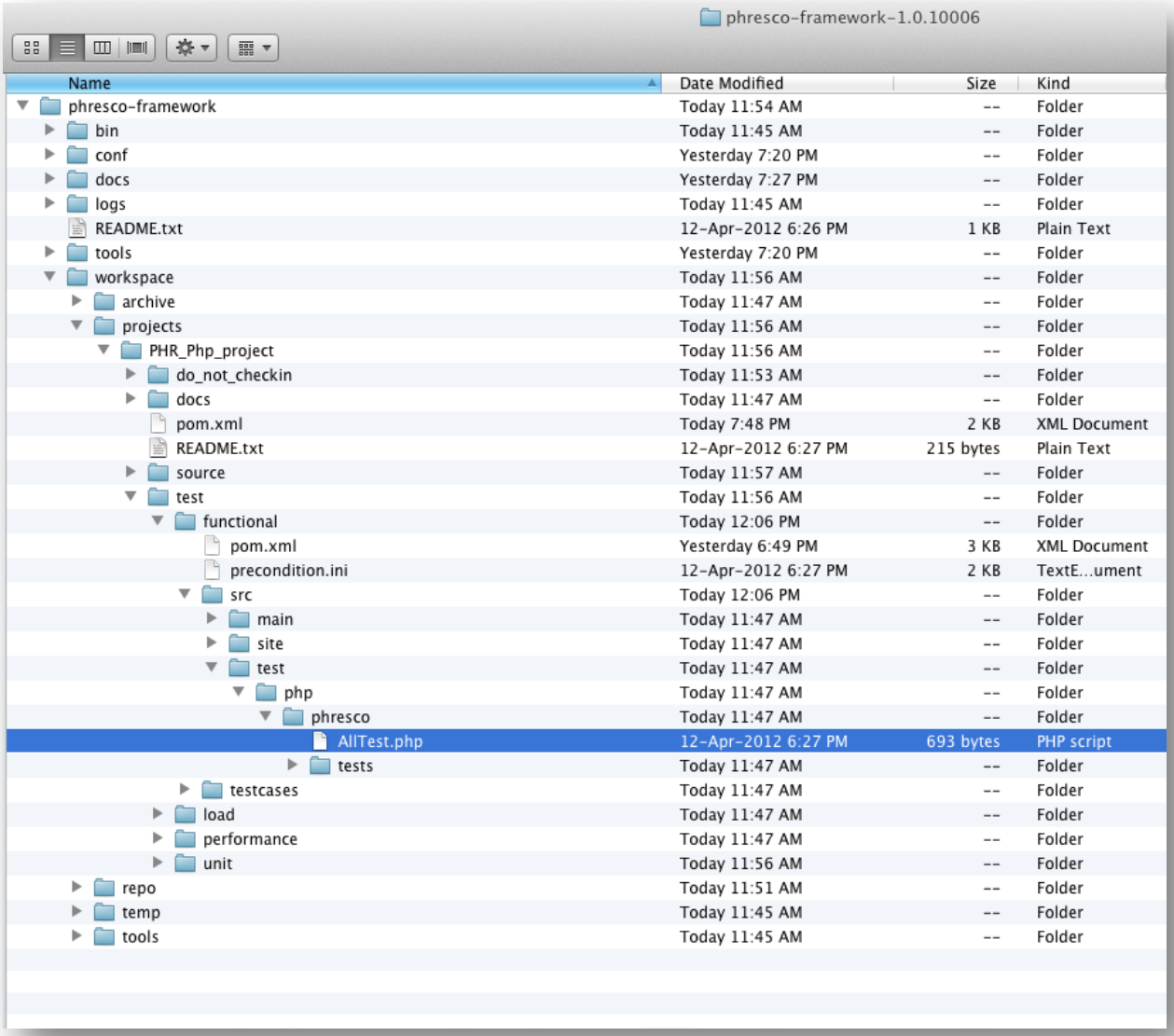


Figure 18: Structure for alltest and test cases

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

8.1.2.2 Existing Test Cases and Suites Out of the Box in Phresco

AllTest for PHP

AllTest is the root category which holds all the test suites. Functional test runs in the order by which the test suites are created. AllTest can call any number of test suites depending on the requirements. Developers / testers can start writing new test suites and test cases by following the syntax and structure of the phresco framework's out of the box test suites structure. AllTest calls the test suites and test suite calls all the test cases written in the test suite. Once test suite and test cases are written developers can execute them from Phresco Framework and can see the report. Following is the sample AllTest file which shows up how to add / include the test suites inside it.

```
<?php

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

class AllTest extends PHPUnit_Framework_TestSuite {

        protected function setUp() {
                parent::setUp();
        }

        public static function suite() {
                $testSuite = new AllTest('AllTestSuite');
                $testSuite->addTestSuite('UserModules');
                $testSuite->addTestSuite('SearchModules');
                $testSuite->addTestSuite('DbUpdateModules');
                return $testSuite;
        }

        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 same syntax.

```
<?php

require_once 'Register_NewUser.php';
require_once 'LoginLogout_User.php';

class UserModules extends PHPUnit_Framework_TestSuite
{
    protected function setUp(){
            parent::setUp();
    }
    public static function suite(){
            $testSuite = new UserModules('UserModules');
            $testSuite->addTest(new Register_NewUser("testRegisters"));
                $testSuite->addTest(new LoginLogout_User("testLoginLogout_User"));
            $testSuite->addTest(new Account_Update("testAccountUpdate"));
            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.

```
<?php

require_once 'PhpCommonFun.php';
require_once 'PHPUnit/Extensions/SeleniumTestCase.php';

class Register_NewUser extends PhpCommonFun{

  protected function setUp(){
            parent::setUp();
  }
  public function testRegisters(){
            parent::Browser();

            $property = new PhpCommonFun;
            $properties = $property->UserInfo();

            try {
                    if ($this->isElementPresent(PHP_REG_LINK))
```

```
                $property->waitForElementPresent('PHP_REG_LINK');
                $this->clickAndWait(PHP_REG_LINK);
                $property->waitForElementPresent('PHP_REG_UNAME_TBOX');
        }
                catch (Exception $e) {}
                $this->type(PHP_REG_UNAME_TBOX,$properties['Username']);
                $property->waitForElementPresent('PHP_REG_EMAIL_TBOX');
                $this->type(PHP_REG_EMAIL_TBOX,$properties['Emailaddress']);
                $property->waitForElementPresent('PHP_REG_PASS_TBOX');
                $this->type(PHP_REG_PASS_TBOX,$properties['Password']);
                $property->waitForElementPresent('PHP_REG_SUBMIT');
                $this->clickAndWait(PHP_REG_SUBMIT);
                $property->waitForElementPresent('PHP_REG_MSG');
                try {
                $this->assertTrue($this->isTextPresent(PHP_REG_MSG));
                }
                catch (Exception $e) {
                parent::doCreateScreenShot(__FUNCTION__);
                array_push($this->verificationErrors, $e->toString());
                }
    }
    public function tearDown(){
                parent::tearDown();
    }

}
?>
```

8.1.2.3 To add new test suite in AllTest

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

```
$testSuite = new AllTest('AllTestSuite');
                $testSuite->addTestSuite('UserModules');
                $testSuite->addTestSuite('SearchModules');
                $testSuite->addTestSuite('DbUpdateModules');
                $testSuite->addTestSuite('DbDeleteModules');➔ newly added test suite
```

8.1.2.4 To add new test case in Test suite

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

```
$testSuite = new UserModules('UserModules');
$testSuite->addTest(new Register_NewUser("testRegisters"));
$testSuite->addTest(new LoginLogout_User("testLoginLogout_User"));
$testSuite->addTest(new Account_Update("testAccountUpdate"));➔ newly added
                                                                Test suite
```

8.1.2.5 Report generated after testing

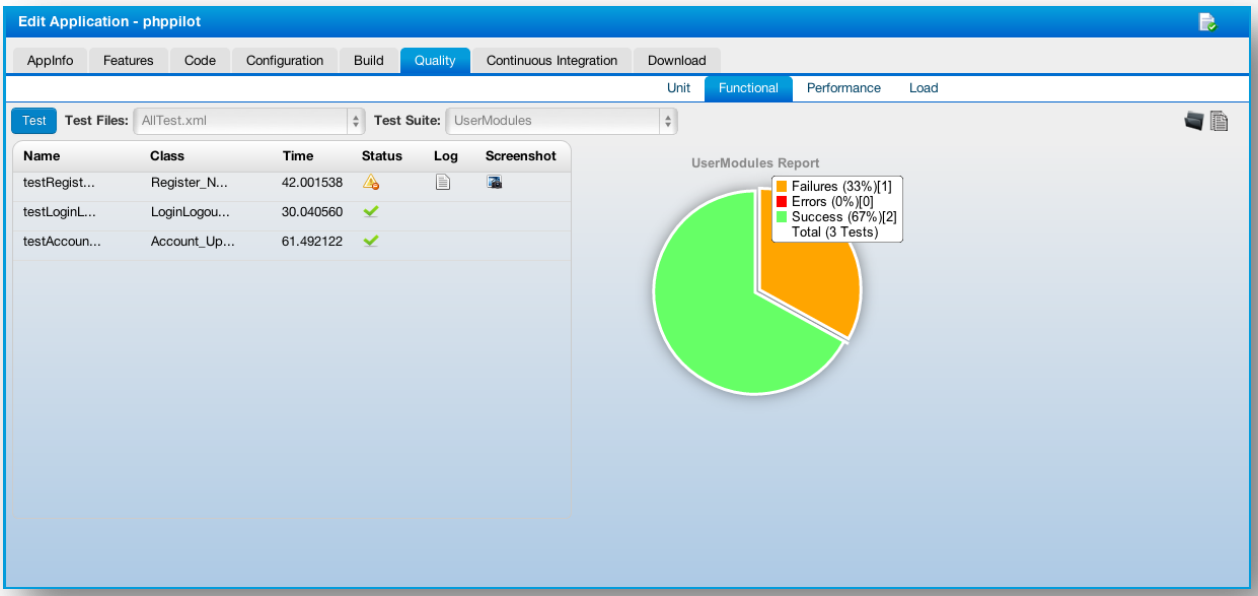


Figure 19: Screenshot for report generated after testing

8.2 Test cases for SharePoint technology

8.2.1 Unit test case

8.2.1.1 Structure of all test and test cases

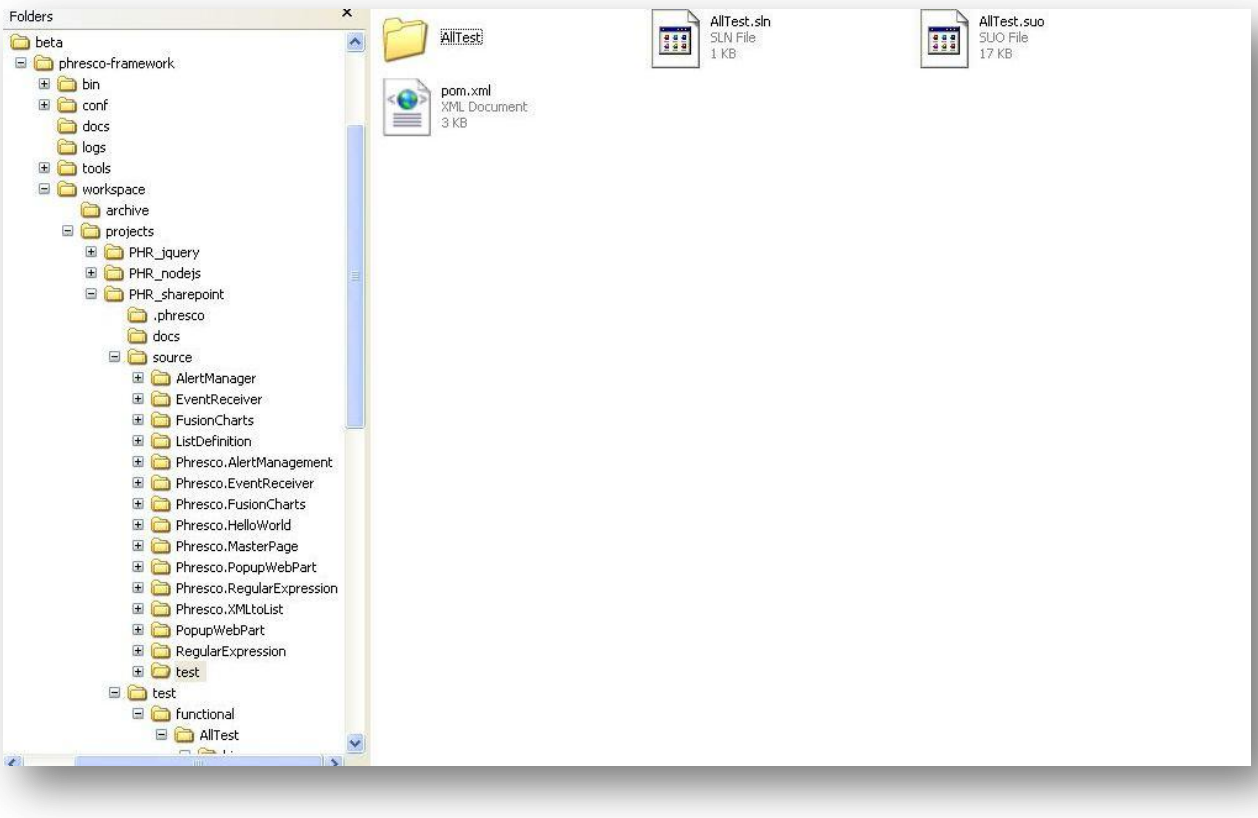


Figure 20: Screenshot for all test and test cases

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

b. Class: All the classes is incorporated in the All Test

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

Alltest for sharepoint

All Test is the root category which holds all the classes. Unit test runs automatically by the alphabetical order of classes. All Test can have any number of classes depending on the requirements. Developers / testers can start writing new classes and test cases by following the syntax and structure of the phresco framework's out of the box class structure. All test calls all the classes and the test

