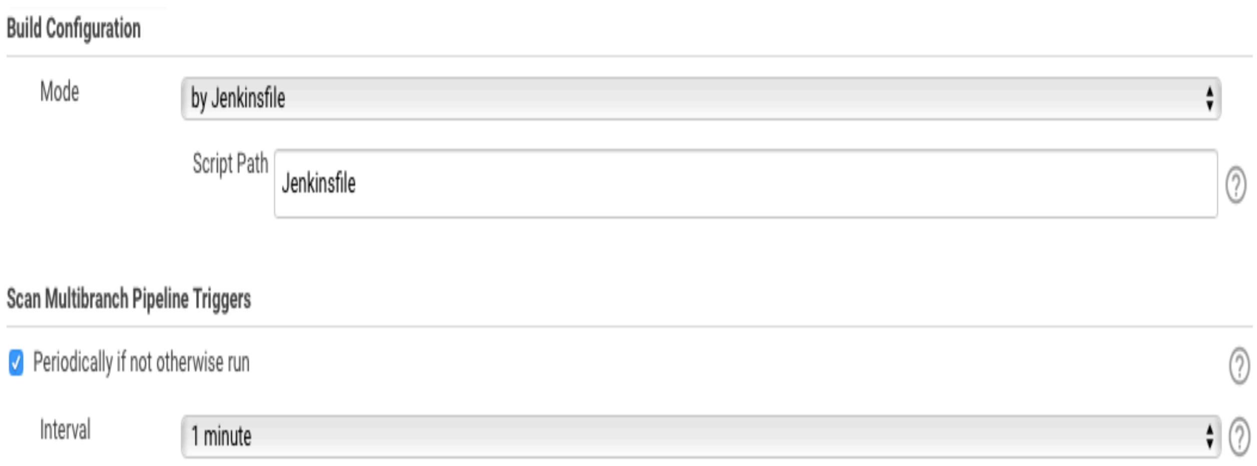
Mobile Apps CI Structure

Pipeline as a Code

This Structure depicts the details of Build and Release activities (Continuous Integration) for GHS Xamarin Apps.

Each developer takes out a feature branch from the DEVELOP branch and once locally tested, pushes the code to the respective feature branch.

With the help of "**JenkinsFile**" and "**Multibranch Pipeline Plugin**" we have configured Jenkins Jobs which runs every minute and scans the Mobile Apps Github repo to check for any new changes.



Build Configuration

Mode: by Jenkinsfile

Script Path: Jenkinsfile

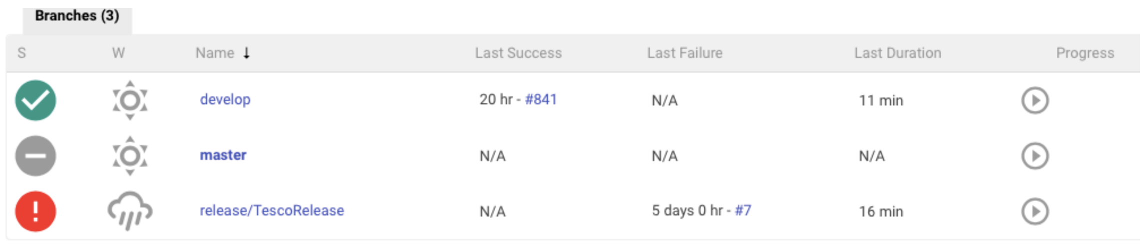
Scan Multibranch Pipeline Triggers

☑ Periodically if not otherwise run

Interval: 1 minute

Note : JenkinsFile is also configured to run on particular environment like OPS , Store5, Store 6 etc.

After the Scan, Pipeline Plugin triggers the Jenkins Job for that Particular branch and starts the build.

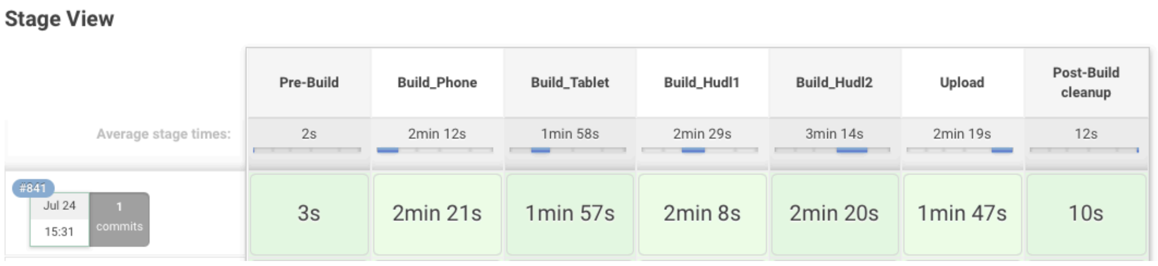


Branches (3)

