THANOS

# QUALITY DASHBOARD

Mukesh Rajput

THANOS

## We will discuss:

- ❑ What is QA Dashboard?
- ❑ Detailed walkthrough of features
- ❑ How it works in backend?
- ❑ How to contribute?

## What is QA Dashboard?

THANOS

## Background

- I was having a team of 35+ QA people distributed in **20 different teams**, so it was pretty hard to track on how these teams are doing
- As a manager, it was really painful to find out all the relevant data for these teams, given data set is spread across **multiple tools**
- Like - Multiple automation frameworks - Api, Web & Mobile automation, different Jira boards for each team
- Multiple testrail projects & suites - hard to get data about how each team is progressing in automation coverage

## Background

- For tracking the team progress, i was using excel sheets, but it was not feasible option, as i was suppose to keep maintaining them on regular basis
- So, Last year, in my free time when Covid lockdown happened, I thought to create a dashboard which can collate data spread in multiple tools/frameworks at one place
- This gave us the much needed single dashboard - QA Dashboard

THANOS

## So, What is QA Dashboard?

- QA Dashboard is a tool which enables us to track QA related metrics from multiple sources at single place, not only at team level but also at Pod and Entity level.

- Right now this can collate data from these 4 sources:
  - **Test Coverage** - Different Testrail projects & suites
  - **Automation Stability** - Multiple Automation Frameworks - api, web, mobile
  - **Bug Metrics data** - From multiple Jira boards across Entity
  - **Code Coverage** - Unit tests coverage data from Dev’s repos across Entity

THANOS

## Tools/Languages Used

- HTML
- CSS
- JavaScript & JQuery
- PHP
- MySQL
- FusionCharts

## Detailed walkthrough of Features

THANOS

# Automation Results

This page fetch data from different automation frameworks (like web, api, mobile) and present the aggregated summary for each of your team

## Entity level Data

- Average **'regression' & 'sanity' percentage** for all projects in last N days [on Staging]
- Average **'prodSanity'** percentage for all projects in last N days [on **Production**]
- Average 'regression' & 'sanity' **execution time** for all projects in last N days [on **Staging**]
- Average 'prodSanity' **execution time** for all projects **in last N days** [on Production]

## Team specific Data

- One click access to results of **last 30 'regression' and 'sanity' builds** of your project
- Trend Chart on how **Pass Percentage** is changing **daily/weekly/monthly**
- Trend Chart on how **Execution Time** is changing **daily/weekly/monthly**
- Trend Chart on how **Count of automation cases** is changing **daily/weekly/monthly**

THANOS

## Testrail Numbers

This page fetch data from multiple testrail projects as well as from suites via testrail API and present the aggregated summary for each of your team

- **Entity level Data**
  - P0, P1 & Full Automation Coverage Percentage of **whole Entity**
  - **Count of number of testcases automated** in last N days [Each Project]
  - **Change in P0 & P1** Automation Percentage in last N days [Each Project]
  - Overall Testcase Distribution in Testrail [Each Project]
- **Team specific Data**
  - P0, P1 & Full Automation Coverage Percentage of **your Project**
  - Pie Chart to present overall testcase breakdown for your Project
  - Column chart to show Already Automated vs Total Automation cases comparison for your Project
  - Trend Chart on how **count of testcases** is changing **daily/weekly/monthly**

THANOS

## Bug Metrics

This page fetch data from multiple JIRA projects using JIRA filters and present the aggregated summary for each of your team

- **Entity level Data**
  - Total Tickets Tested, Total Bugs as well as Only Production bugs found in last N days for **whole Entity**
  - Count of Total Bugs found in Last N days in **each Team within your Entity**
  - **Teamwise comparison chart** on Bugs found per 100 Jira tickets tested **(Bug percentage)**
  - Similar chart for **Only Production** Issues found so far

- **Team specific Data**
  - Total Tickets Tested, Total Bugs as well as Only Production bugs found for **your Project**
  - **Priority wise** Bugs Breakdown in last N days for your Project
  - Trend Chart on how **Number of Bug found** is changing **daily/weekly/monthly**
  - Trend Chart on how **Bug Percentage** is changing **daily/weekly/monthly**

THANOS

## Unit Test Coverage

This page fetch unit test coverage data of Developer’s repo and present the aggregated summary for each of your team

- **Entity level Data**
  - Unit tests coverage percentage for **whole Entity**
  - **Change in Unit Tests Coverage percentage** in last N days for each project
  - **Current Unit Tests Coverage percentage** for each project in whole Entity
- **Team specific Data**
  - Unit tests coverage percentage for **your Project**
  - Code coverage bar graph of last N days for your project.
  - One click access to module wise coverage reports of your project.
  - Trend Chart on how **coverage is changing for Lines, Statements, Branches & Functions** in last N days for your project.

THANOS

## How it works in Backend?

THANOS

## Behind The Scenes

- **MySQL database**
  - Each Entity have 4 separate tables like this:
  - <entityName>_results
  - <entityName>_testrail
  - <entityName>_jira
  - <entityName>_bugs
  - <entityName>_units

- **For Automation Results**
  - Automation Results are directly inserted into the database (if have Midtrans vpn access)
  - Else, Automation Results use GCP bucket flow to send the data into results table
  - All the data whether its for Api, Web or Mobile is stored in same format and in same table
  - From that **<entityName>_table** various queries are executed via PHP to present the data in the dashboard

THANOS

## Behind The Scenes

- **For Testrail Numbers**
  - For Each Entity we are fetching data directly from testrail using testrail API
  - For fetching the data we need to put Testrail Project ID and suite ID in a predefined config file
  - After that we store this data in **<entityName>_testrail table** which is separate for every Entity

- **For Bug Metrics**
  - For Each Entity we are fetching data directly from JIRA API by passing the required filter name
  - This means we need to create a filter in Jira for each Entity for maintaining the linking between individual projects and Entity
  - Then for fetching the data we need to put Jira Project KEY in the in the predefined config file
  - After that we store this data in **<entityName>_jira & _bugs tables**, these are separate for every Entity
  - From that table various queries are executed via PHP to present the data in the dashboard

# Behind The Scenes

- **For Unit Tests**
  - Developer’s repo should have tool like Jacoco to calculate unit test coverage.
  - Unit test coverage data should be sent to GCP bucket in particular csv format.
  - Data from GCP is extracted and stored in respective Entity table: **<entityName>_units.**
  - From that **<entityName>_units table** various queries are executed via PHP to present the data in the dashboard

THANOS

## How to Contribute?

## Want to contribute further?

- As source code is publicly available, so refer readme for more technical details
- Anyone who wants to contribute can just clone and raise the MR
- Everything is present in same repo be it Backend or Frontend
- Code related to Frontend is placed in "**Website**" folder
- And Code related to Backend is in "**src**" folder
- Needless to say, MR will follow the review process

# Thank you!

THANOS