cases inside them. Once class and test cases are written developers can execute them from Phresco Framework and can see the report. Following is the sample 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. Class is a collection of test cases.

```
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 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.

8.2.1.2 Report generated after testing

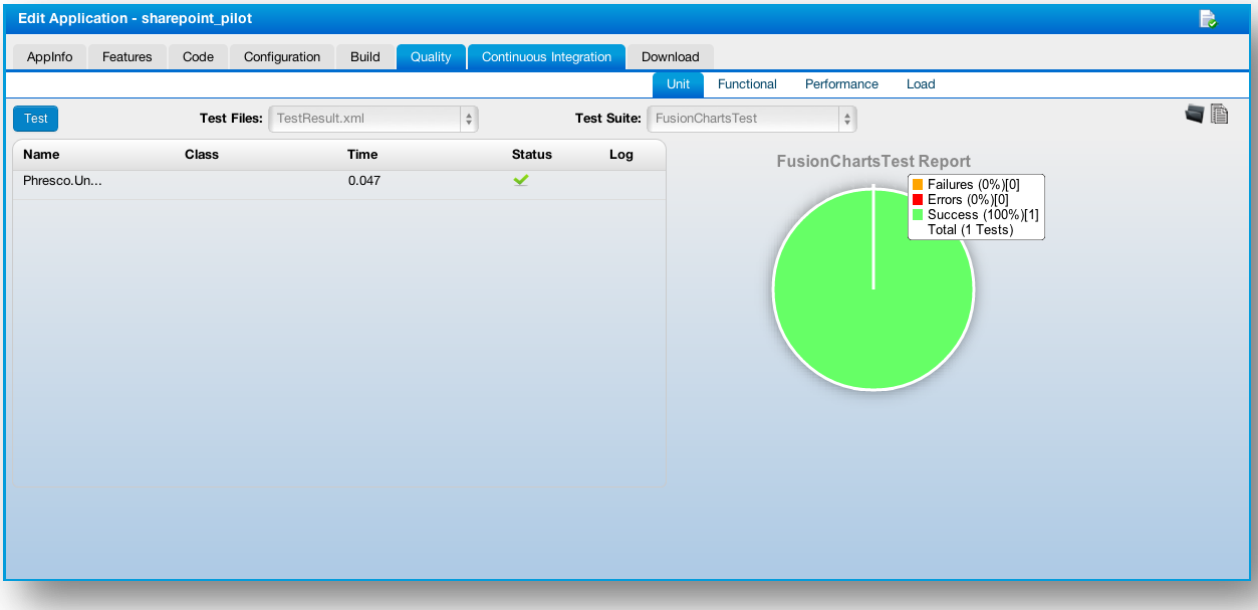


Figure 21: Screenshot for report generated after testing

8.2.2 Functional test case

8.2.2.1 Structure of classes and test case

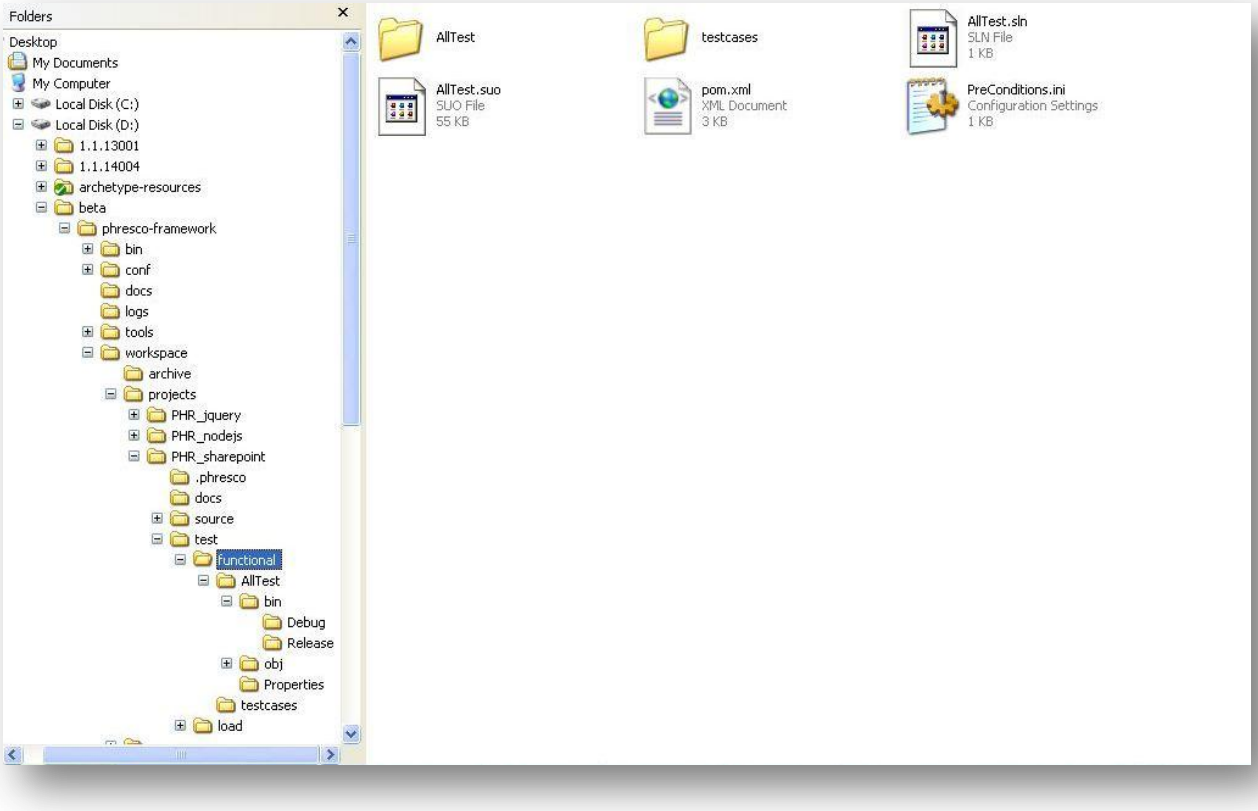


Figure 22: Screenshot for structure of all test and test cases

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

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

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

8.2.2.2 Existing Test Cases and classes Out of the Box

All Test for Share point

All Test is the root category which holds all the classes. Functional test runs automatically by the alphabetical order of classes. All Test can have any number of classes depending on the requirements. Developers / testers can start writing new classes and test cases by following the syntax and structure of the phresco framework's out of the box class structure. All test calls all the classes and the test cases inside them. Once class and test cases are written developers can execute them from Phresco Framework and can see the report. Following is the sample AllTest file which shows up how to add / include the class inside it.

```
using System;
using System.Configuration;
using System.Data;
using System.IO;
using System.Text;
using System.Threading;
using NUnit.Framework;
using OpenQA.Selenium;
using OpenQA.Selenium.IE;
using Selenium;
namespace AllTest{

  [TestFixture]
  public class HelloWorld {

    public IWebDriver driver;
    private StringBuilder verificationErrors;
    String baseUrl, sceenshotpath;
    [SetUp]
    public void SetupTest() {

      DataSet helloworld = new DataSet();
      helloworld.ReadXml(ConfigurationSettings.AppSettings["XmlPath"].ToString());
       baseUrl = helloworld.Tables[1].Rows[0].ItemArray[1].ToString() + ":" + "//" +
      helloworld.Tables[1].Rows[0].ItemArray[2].ToString() + ":" +
      helloworld.Tables[1].Rows[0].ItemArray[3].ToString() + "/" +
      helloworld.Tables[1].Rows[0].ItemArray[0].ToString();
      try {

        driver = new InternetExplorerDriver();
        driver.Navigate().GoToUrl(baseUrl);
        verificationErrors = new StringBuilder();
      }
      catch (Exception e) {

        Assert.AreEqual(helloworld.Tables[0].Rows[0].ItemArray[2].ToString(), e.ToString());
      }
```

```
        }

        [TearDown]
        public void TeardownTest() {

            try {

                driver.Quit();
            }
            catch (Exception){

                // Ignore errors if unable to close the browser
            }
            Assert.AreEqual("", verificationErrors.ToString());
        }
        public void TakeScreenshot(IWebDriver driver, string path) {

            ITakesScreenshot screenshotDriver = driver as ITakesScreenshot;
            Screenshot screenshot = screenshotDriver.GetScreenshot();
            screenshot.SaveAsFile(path, System.Drawing.Imaging.ImageFormat.Png);
            screenshot.ToString();
        }

        [Test]
        public void helloWorldTest() {

            DataSet helloworld = new DataSet();
            helloworld.ReadXml(ConfigurationSettings.AppSettings["XmlPath"].ToString());
            sceenshotpath=helloworld.Tables[0].Rows[0].ItemArray[10].ToString();
            baseUrl = helloworld.Tables[1].Rows[0].ItemArray[1].ToString() + ":" + "//" +
helloworld.Tables[1].Rows[0].ItemArray[2].ToString() + ":" +
helloworld.Tables[1].Rows[0].ItemArray[3].ToString() + "/" +
helloworld.Tables[1].Rows[0].ItemArray[0].ToString();
            Directory.CreateDirectory(helloworld.Tables[0].Rows[0].ItemArray[9].ToString());
           try {

              //Opens Phresco Home Page.
              driver.Navigate().GoToUrl(baseUrl);
              //Click on site actions and go to edit page.

driver.FindElement(By.XPath(helloworld.Tables[0].Rows[0].ItemArray[4].ToString())).Click();

driver.FindElement(By.XPath(helloworld.Tables[0].Rows[0].ItemArray[5].ToString())).Click();
              Thread.Sleep(Convert.ToInt32(helloworld.Tables[0].Rows[0].ItemArray[3].ToString()));
              //Close the hello world web part.

driver.FindElement(By.XPath(helloworld.Tables[0].Rows[0].ItemArray[6].ToString())).Click();
              Thread.Sleep(Convert.ToInt32(helloworld.Tables[0].Rows[0].ItemArray[3].ToString()));
              driver.Navigate().Refresh();
              driver.Navigate().GoToUrl(baseUrl);

              }
            catch (Exception e) {

               TakeScreenshot(driver, helloworld.Tables[0].Rows[0].ItemArray[10].ToString());
               throw e;

              }
```

```
        }
    }
}
```

Class example for Chartlist

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.class is a collection of test cases

```
using System;
using System.Configuration;
using System.Data;
using System.IO;
using System.Text;
using System.Threading;
using NUnit.Framework;
using OpenQA.Selenium;
using OpenQA.Selenium.IE;
using Selenium;
using OpenQA.Selenium.Support.UI;

namespace ChartList_Create_Delete{

  [TestFixture]
  public class Chart_Create_Delete {

    public IWebDriver driver;
    private StringBuilder verificationErrors;
    String baseUrl;
    [SetUp]
    public void SetupTest() {

      DataSet phresco = new DataSet();
      phresco.ReadXml(ConfigurationSettings.AppSettings["XmlPath"].ToString());
      baseUrl = phresco.Tables[1].Rows[0].ItemArray[0].ToString() + ":" + "//" +
phresco.Tables[1].Rows[0].ItemArray[1].ToString() + ":" +
phresco.Tables[1].Rows[0].ItemArray[2].ToString() + "/" +
phresco.Tables[1].Rows[0].ItemArray[3].ToString();
      try{

        verificationErrors = new StringBuilder();
        driver = new InternetExplorerDriver();
        driver.Navigate().GoToUrl(baseUrl);
      }
      catch (Exception e) {

        Assert.AreEqual(phresco.Tables[0].Rows[0].ItemArray[3].ToString(), e.ToString());
      }
    }

    [TearDown]
    public void TeardownTest(){

      try{
```

```
            driver.Quit();
        }
        catch (Exception){

            // Ignore errors if unable to close the browser
        }
        Assert.AreEqual("", verificationErrors.ToString());
    }
    public void TakeScreenshot(IWebDriver driver, string path){

        ITakesScreenshot screenshotDriver = driver as ITakesScreenshot;
        Screenshot screenshot = screenshotDriver.GetScreenshot();
        screenshot.SaveAsFile(path, System.Drawing.Imaging.ImageFormat.Png);
        screenshot.ToString();
    }
    [Test]
    public void ChartListPrepTest(){

        DataSet phresco = new DataSet();
        phresco.ReadXml(ConfigurationSettings.AppSettings["XmlPath"].ToString());
        try {