S	W	Name ↓	Last Success	Last Failure	Last Duration	Progress
✓	☀	develop	20 hr - #841	N/A	11 min	▶
⊖	☀	master	N/A	N/A	N/A	▶
!	🌧	release/TescoRelease	N/A	5 days 0 hr - #7	16 min	▶

Sample JenkinsFile:

```
node('hsc-build-02') {
            checkout scm
            def workspace = pwd()
            def build_number = "${env.BUILD_NUMBER}"
            def xcode_path = '/Applications/"Xamarin Studio.app"'

stage('Pre-Build'){
                        // Creating output dir; Installing NuGet libraries
                        sh '''
                                    chmod 755 '''+workspace+'''
                                    mkdir -p '''+workspace+'''/buildArtifacts
                                    cd '''+workspace+'''
                            '''

stage('Build_Phone') {
            sh '''
            xbuild  '''+workspace+'''  Grocery.Droid.csproj /p:Configuration=ReleaseWIP
             /t:SignAndroidPackage   /p:OutputPath=bin/Release
            ENT_BUILD=`find  '''+workspace+''' Release/ -name *-Signed.apk -exec basename {} \\; `
            mv  '''+workspace+'''/buildArtifacts/GroceryDroidRelease-'''+build_number+'''.apk
                 '''
            }

stage('Build_Tablet'){
            sh '''
            xbuild '''+workspace+'''  Grocery.Droid.csproj /p:Configuration=ReleaseTabletWIP
           /t:SignAndroidPackage    /p:OutputPath=bin/ReleaseTablet
            ENT_BUILD=`find  '''+workspace+'''ReleaseTablet/ -name *-Signed.apk -exec basename {}\\;
            mv '''+workspace+''' /buildArtifacts/GroceryDroidReleaseTablet-'''+build_number+'''.apk
                '''
            }
```



Stage View

	Pre-Build	Build_Phone	Build_Tablet	Build_Hudl1	Build_Hudl2	Upload	Post-Build cleanup
Average stage times:	2s	2min 12s	1min 58s	2min 29s	3min 14s	2min 19s	12s
#841 Jul 24 15:31 1 commits	3s	2min 21s	1min 57s	2min 8s	2min 20s	1min 47s	10s

We are using Xamarin (Cross Platform Tool for both iOS and Android Projects) for our Build activities. Once the build Completes, Jenkins job uploads the generated *.ipa or *.apk to Hockey app which facilitates the Testers to get the app on the testing device and start the testing of new features.

QA Automation

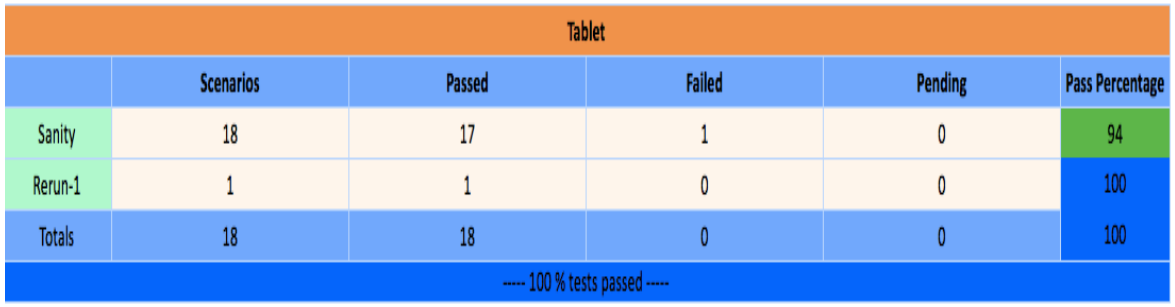
This shows the Continuous Integration for Testing the Apps.

We have configured different Jenkins Jobs which is scheduled to trigger every three hours once and takes the particular (*.ipa for iOS and *.apk for Android) latest app from the hockey and starts the Automation.

Details of Automation Jobs in Jenkins:

1. **GHS_Android_Sanity_Phone/Tablet**
   This job is configured to run on Master branch build and completes the Basic Sanity for the App. As mentioned, it runs every three hours once and generates the Sanity report which is then sent out in a mail to the entire team.
2. **GHS_Android_Priority_Phone/Tablet**
   This Job is executed during the Christmas Peak time and it is configured to run on any branch specified by the Testing team. This Job is executed every hour during the Christmas season.
3. **GHS_Android_Regression_Phone/Tablet**
   This Job is configured to run on Develop branch every night by taking a new build and uploading it to hockey. The Regression job consists of more number of test cases and more scenarios which does a complete test of the App.