            //Opens Phresco Home Page.
            driver.Navigate().GoToUrl(baseUrl);
            Thread.Sleep(Convert.ToInt32(phresco.Tables[0].Rows[0].ItemArray[2].ToString()));
            driver.FindElement(By.XPath(phresco.Tables[0].Rows[0].ItemArray[4].ToString())).Click();
            //Opens VM_allocation List Web Page.
            driver.FindElement(By.XPath(phresco.Tables[0].Rows[0].ItemArray[5].ToString())).Click();
            Thread.Sleep(Convert.ToInt32(phresco.Tables[0].Rows[0].ItemArray[2].ToString()));
            //Opens New VM_Allocation List Definition Page.
            driver.FindElement(By.XPath(phresco.Tables[0].Rows[0].ItemArray[6].ToString())).Click();
            Thread.Sleep(Convert.ToInt32(phresco.Tables[0].Rows[0].ItemArray[2].ToString()));

driver.FindElement(By.XPath(phresco.Tables[0].Rows[0].ItemArray[7].ToString())).SendKeys(phresco.Ta
bles[0].Rows[0].ItemArray[8].ToString());

driver.FindElement(By.XPath(phresco.Tables[0].Rows[0].ItemArray[9].ToString())).SendKeys(phresco.Ta
bles[0].Rows[0].ItemArray[10].ToString());
            IWebElement selectWindows =
driver.FindElement(By.XPath(phresco.Tables[0].Rows[0].ItemArray[11].ToString()));
            SelectElement clickThis = new SelectElement(selectWindows);
            clickThis.SelectByText(phresco.Tables[0].Rows[0].ItemArray[12].ToString());

driver.FindElement(By.XPath(phresco.Tables[0].Rows[0].ItemArray[13].ToString())).SendKeys(phresco.T
ables[0].Rows[0].ItemArray[14].ToString());
            IWebElement hardwareType =
driver.FindElement(By.XPath(phresco.Tables[0].Rows[0].ItemArray[15].ToString()));
            SelectElement clickHT = new SelectElement(hardwareType);
            clickHT.SelectByText(phresco.Tables[0].Rows[0].ItemArray[16].ToString());

driver.FindElement(By.XPath(phresco.Tables[0].Rows[0].ItemArray[17].ToString())).SendKeys(phresco.T
ables[0].Rows[0].ItemArray[18].ToString());

driver.FindElement(By.XPath(phresco.Tables[0].Rows[0].ItemArray[19].ToString())).SendKeys(phresco.T
ables[0].Rows[0].ItemArray[20].ToString());
            IWebElement statusType =
driver.FindElement(By.XPath(phresco.Tables[0].Rows[0].ItemArray[21].ToString()));
```

```
            SelectElement clickActive = new SelectElement(statusType);
            clickActive.SelectByText(phresco.Tables[0].Rows[0].ItemArray[22].ToString());
            driver.FindElement(By.XPath(phresco.Tables[0].Rows[0].ItemArray[23].ToString())).Click();
            Thread.Sleep(Convert.ToInt32(phresco.Tables[0].Rows[0].ItemArray[2].ToString()));

        }
        catch (Exception e){

            TakeScreenshot(driver, phresco.Tables[0].Rows[0].ItemArray[24].ToString());
            throw e;
        }
    }
```

Test case for ChartListPrepTest

Test cases examine all the aspects including inputs and outputs. It gives the detailed steps that should 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.

```
    [Test]
    public void ChartListPrepTest(){

        DataSet phresco = new DataSet();
        phresco.ReadXml(ConfigurationSettings.AppSettings["XmlPath"].ToString());
        try {

            //Opens Phresco Home Page.
            driver.Navigate().GoToUrl(baseUrl);
            Thread.Sleep(Convert.ToInt32(phresco.Tables[0].Rows[0].ItemArray[2].ToString()));
            driver.FindElement(By.XPath(phresco.Tables[0].Rows[0].ItemArray[4].ToString())).Click();
            //Opens VM_allocation List Web Page.
            driver.FindElement(By.XPath(phresco.Tables[0].Rows[0].ItemArray[5].ToString())).Click();
            Thread.Sleep(Convert.ToInt32(phresco.Tables[0].Rows[0].ItemArray[2].ToString()));
            //Opens New VM_Allocation List Definition Page.
            driver.FindElement(By.XPath(phresco.Tables[0].Rows[0].ItemArray[6].ToString())).Click();
            Thread.Sleep(Convert.ToInt32(phresco.Tables[0].Rows[0].ItemArray[2].ToString()));

driver.FindElement(By.XPath(phresco.Tables[0].Rows[0].ItemArray[7].ToString())).SendKeys(phresco.Tabl
es[0].Rows[0].ItemArray[8].ToString());

driver.FindElement(By.XPath(phresco.Tables[0].Rows[0].ItemArray[9].ToString())).SendKeys(phresco.Tabl
es[0].Rows[0].ItemArray[10].ToString());
            IWebElement selectWindows =
driver.FindElement(By.XPath(phresco.Tables[0].Rows[0].ItemArray[11].ToString()));
            SelectElement clickThis = new SelectElement(selectWindows);
            clickThis.SelectByText(phresco.Tables[0].Rows[0].ItemArray[12].ToString());

driver.FindElement(By.XPath(phresco.Tables[0].Rows[0].ItemArray[13].ToString())).SendKeys(phresco.Tabl
es[0].Rows[0].ItemArray[14].ToString());
            IWebElement hardwareType =
driver.FindElement(By.XPath(phresco.Tables[0].Rows[0].ItemArray[15].ToString()));
            SelectElement clickHT = new SelectElement(hardwareType);
```

```
            clickHT.SelectByText(phresco.Tables[0].Rows[0].ItemArray[16].ToString());

        driver.FindElement(By.XPath(phresco.Tables[0].Rows[0].ItemArray[17].ToString())).SendKeys(phresco.Tabl
        es[0].Rows[0].ItemArray[18].ToString());

        driver.FindElement(By.XPath(phresco.Tables[0].Rows[0].ItemArray[19].ToString())).SendKeys(phresco.Tab
        les[0].Rows[0].ItemArray[20].ToString());
            IWebElement statusType =
        driver.FindElement(By.XPath(phresco.Tables[0].Rows[0].ItemArray[21].ToString()));
            SelectElement clickActive = new SelectElement(statusType);
            clickActive.SelectByText(phresco.Tables[0].Rows[0].ItemArray[22].ToString());
            driver.FindElement(By.XPath(phresco.Tables[0].Rows[0].ItemArray[23].ToString())).Click();
            Thread.Sleep(Convert.ToInt32(phresco.Tables[0].Rows[0].ItemArray[2].ToString()));

        }
        catch (Exception e) {

            TakeScreenshot(driver, phresco.Tables[0].Rows[0].ItemArray[24].ToString());
            throw e;
        }
```

8.2.2.3 To add a new class

To add a class you can just right click and add new class. Name of the class can be entered and saved. Any number of test cases can be added in this new class

8.2.2.4 To add new test case in class

You can add a new test case to the class using the following method. Here is an example for creating a new test case in the name 'testAccountUpdate'.

8.2.2.5 Report generated after testing

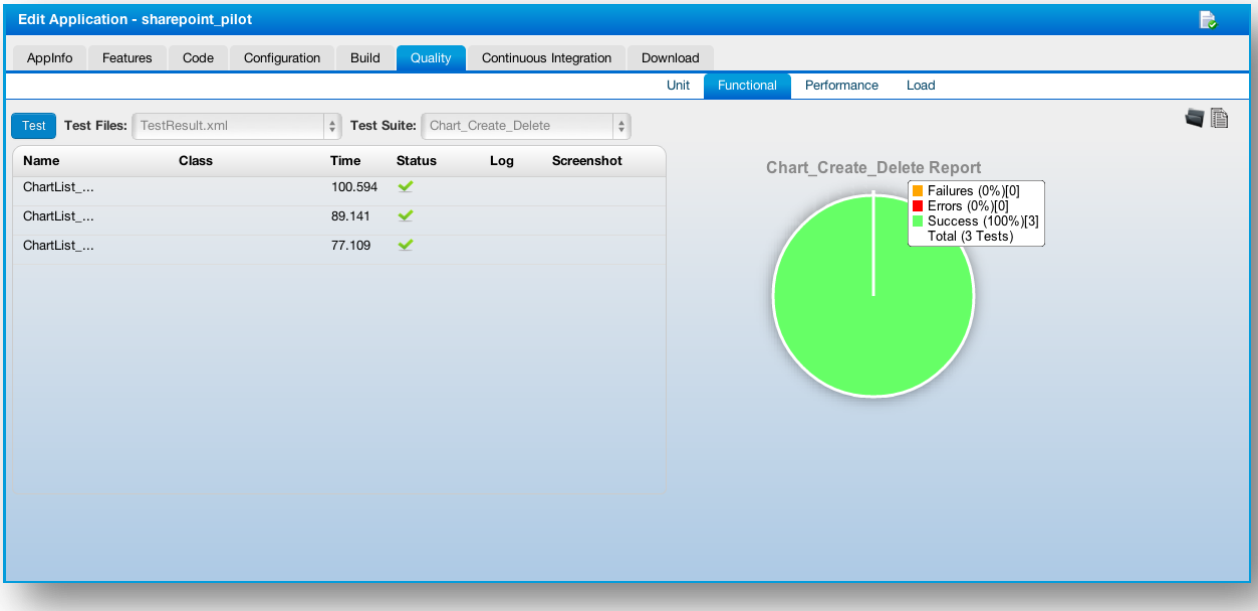


Figure 23: Screenshot for report generated after testing

8.3 Test cases for java technology

8.3.1 Unit test

8.3.1.1 Structure of alltest and test cases

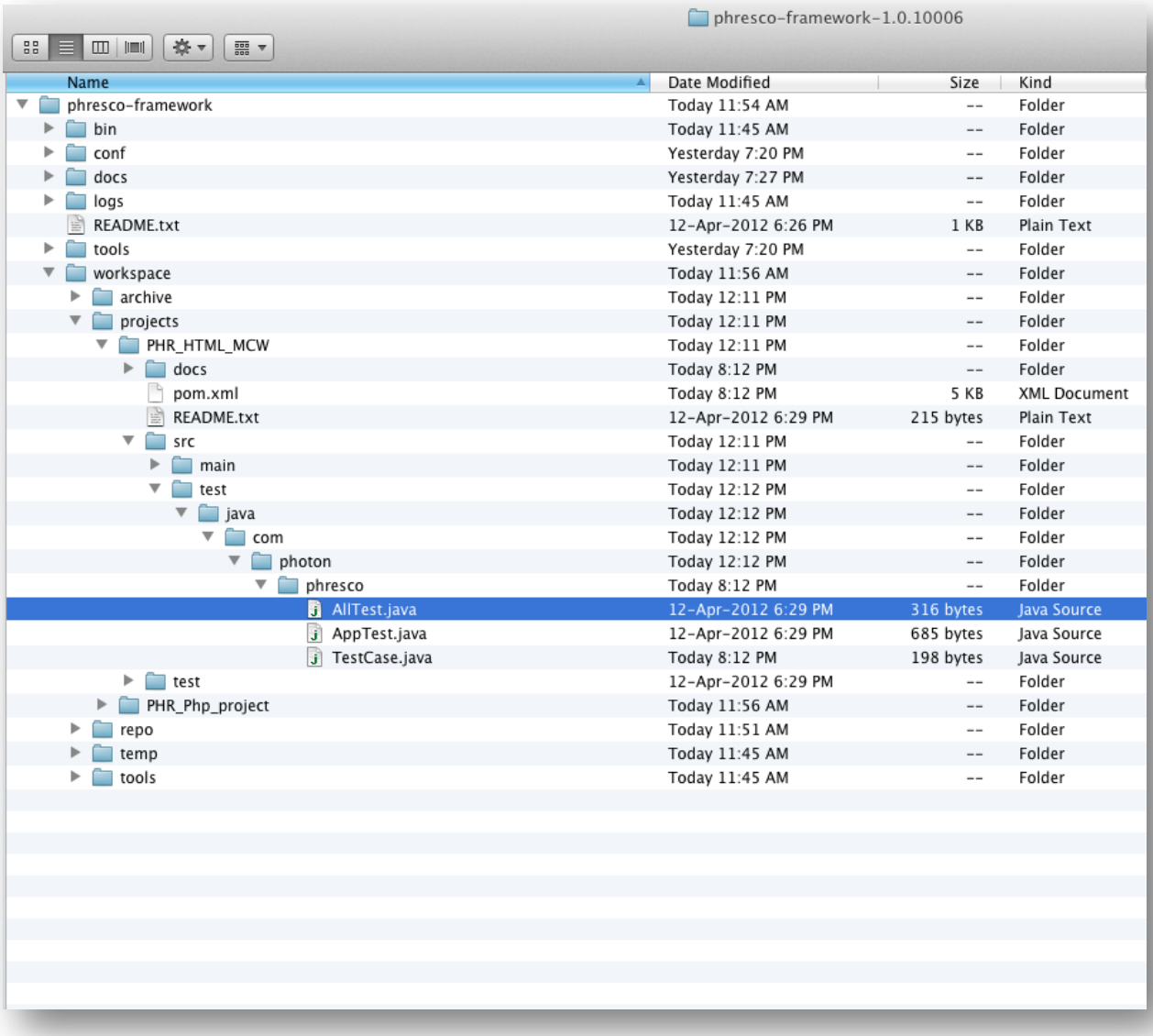


Figure 24: Screenshot for structure of all test and test cases

a. **All test :** Alltest is the root file that carries all the test cases to initiate the testing process. Each test cases 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.

8.3.1.2 Existing Test Cases Out of the Box in Phresco

AllTest for Java technology

AllTest is the root category which holds all the test cases. AllTest can call any number of test cases depending on the requirements. Developers / testers can start writing new test cases by following the syntax and structure of the phresco framework's out of the box test case structure. Once test cases are written developers can execute them from Phresco Framework and can see the report. Following is the sample AllTest file which shows up how to add / include the test cases 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

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 build will succeed only if the characters match. Testing the username is one unit test case and the test cases can be many in number.

```
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 );
    }
}
```

8.3.1.3 Report generated after testing

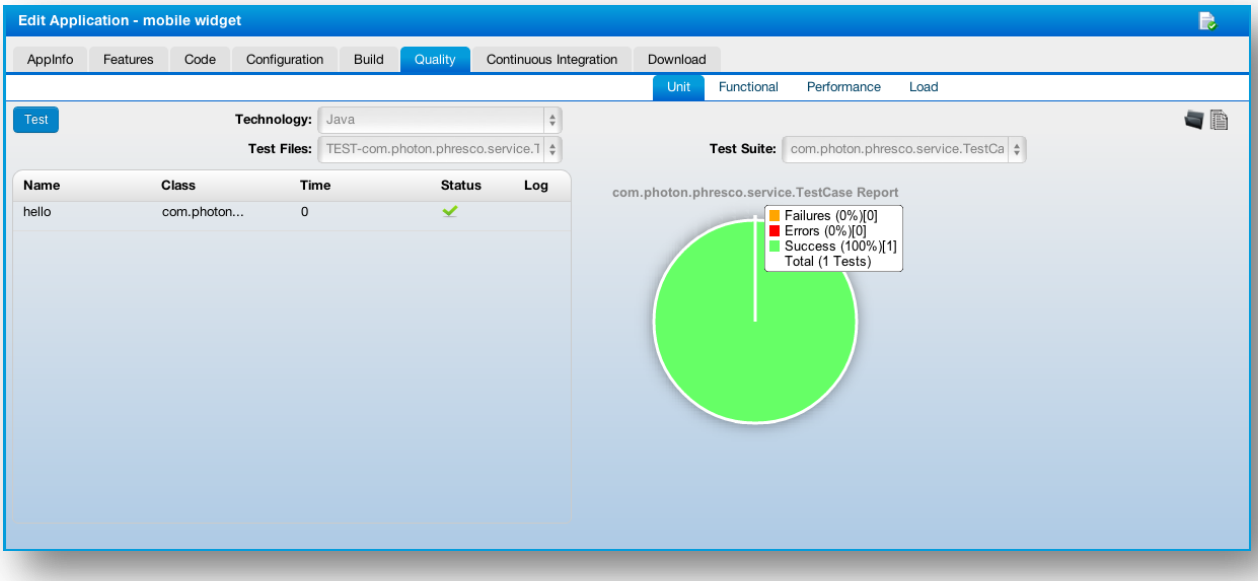


Figure 25: Screenshot for report generated after testing

8.3.2 Functional Test cases

8.3.2.1 Structure of functional test in java

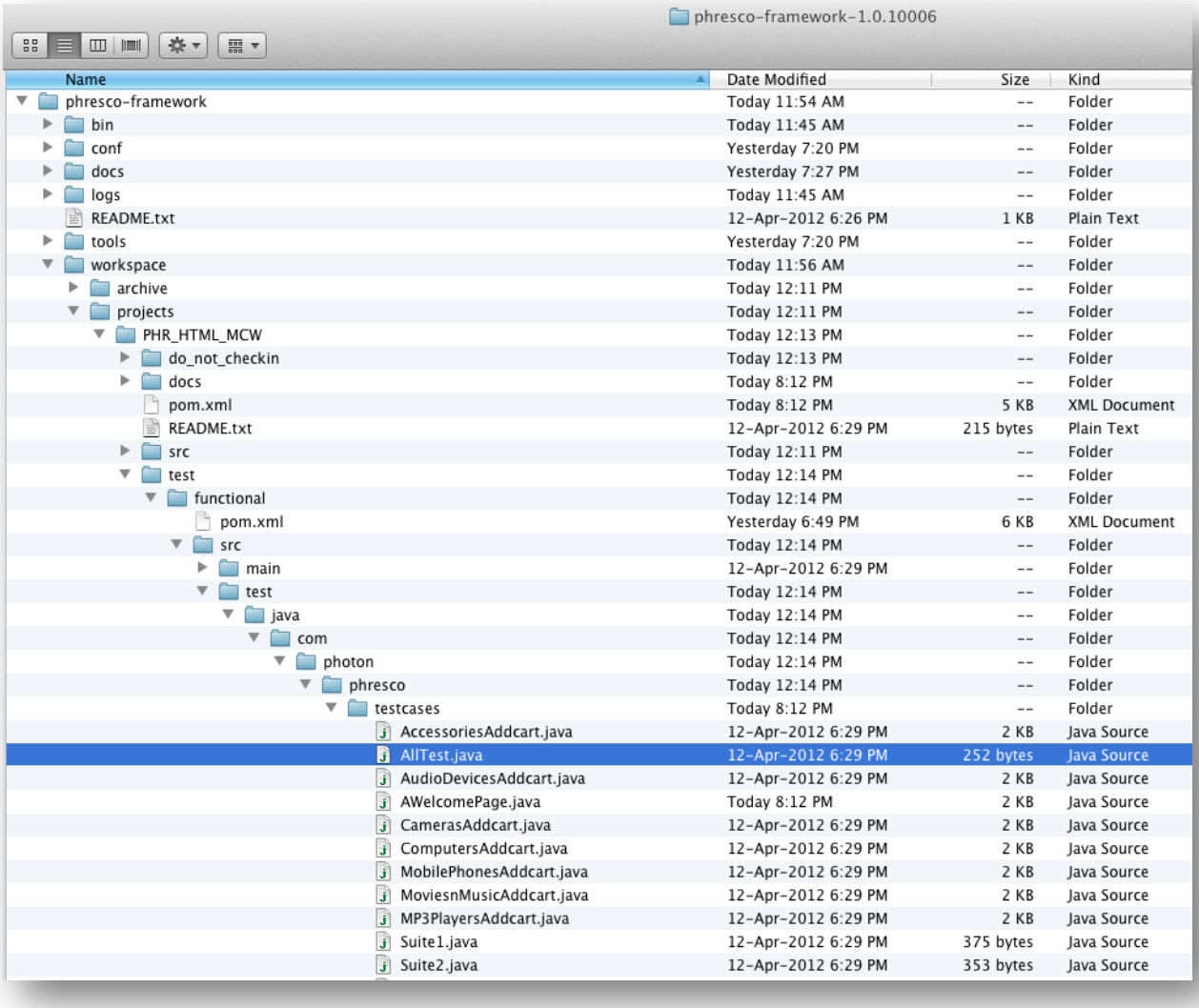


Figure 26: screenshot for structure of all test and test case

a. All Test: This is a root folder/file that calls all the suite classes to initiate the testing process. In java technology all test is also called as suite classes.

b. Suite Class: Suite class calls the test cases specified within it.

c. Test Case: All the test cases can be written separately and can be called by the suite.

8.3.2.2 Existing Test Cases and Suites Out of The Box In Phresco

Alltest (suite class)

All Test is the root category which calls suite classes. All Test can have any number of suite classes depending on the requirements. Developers / testers can start writing new suite classes and test cases by following the syntax and structure of the phresco framework's out of the box class structure. All test calls all the suite classes and the test cases inside them. Once suite classes and test cases are written developers can execute them from Phresco Framework and can see the report. Following is the sample 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 })
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

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. Everything is a class in java technology. A new test case is also a class which can be created in any test suite or it can be called separately.

```
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();
            }
    }

}
```

8.3.2.3 To add new test suite in all test

You can add a new test suite to the AllTest using the following method. Here is an example for creating a new test suite in the name Suite2

```
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})     newly added test suite
public class AllTest {
}
```

8.3.2.4 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      newly created test case
                })
public class Suite1 {

}
```

8.3.2.5 Report generated after testing

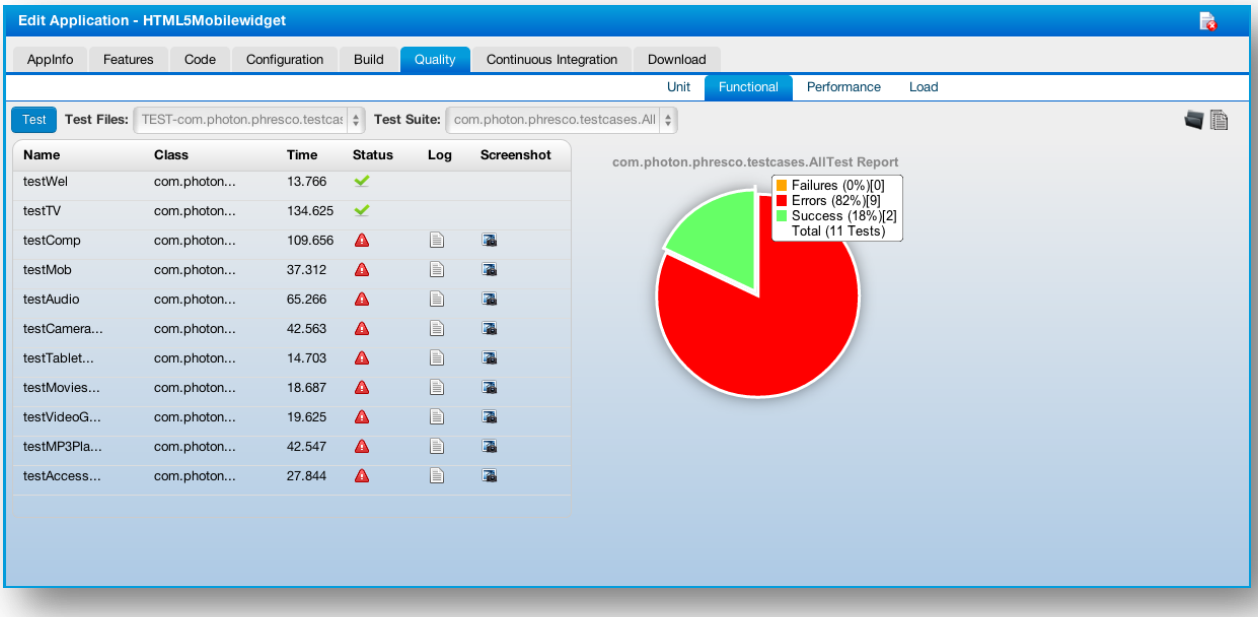


Figure 27: Screenshot for report generated after testing

8.4 Android application

8.4.1 Unit test case

8.4.1.1 Structure of alltest and test cases

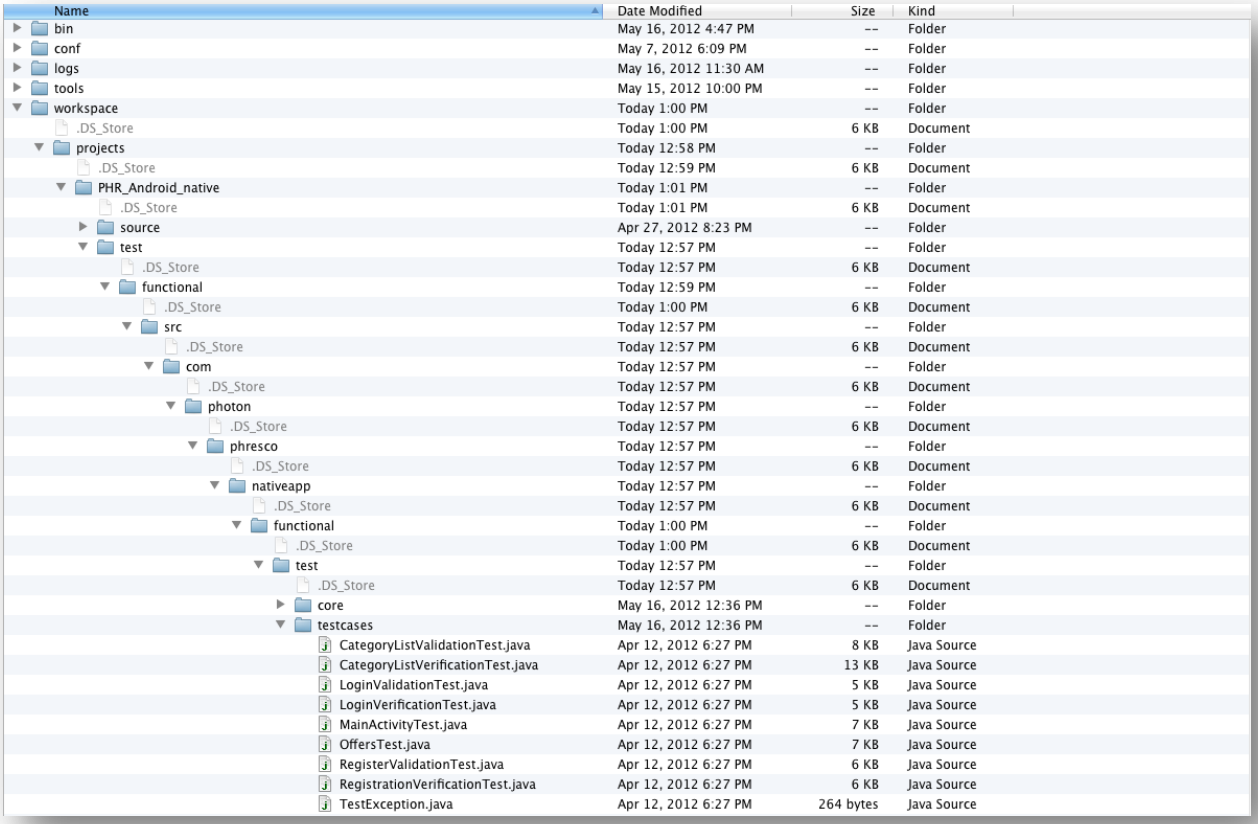


Figure 28: Screenshot for structure of all test and test cases.

a. **All test/test suite :** Alltest is the root file that carries all the test classes to initiate the testing process. Each test classes can be called within the Alltest.