Example of the Sanity Mailer Report:

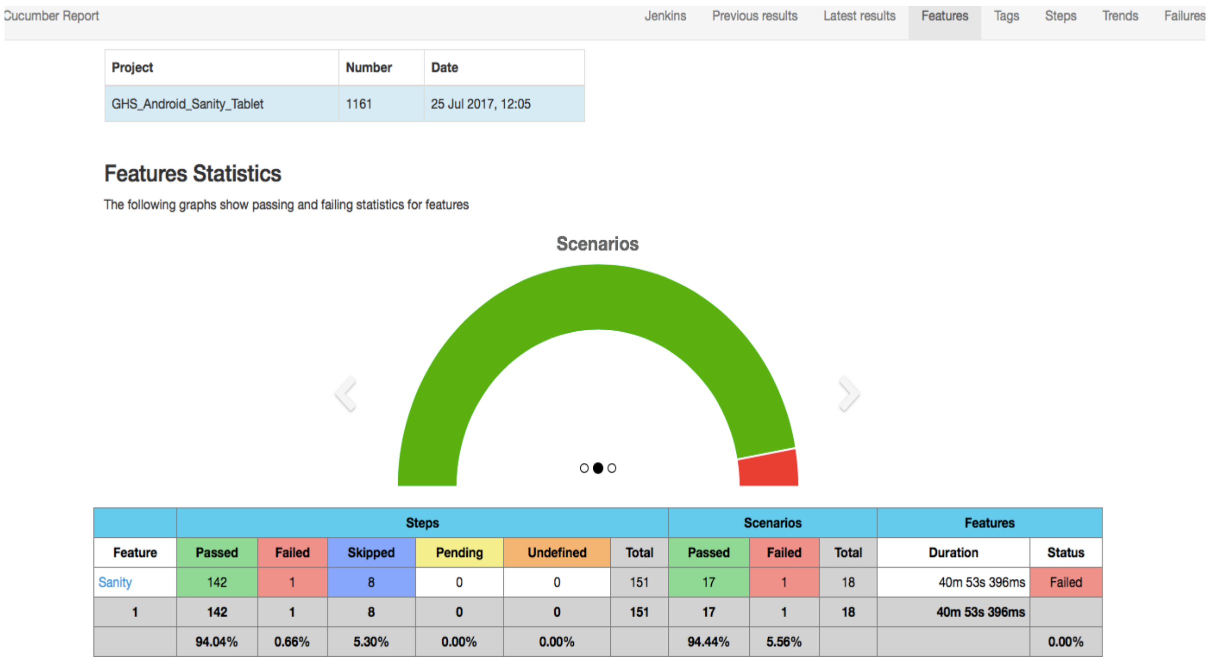


Tablet

	Scenarios	Passed	Failed	Pending	Pass Percentage
Sanity	18	17	1	0	94
Rerun-1	1	1	0	0	100
Totals	18	18	0	0	100

----- 100 % tests passed -----

We have configured the Jenkins Job to publish the Cucumber report which is generated with the testing which shows the complete analysis.


Cucumber Report

Project	Number	Date
GHS_Android_Sanity_Tablet	1161	25 Jul 2017, 12:05

Features Statistics

The following graphs show passing and failing statistics for features

	Steps						Scenarios			Features	
Feature	Passed	Failed	Skipped	Pending	Undefined	Total	Passed	Failed	Total	Duration	Status
Sanity	142	1	8	0	0	151	17	1	18	40m 53s 396ms	Failed
1	142	1	8	0	0	151	17	1	18	40m 53s 396ms	
	94.04%	0.66%	5.30%	0.00%	0.00%		94.44%	5.56%			0.00%