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

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

8.4.1.2 Existing Test Cases Out of the Box in Phresco

AllTest for Android technology

AllTest is the root category which holds all the test classes. AllTest can call any number of test classes depending on the requirements. Developers / testers can start writing new test cases by following the syntax and structure of the phresco framework's out of the box test case structure. Once test cases are written developers can execute them from Phresco Framework and can see the report. Following is the sample AllTest file which shows up how to add / include the test cases 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 t 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;

/**
 * @author viral_b
 *
 */
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. Test method can be many in number and they are written under test classes. Test classes calls the test methods. An example of a test method 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.getWebContextURL() +
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

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. Test cases can be added in the test methods by just continuing with the next test case. Example of test case in login activity is given as an example

```
@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.getWebContextURL() +
```

```
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);
                }
}
```

8.4.1.3 To add new test method in all test

You can add a new test method to the AllTest using the following method. 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;
```

8.4.1.4 Report generated after testing

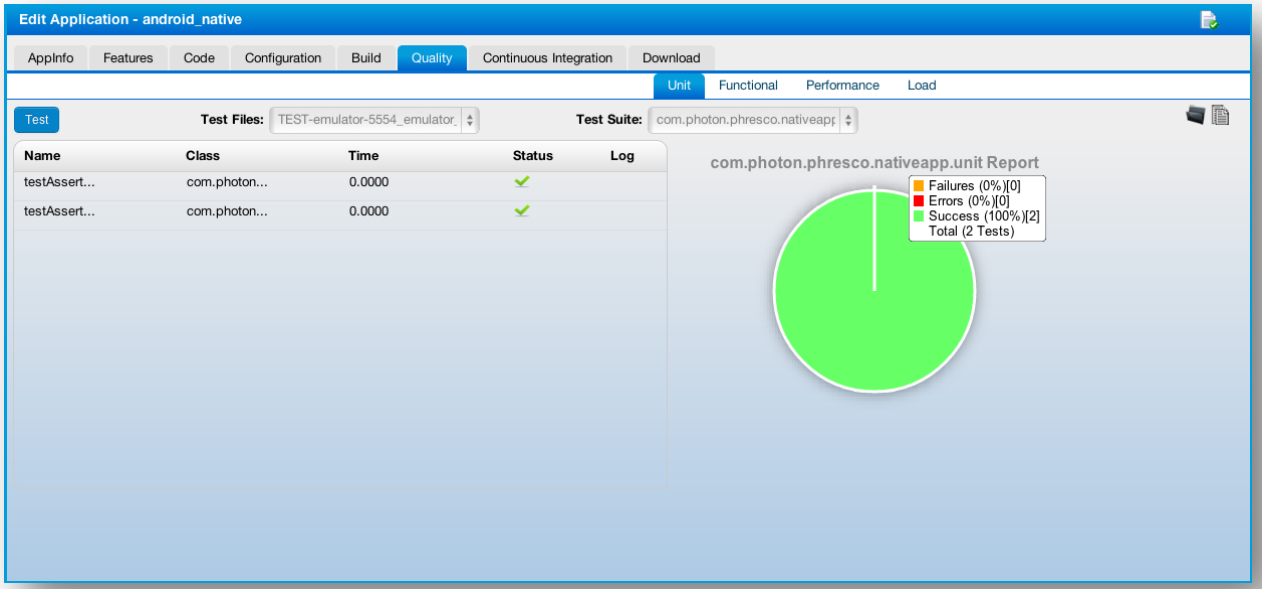


Figure 29: Screenshot for report generated after testing

8.4.2 Functional test case

8.4.2.1 Structure of functional test of android(functional)

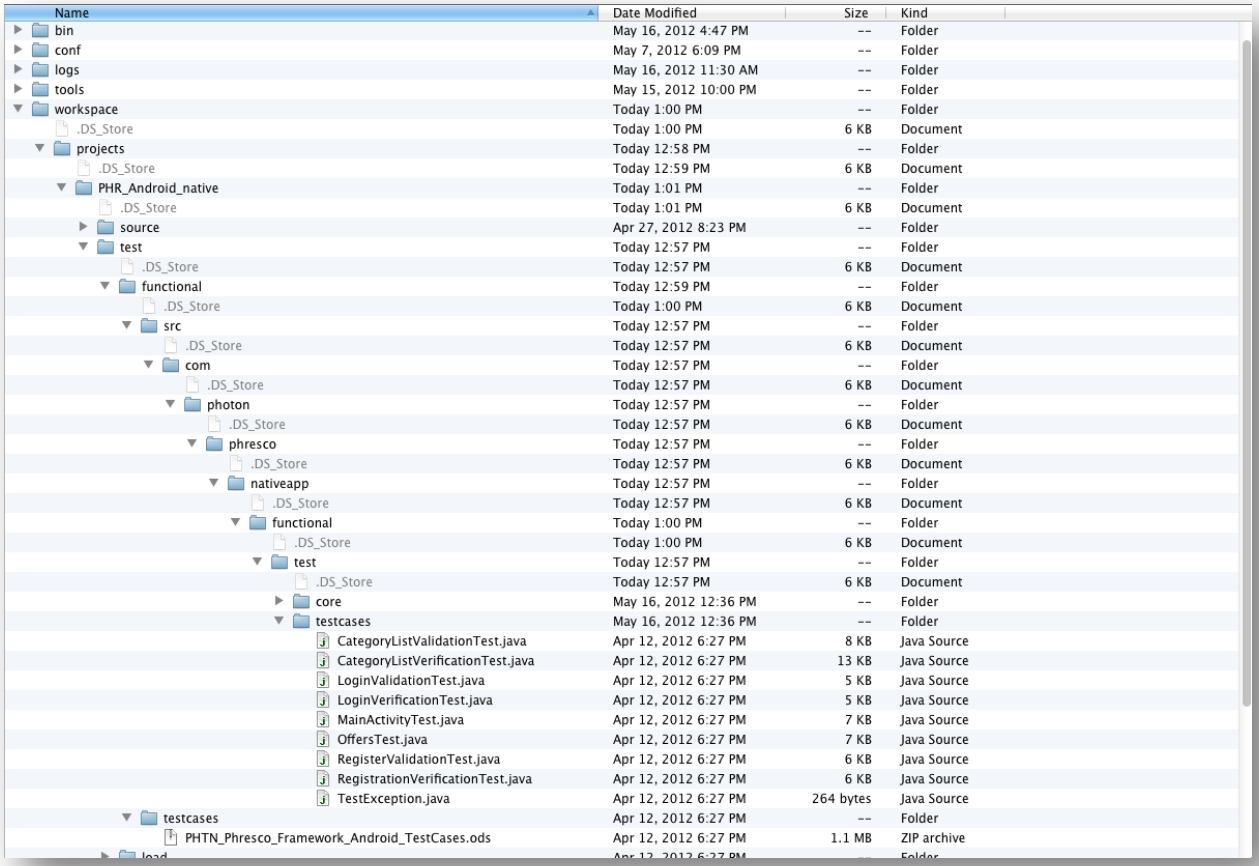


Figure 30: Screenshot for structure of all test and test cases.

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:

In java technology, main activity test is a class that calls all the test cases within itself. Developers/ testers can write any number of test cases depending upon the requirement. They can also start writing new test cases following the structure and syntax of the Phresco framework's out of the box class structure. Once test cases are written developers can execute them from Phresco Framework and can see the report. Following is the sample main activity test 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 call testLoginValidation() 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:

Test cases examine all the aspects including inputs and outputs. It gives the detailed steps that should 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. 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();
                }

        }

}
```

8.4.2.2 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 'category list verification test browse test case'

```
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 CtegoryListVerificationTest browseTestCase;        newly added test case
  private final String TAG = "MainTestCase****";
```

8.4.2.3 Report generated after testing

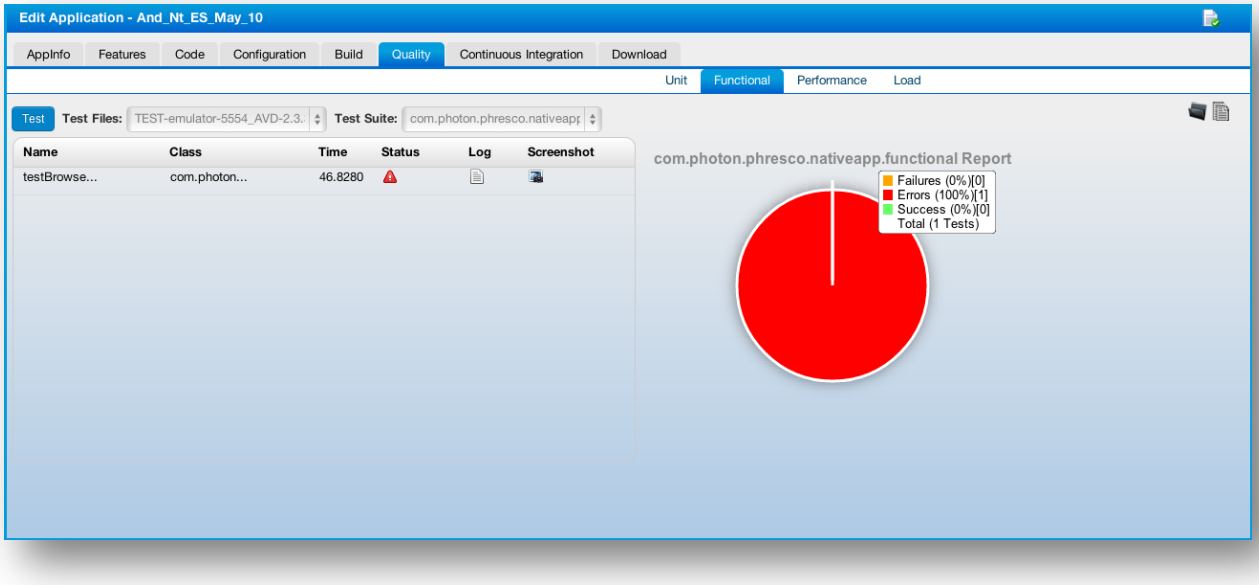


Figure 31: Screenshor for report generated after testing

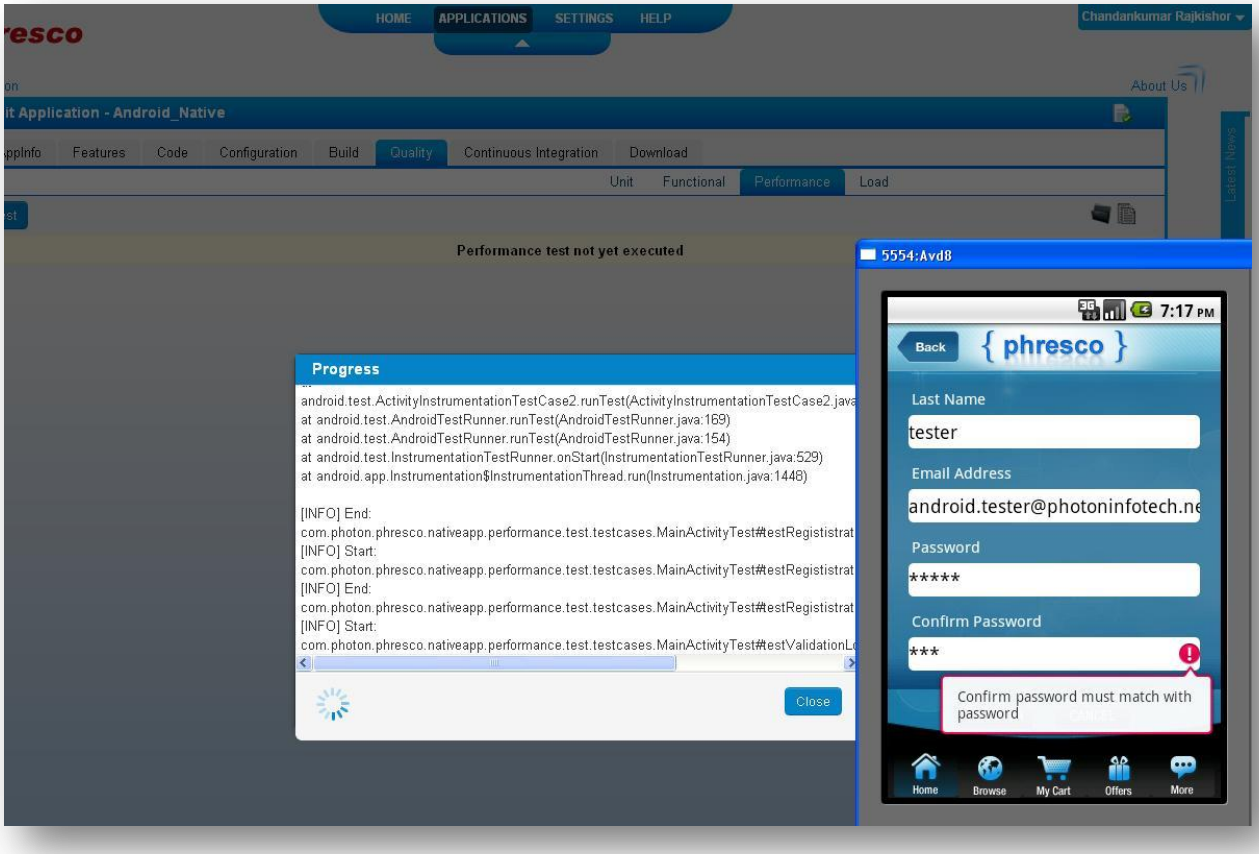


Figure 32: Screenshot for functional test

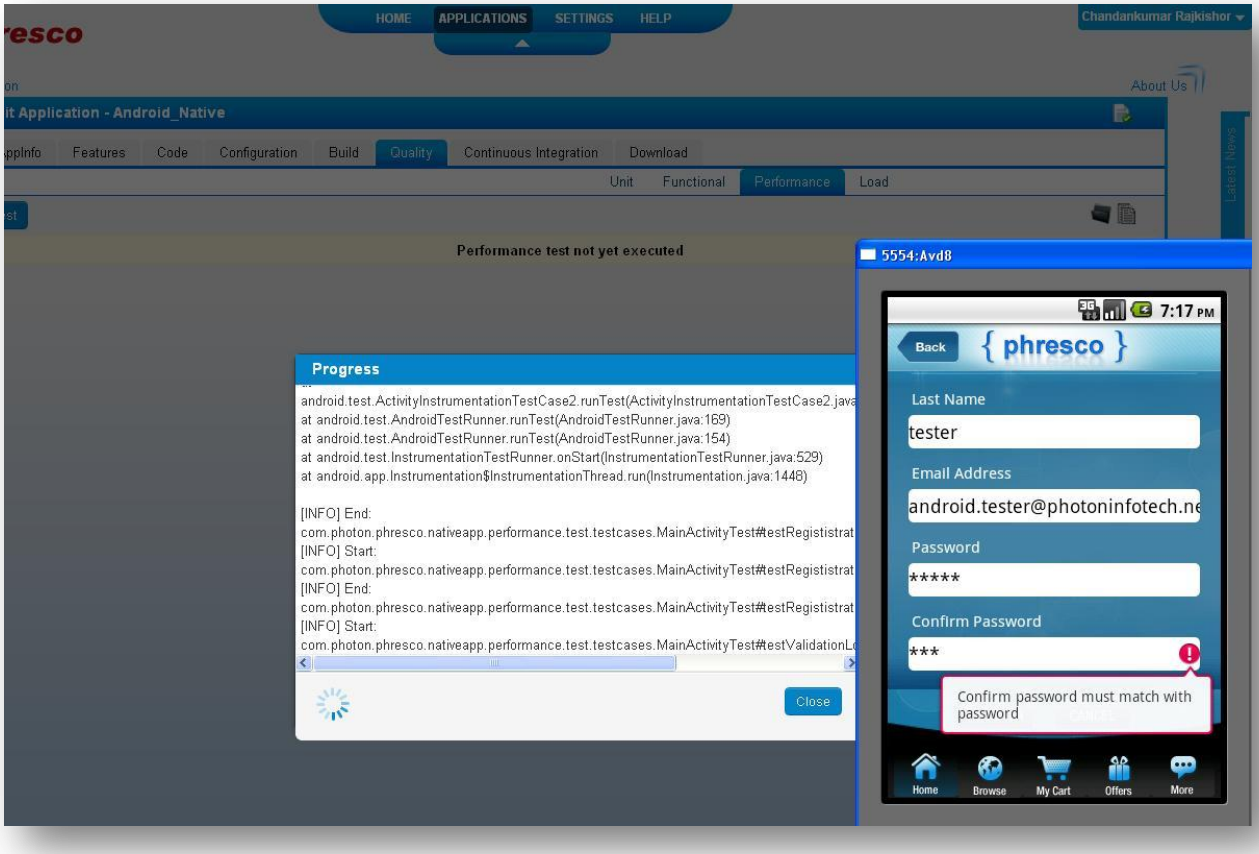


Figure 33: Screenshot for android functional test

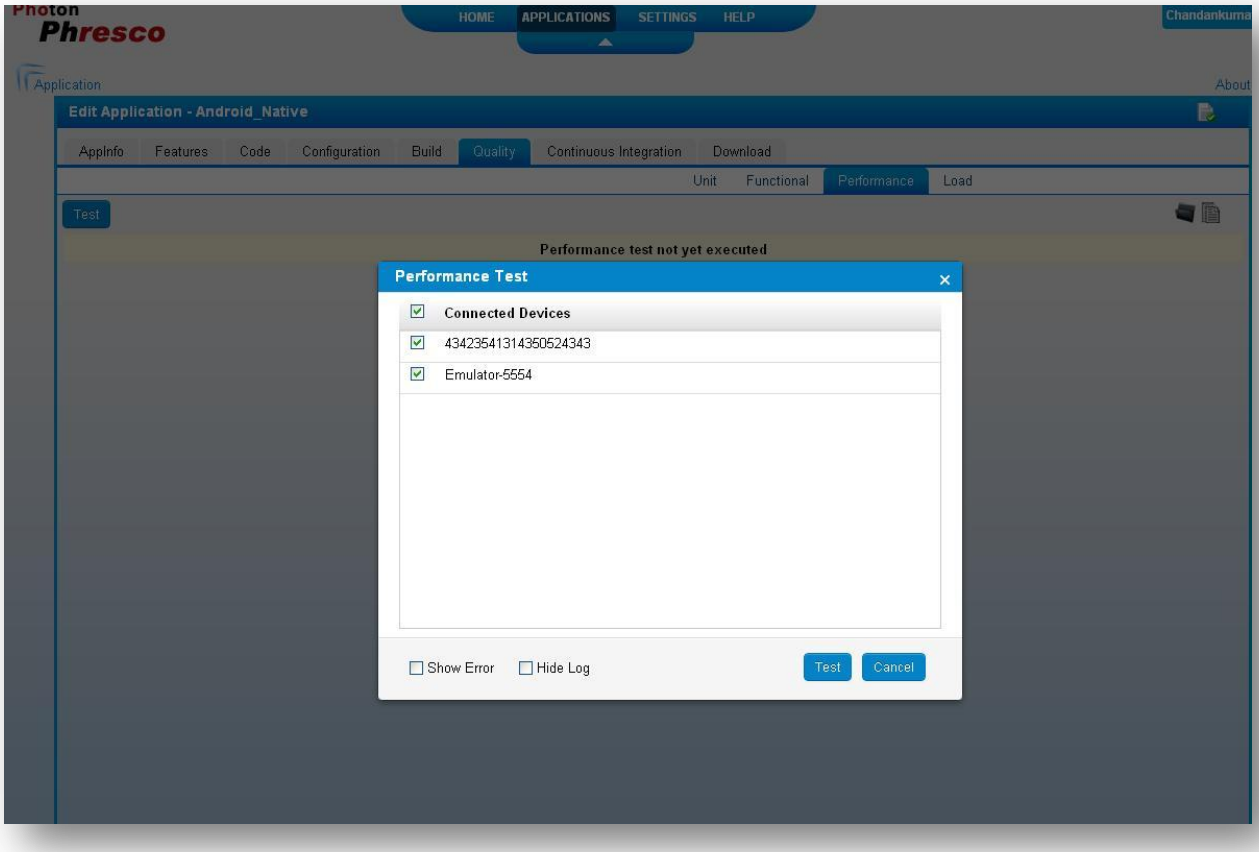


Figure 34: Screenshot

8.5 iPhone application

8.5.1 Unit test case

8.5.1.1 Structure of alltest and test cases

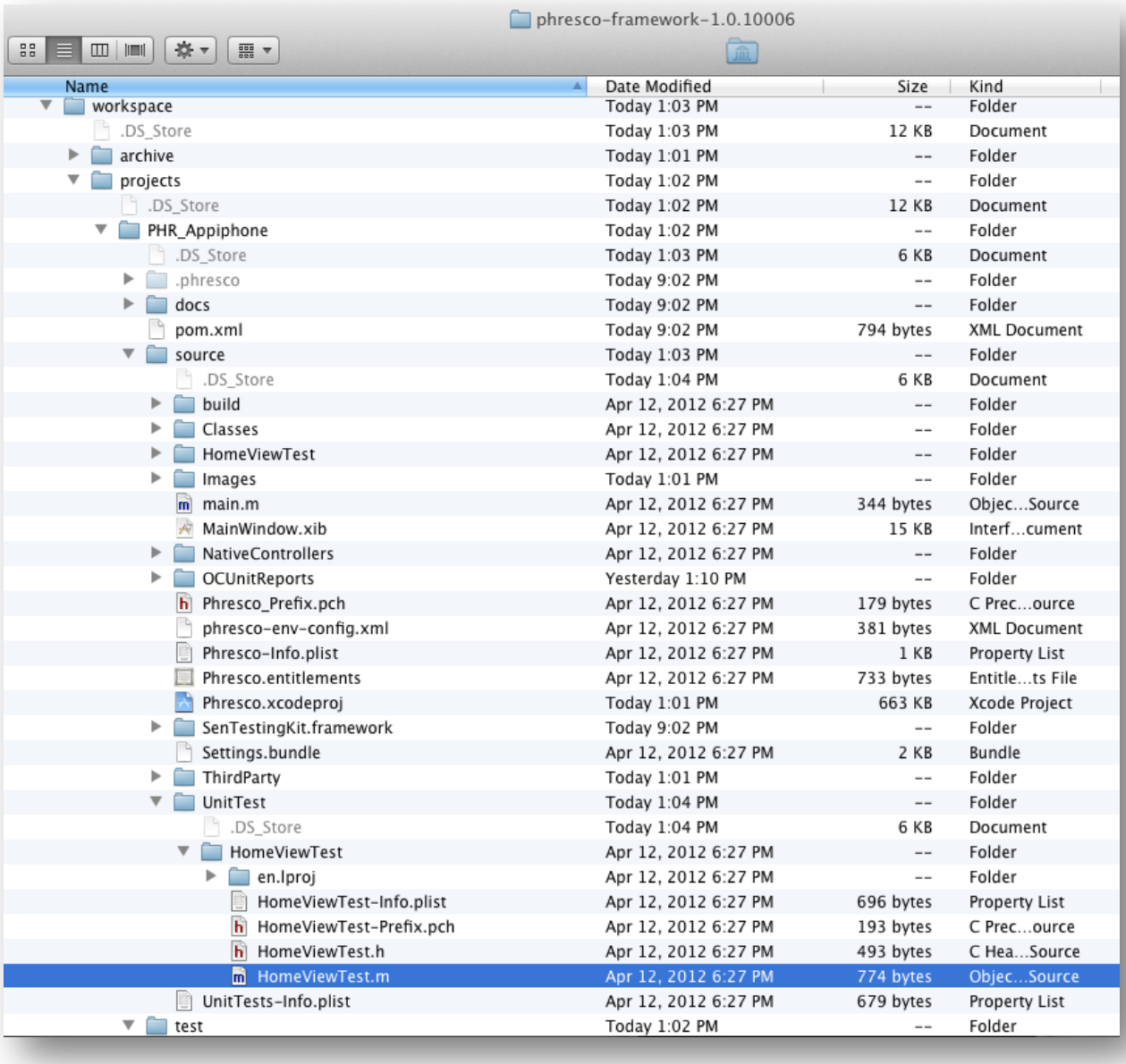


Figure 35: Screenshot for structure of al test and test cases.

a. **Home test view:** this is a test suite in the name of home test view. This is the root folder that contains all the test cases in it
b. **Test cases:** Test cases are present inside the test suites

HomeViewTest for iPhone

This is the root category which holds all the test cases in the name 'Home view Test'. HomeViewTest call any number of test cases depending on the requirements. Developers / testers can start writing new test cases by following the syntax and structure of the phresco framework's out of the box test case structure. Once the test cases are written developers can execute them from Phresco Framework and can see the report. Following is the sample HomeViewTest file which shows up how to add / include the test cases 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:@"o"];

}
-(void) testReview{

    [reviewController tableView:tblView didSelectRowAtIndexPath:o];

}
-(void) testComments{

    [reviewCommentsController goBack:o];

}

-(void) testLogin{

    [loginController registerButtonSelected:o];

}

-(void) testRegister{

    [registerController registerButtonSelected:o];

}
@end
```

Test case example for test register

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.