SonarQube Code Analysis

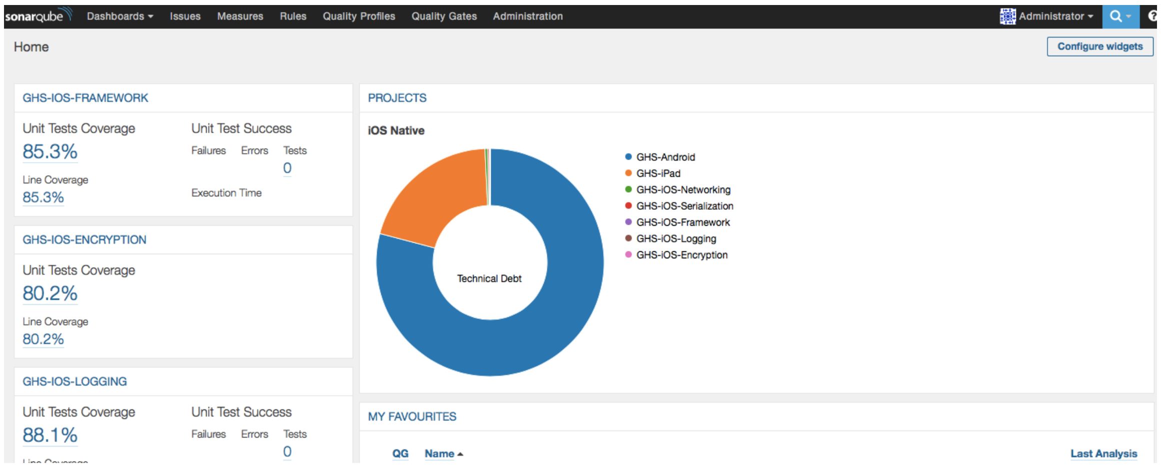
We have configured SonarQube which is a open source platform for continuous inspection of Code Quality. We have created separate Jenkins job for each Android(Phone & tablet) and iOS(iPhone & iPad).

In each repository, we have sonar.properties file which is having the details of the solution file , rules to be executed etc.

Sample Sonar.Properties File:

```
# project configuration
sonar.projectKey=iOS:Native
sonar.projectVersion=1
sonar.projectName=GHS-iOS-Framework

sonar.sources=TescoiOSFramework
sonar.language=swift

sonar.fxcop.mode=
sonar.gendarme.mode=
sonar.gallio.mode=
sonar.ndeps.mode=
sonar.stylecop.mode=

sonar.sourceEncoding=UTF-8

#Xcode project configuration
sonar.swift.workspace=TescoiOSFramework.xcworkspace
sonar.swift.projects=TescoiOSFramework.xcodeproj
sonar.swift.appscheme=TescoiOSFramework
sonar.swift.testscheme=TescoiOSFrameworkTests

sonar.junit.reportsPath=fastlane/test_output/
sonar.swift.coverage.reportPattern=fastlane/test_output/cobertura.xml
sonar.swift.swiftlint.report=fastlane/test_output/*swiftlint.txt
```

We have configured different plugins to support our projects, Swift plugin for iOS and java,CSharp plugins for Android which consists of pre-defined set of rules which executes on our code and provides us the Static as well as Dynamic code analysis results.



Home

GHS-IOS-FRAMEWORK

Unit Tests Coverage 85.3%
Line Coverage 85.3%
Unit Test Success
Failures Errors Tests 0
Execution Time

GHS-IOS-ENCRYPTION

Unit Tests Coverage 80.2%
Line Coverage 80.2%

GHS-IOS-LOGGING

Unit Tests Coverage 88.1%
Unit Test Success
Failures Errors Tests 0

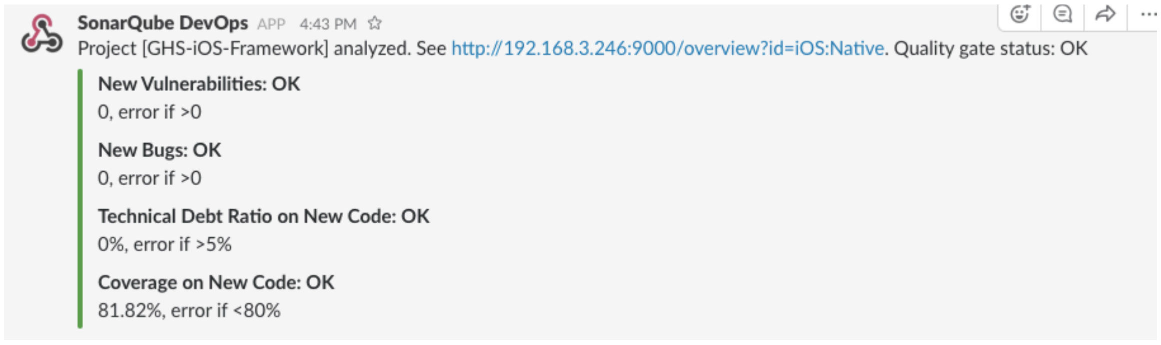
PROJECTS

iOS Native

MY FAVOURITES

QG Name Last Analysis

We have even integrated SonarQube code Analysis with our Slack Channel and provides us the details of the latest sonar execution.