```
-(void) testRegister{

    [registerController registerButtonSelected:o];
 }
```

8.5.1.2 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'.

 **Note:**

Provided the reference files should be imported and the object should be created.

8.5.1.3 Report generated after testing

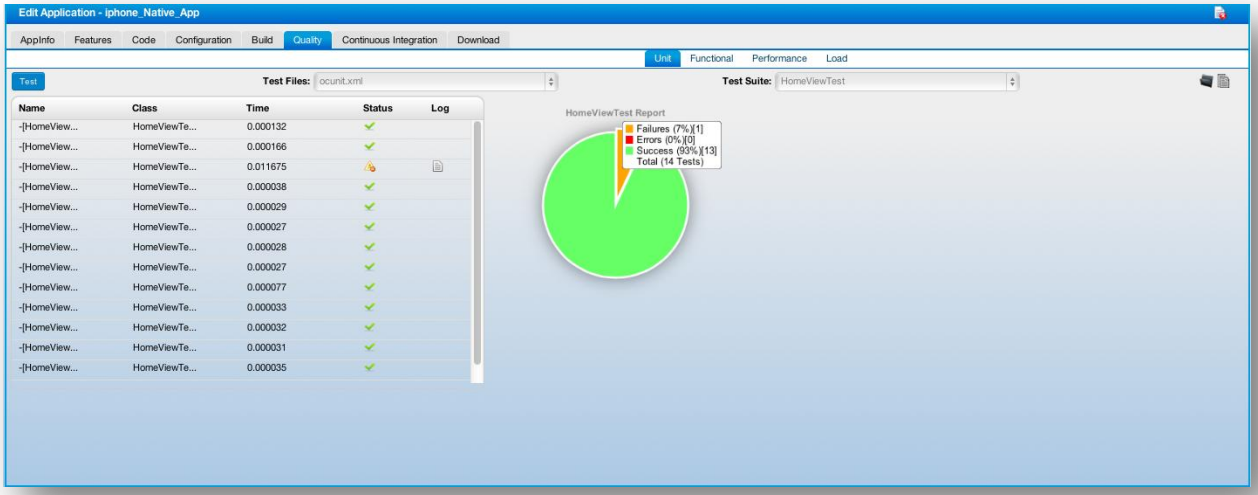


Figure 36: Screenshot for report generated after testing

8.5.2 Functional test case

8.5.2.1 Structure of alltest and test case

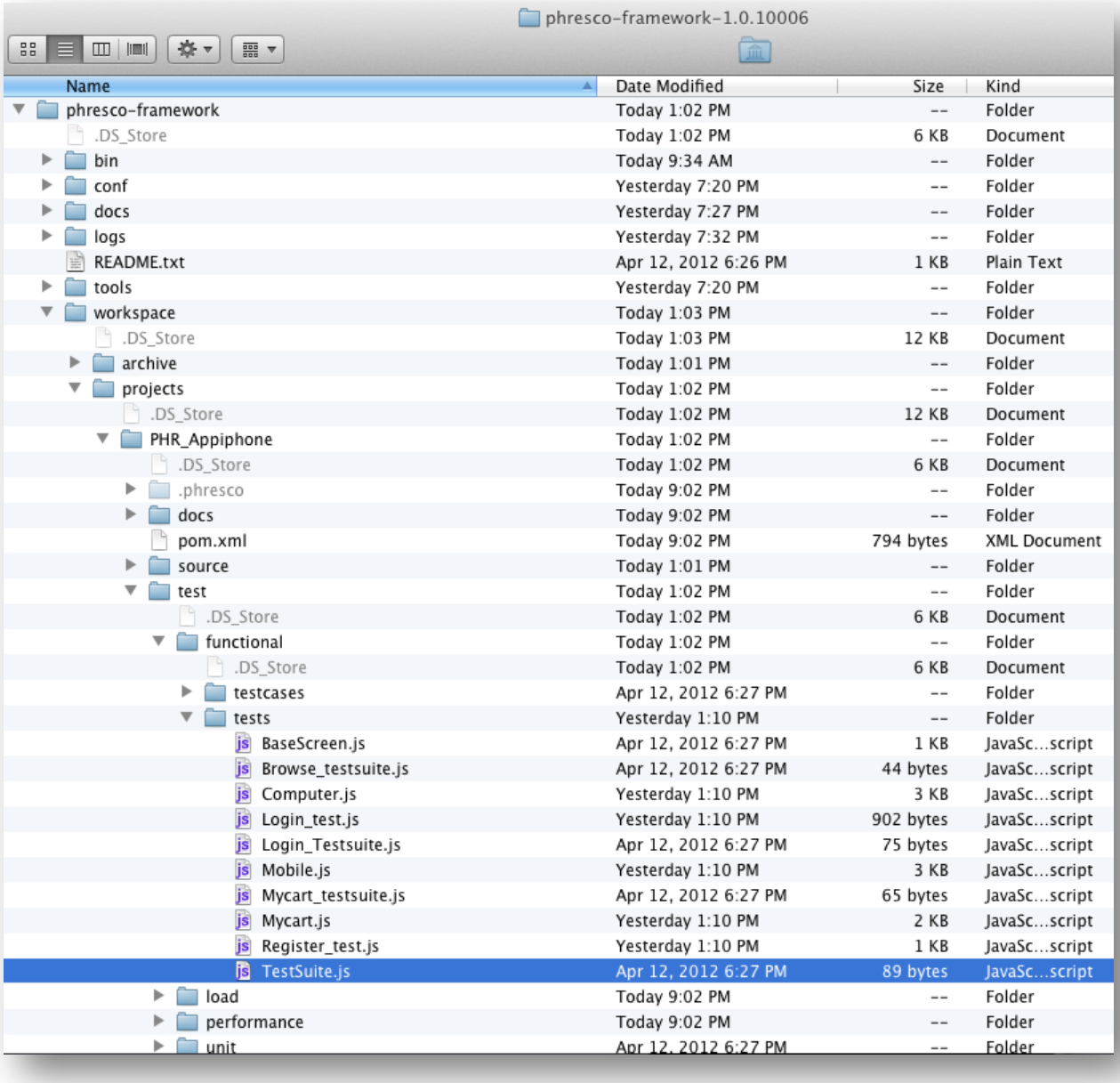


Figure 37: Screenshot for the structure of all test and test cases.

a. **Main Test Suite:** This is the main folder in the name of test suite which calls all the test suites.
b. **Test suite:** Test suites calls the test cases specified within it.
c. **Test cases :** Test cases are the called by the test suites and they are written separately.

8.5.2.2 Existing Test Cases and Suites Out of the Box in Phresco

Main Test Suite for iPhone

Main Testsuite is the root category which holds all the test suites. Functional test runs in the order by which the test suites are created. Main Test Suite can call any number of test suites depending on the requirements. Developers / testers can start writing new test suites and test cases by following the syntax and structure of the phresco framework's out of the box test suites structure. Main Test Suite calls the test suites and test suite calls all the test cases. Once test suite and test cases are written developers can execute them from Phresco Framework and can see the report. Following is the sample MainTestSuite file which shows up how to add / include the test suites inside it. By just using the '#import' syntax all the test suites can be called.

```
#import "Login_Testsuite.js"
#import "Mycart_testsuite.js"
```

Test Suites example for login Test Suite

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 same syntax '#import.'

```
#import "BaseScreen.js"
#import "Register_test.js"
```

Test case example for Register_test

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.

```
#import "BaseScreen.js"
function Register_Test(){

   var reg= "Register_Test";
   var app=UIATarget.localTarget();
```

```
    var target=app.frontMostApp();
    var main=target.mainWindow();
    var button1=main.buttons()[register].tap();
    target.logElementTree();
    main.scrollViews()[0].textFields()[0].setValue(first);
      app.delay(2);
    main.scrollViews()[0].textFields()[1].setValue(last);
      app.delay(2);
    main.scrollViews()[0].textFields()[2].setValue(Email_id);
      app.delay(2);
    main.scrollViews()[0].secureTextFields()[0].setValue(password);
      app.delay(2);
    main.scrollViews()[0].secureTextFields()[1].setValue(password);
      app.delay(2);
    var buttons = main.buttons();
    target.keyboard().buttons()[RETURN].tap();
    app.delay(2);
    buttons[register2].tap();
    app.delay(2);
    main.buttons()[1].tap();
    app.delay(5);
    UIALogger.logPass(reg);

}
UIALogger.logStart("Iphone Test");
Register_Test();
```

8.5.2.3 To add new test suite in Main Test Suite

You can add a new test suite to the Main Test Suite using the following method. Here is an example for creating a new test suite in the name 'Browse_testsuite'

```
#import "Login_Testsuite.js"
#import "Mycart_testsuite.js"
#import "Browse_testsuite.js"
```

8.5.2.4 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 'login_test'

```
#import "BaseScreen.js"
#import "Register_test.js"
#import "Login_test.js"
```

8.5.2.5 Functional testing

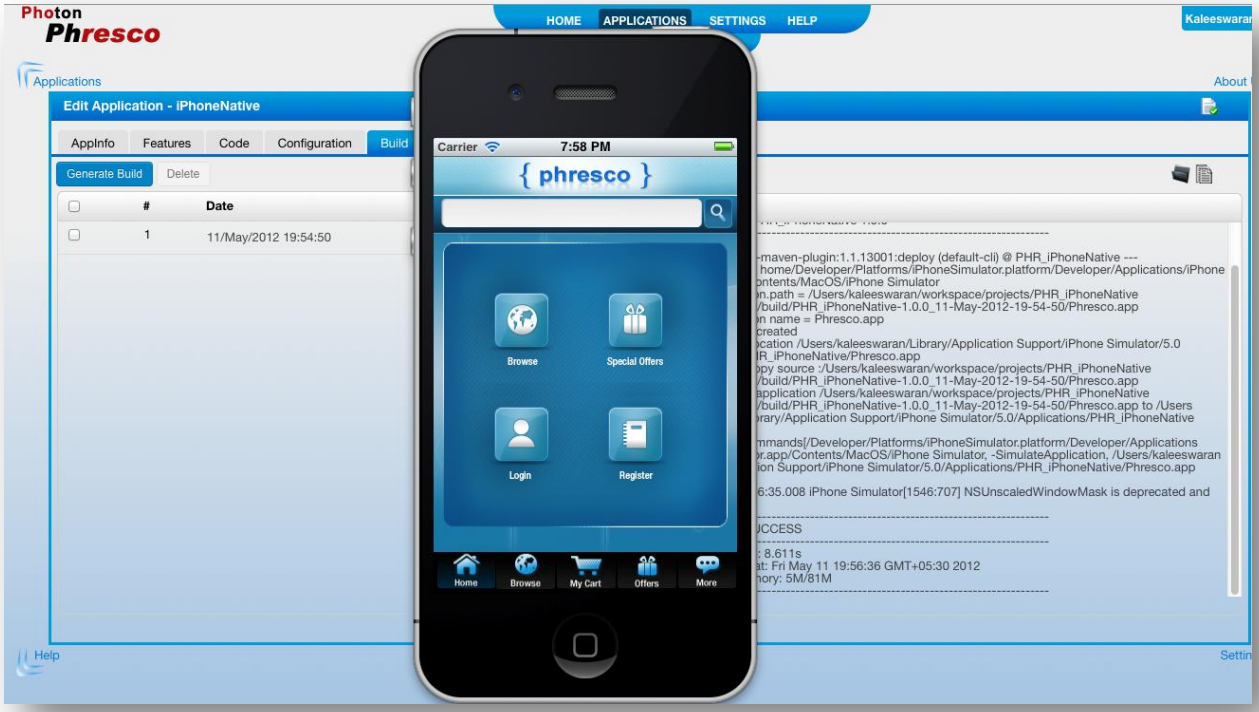


Figure 38: Screenshot for functional test case

Figure 39: Screenshot for functional test case

Figure 40: Screenshot for functional test case

8.6 NODE JS test cases

8.6.1 Unit test case

8.6.1.1 Structure of Test Suites and Test Case

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

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

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

8.6.1.2 Existing Test Cases and Suites Out of the Box in Phresco

AllTest for NodeJS

AllTest is the root category which holds the test suite. Developers / testers can start writing new test suites and test cases by following the syntax and structure of the phresco framework's out of the box test suite structure. AllTest calls the test suites and test suite calls all the test cases written in the test suite. Once test suite and test cases are written developers can execute them from Phresco Framework and can see the report. Following is the sample AllTest for NodeJS

```
        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 same syntax.

Test case example

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.

```
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();
};
```

8.6.1.3 To add new test case in Test suite

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

```
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();
};
```

8.6.1.4 Report generated after testing

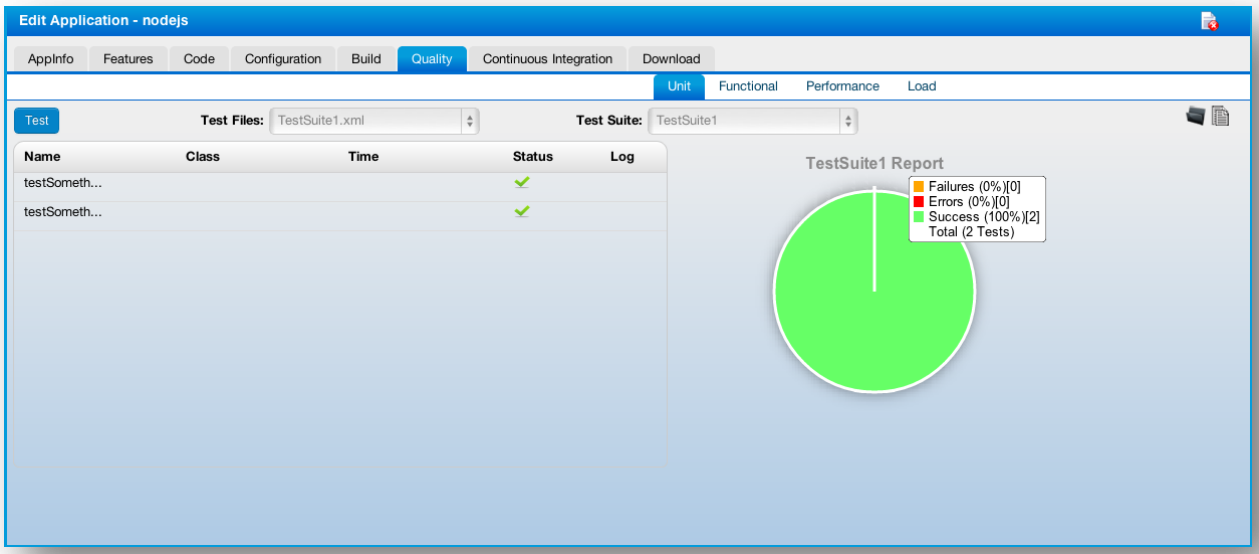


Figure 41: Screenshot for report generated after testing

8.7 Performance Testing

Phresco enables the users to test the performance of their project in an elementary way. It integrates J meter as a tool and as a result generates a report that can be viewed by the Phresco 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 the Phresco 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 the Phresco framework.

Performance testing is testing the performance of a URL 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 URL responds for any given input. This is calculated by using jMeter.

Throughput:

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

Minimum and Maximum Response time:

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

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

Performance testing can be done against server, web service and database.

Performance test against server by using Phresco 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. 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. One or more number of URLs can be tested by clicking the add button while the minus button is used to deselect the URLs.

For Example:

Performance Test against Web Service using Phresco 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. 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. One or more number of URLs can be tested by clicking the add button while the minus button is used to deselect the URLs.

For Example:

Report generated after performance testing :

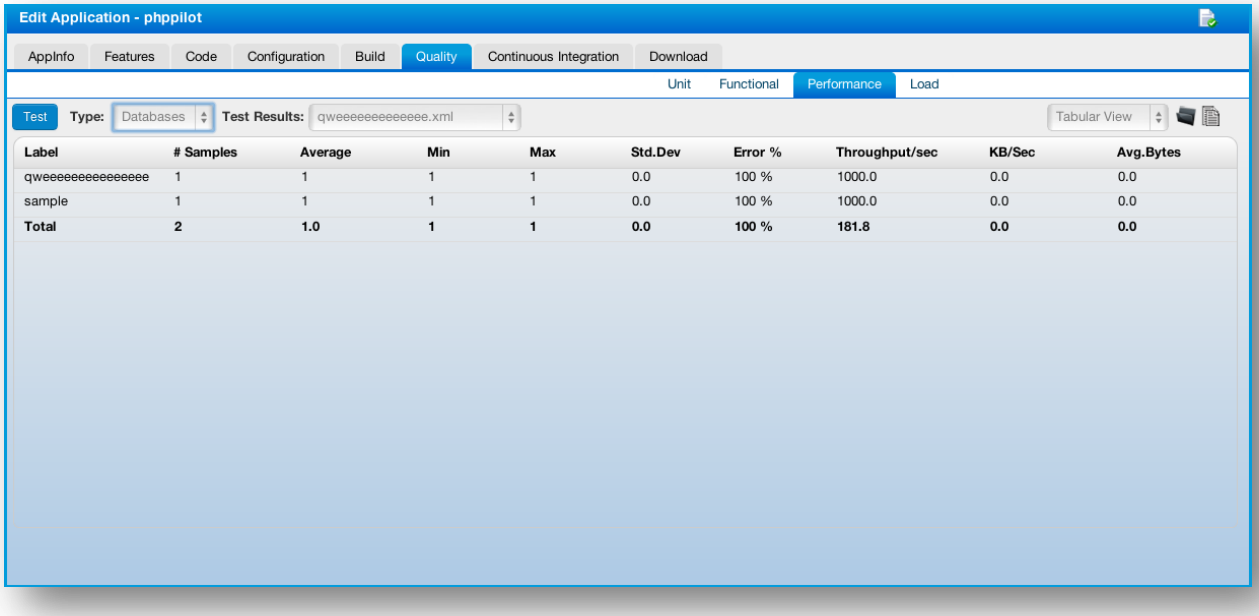


Figure 42: Screenshot for report generated after testing

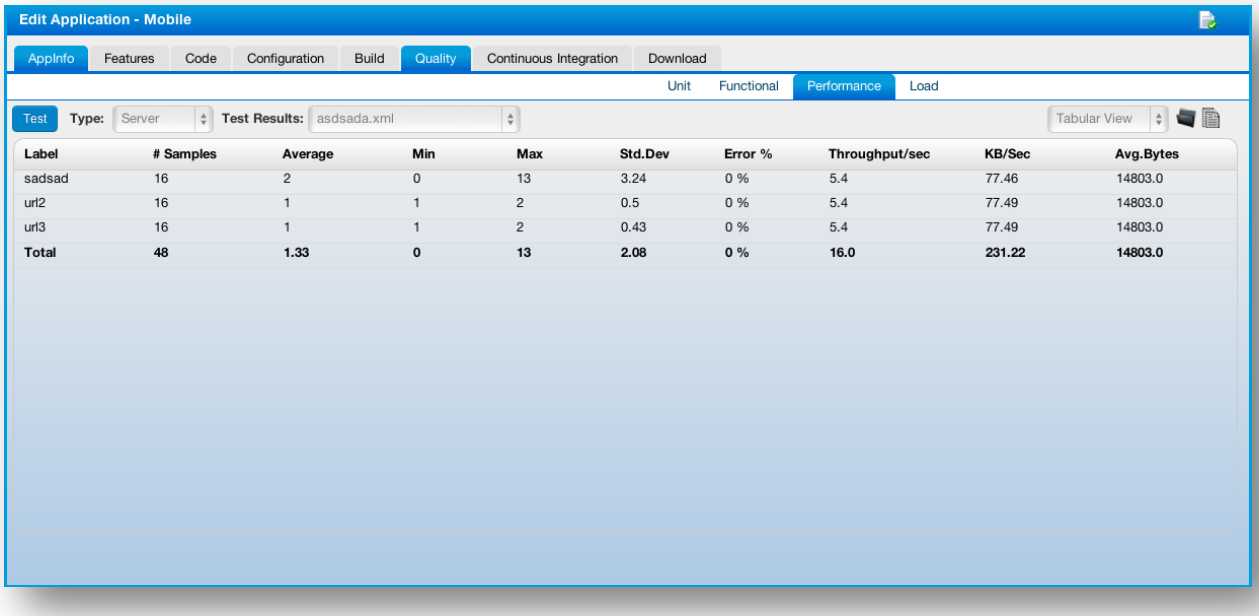


Figure 43: Screenshot for report generated after testing

8.8 Load test

Load testing generally refers to the practice of modeling the expected usage of a project 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, the 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 the project is permitted for real time usage.

Phresco 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.

8.8.1 Load test against server using Phresco 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 Phresco user interface.

8.8.2 Report generated after load testing against server:

8.8.3 Load test against Web service using Phresco 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 Phresco user interface.

8.8.4 Report generated after load testing:

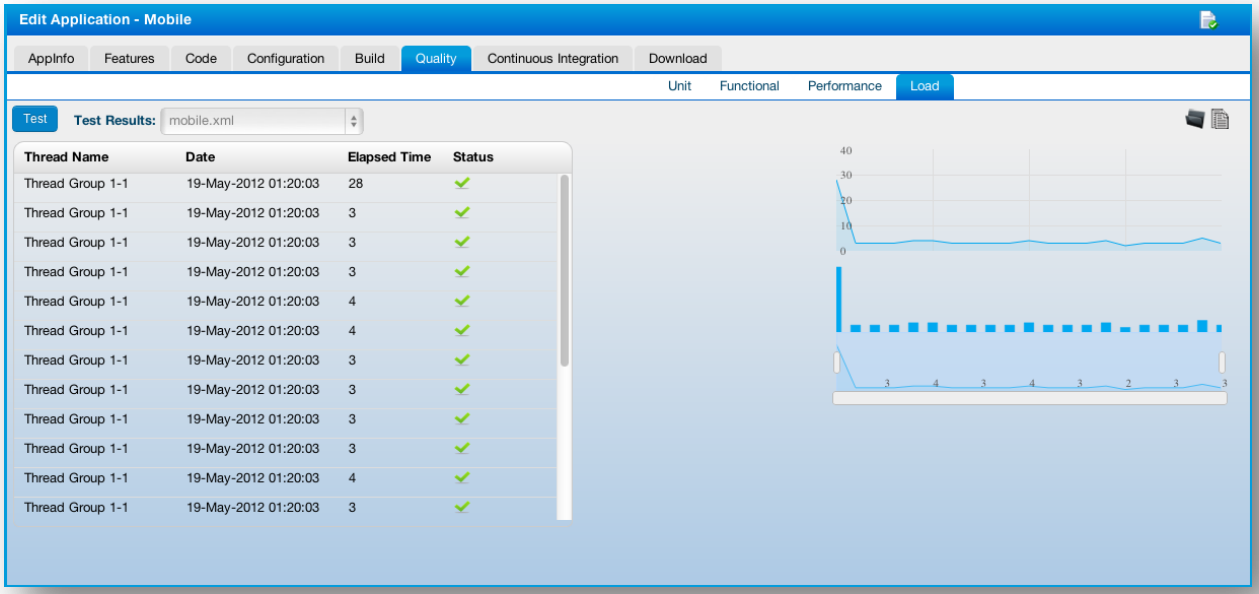


Figure 44: Screenshot for report generated after testing

9 Continuous Integration

Continuous integration is the process where the builds are automatically generated by scheduling it to a particular time.

This deducts the error that occurs in the build and automatically gives the report through provided mail id. Continuous integration uses the tool called Jenkins which is integrated within the framework. This is the server which starts separately in the localhost and the port number is 3579. 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 failure and success through email. If there in no change in the code, build will not start in the server. It can be a challenge to buy unique Halloween costumes if you have a whole family to buy for, because rarely do you ever find costumes for the whole family under one roof. Running from store to store with bored and cranky kids can drive anyone nuts.

The internet has evolved and this year more than ever, people will shop online for their Halloween costumes, candies, costume accessories and decorations. Online stores thus are stocking up for the anticipated rush in the moths before Halloween.

Performance of continuous integration

The continuous integration is the best feature provided by Phresco which integrates the Jenkins tools and does the automatic build generation. Setup function present in the Phresco user interface loads the Jenkins and Jenkins can be started using start function. 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 done from setup and start.

screenshot can be inserted here

Field in screenshot

name: any name can be filled in for identifying the continuous integration file
svn url: the url from which the projects are checked out can be copied in this field.
username: system username and password should be entered in
email: the email id and password to which the email has to be sent can be typed in here
checkbox for failure and success should be clicked regarding the requirement when the mail is to be received.
the date and time intervals can be selected which builds the project automatically and sends the report to the mail in the

this process reduces the integration problem and it gives the development team to produce a full quality project.

iPhone prerequisites:

 **Note:**

developers or testers should have registered in develop.apple.com or should have an id for downloading the following softwares.

For developers

1. Mac machine is required
2. Mac OS 10.6 & above
3. Xcode tool 4.2 & above
4. IOS SDK 4.0 & above
5. iTunes !0.6 & above

For testers

1. Mac machine is required
2. Mac OS 10.6 & above
3. Xcode tool 4.2 & above
4. IOS SDK 4.0 & above
5. itunes !0.6 & above

2. All the requisites for developers are needed
2. instrumentation tool for automation test is needed

Deploying Devices

1. iPhone 4, 4s, iPad 2

note: Development certificate is required for deploying the project in the device.

Supported versions for Android:

To build a project:

Phresco supports versions like 2.2, 2.3.3 and 4.03 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 the project can be deployed in the versions of 7 and above 7.

- These are the different versions and the API levels:

Platform	Codename	API Level	Distribution

Android 1.5	Cupcake	3	0.3%
Android 1.6	Donut	4	0.7%
Android 2.1	Éclair	7	5.5%
Android 2.2	Froyo	8	20.9%
Android 2.3-Android 2.3.2	Gingerbread	9	0.5%
Android 2.3.3-Android2.3.7		10	63.9%
Android 3.0	Honeycomb	11	0.1%
Android 3.1		12	1.0%
Android 3.2		13	2.2%
Android 4.0-Android 4.0.2	Ice Cream Sandwich	14	0.5%
Android 4.0.3-Android 4.0.4		15	4.4%

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.