SonarQube DevOps APP 4:43 PM
Project [GHS-iOS-Framework] analyzed. See http://192.168.3.246:9000/overview?id=iOS:Native. Quality gate status: OK

New Vulnerabilities: OK
0, error if >0

New Bugs: OK
0, error if >0

Technical Debt Ratio on New Code: OK
0%, error if >5%

Coverage on New Code: OK
81.82%, error if <80%

Jenkins URL :

http://192.168.2.201:8086

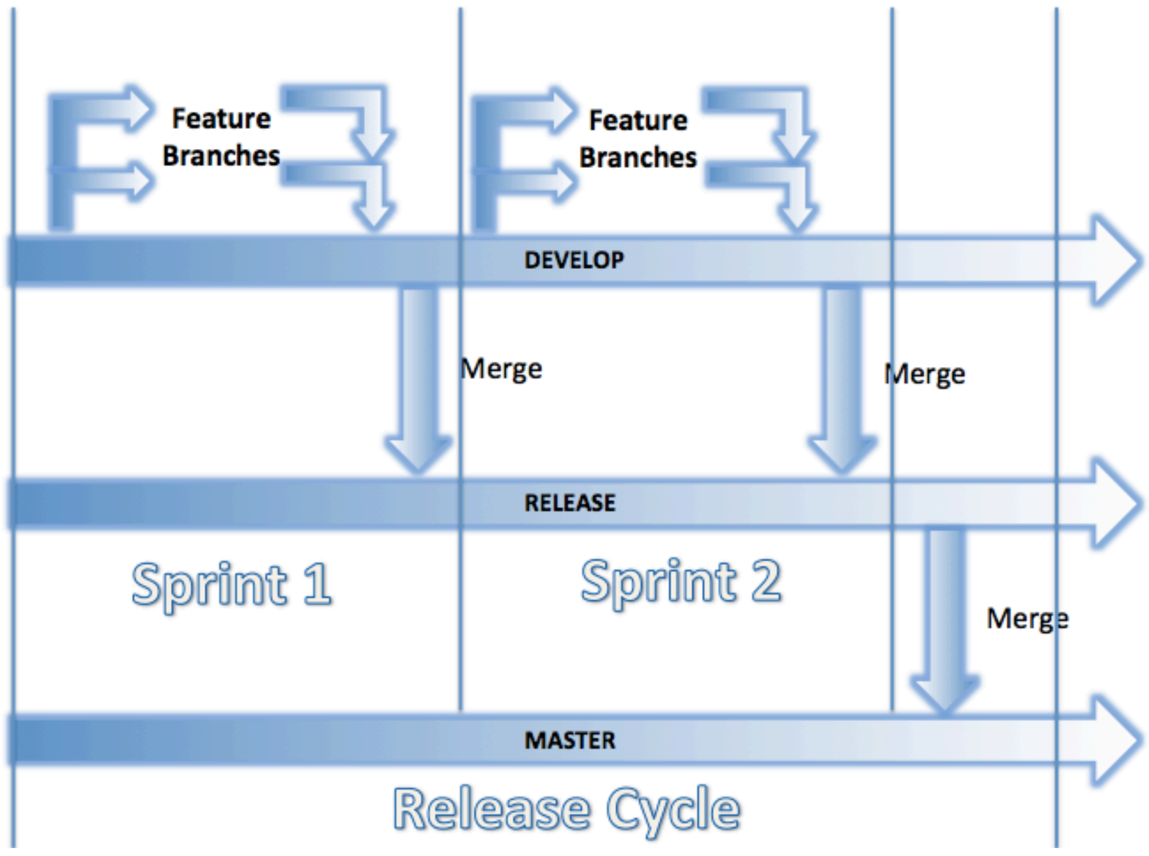
SonarQube URL :

http://192.168.3.246:9000

GitHub branch Strategy

This shows the branching Structure which we have implemented across Mobile Apps projects.

- **DEVELOP** – All the development work during the sprint goes into this branch. Each developer takes out a feature branch and once locally tested, merge back to develop.
- **RELEASE** – Develop branch is merged with this branch after all the development and testing. Regression test for that sprint is triggered on this branch at the end of the sprint.
- **MASTER** – After all the tests on Release branch and after the sign off from the testers, Release branch is merged into Master. Build generated from this branch is uploaded to iTunes for beta testing(internal). After a successful beta testing, this build will be sent to appstore for review.


Feature Branches
Feature Branches
DEVELOP
Merge
Merge
RELEASE
Sprint 1
Sprint 2
Merge
MASTER
Release Cycle