**OpenSource ADOPT**

**How Jira APIs are used for reporting in ADOPT WebApp?**

In the “**opensourceadopt**” code base, all the code related to planning reports is written in the class “**com.tech.adopt.service.ProjectReportsServiceImpl**”, which implements the “**ProjectReportsService**” interface.

Following are the list of REST API end points used for reporting. A full detail about each URL is explained in the immediate section.

**Sprint Report (Sprint name, start Date, End Date, Closed Story points, and total story points)**

<http://172.19.82.13:8050/rest/greenhopper/1.0/rapid/charts/sprintreport?rapidViewId=$RAPIDVIEWID&sprintId=$SPINTID>

**Velocity (Sprint name, completed story points, planned story points (commitment))**

<http://172.19.82.13:8050/rest/greenhopper/1.0/rapid/charts/sprintreport?rapidViewId=$RAPIDVIEWID&sprintId=$SPINTID>

**Test Trend (Test Date, Passed Count, Failed Count)**

**NOT DONE**

**Sprint Details (Sprint Name, Start Date, and End date)**

<http://172.19.82.13:8050/rest/agile/1.0/board/$BOARDID/sprint>

**Board ID (Boards, Id)**

<http://172.19.82.13:8050/rest/greenhopper/1.0/rapidviews/list>

**Sprint Burn down (Sum of estimates in “Open “state in Sprint for a given date)**

<http://172.19.82.13:8050/rest/greenhopper/1.0/rapid/charts/scopechangeburndownchart?rapidViewId=$RAPIDVIEWID&sprintId=2$SPRINTID>

**Release Burn down (Sprint name, work added, work completed, and work remaining)**

http://172.19.82.13:8050/rest/greenhopper/1.0/rapid/charts/releaseburndownchart?rapidViewId=$RAPIDVIEWID&versionId=$RELEASEID

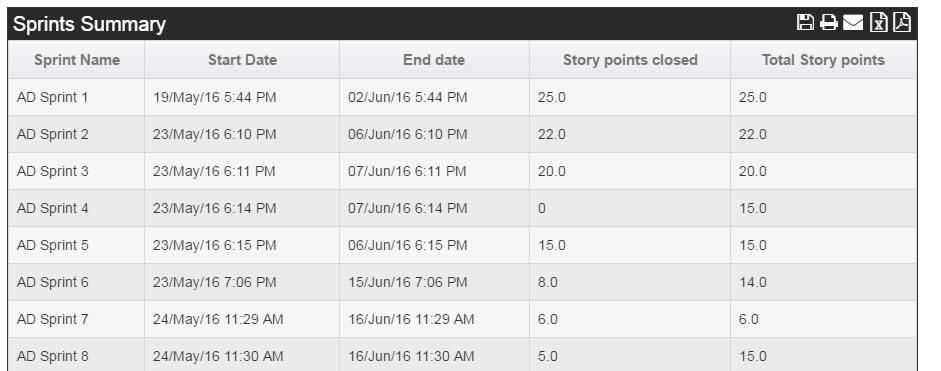
**Note**: BoardId and RapidViewId are one and the same.

* **Report Name: Sprint Report:**

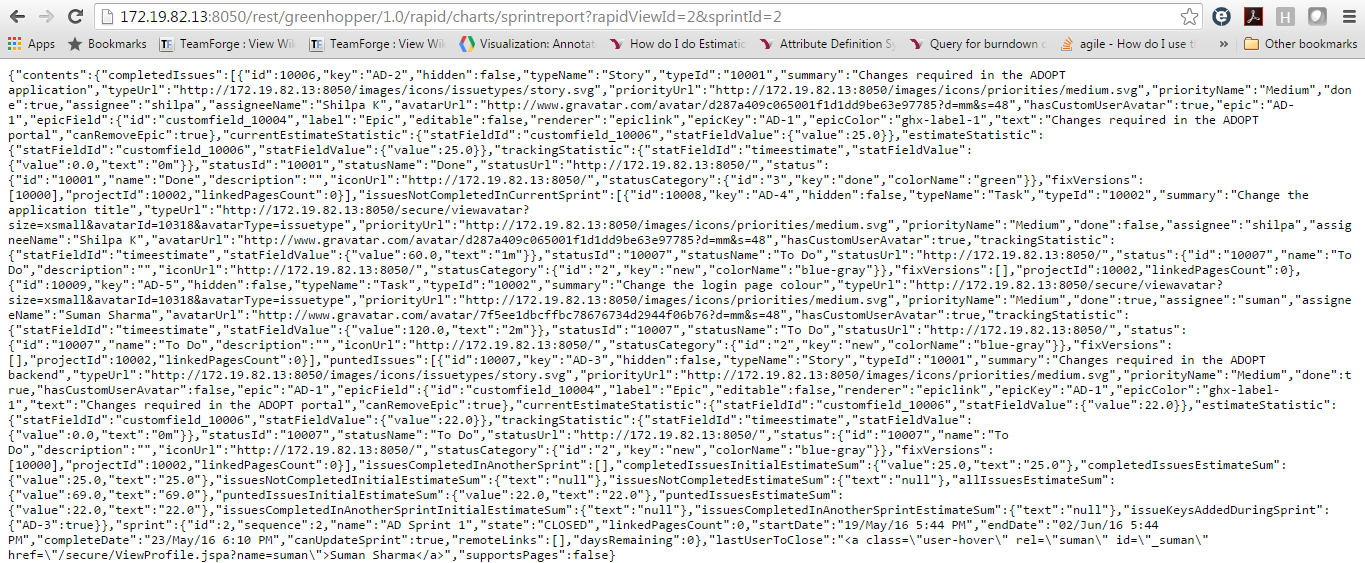
**This report displays below details in the report**

1. **Sprint name**
2. **Sprint Start Date**
3. **Sprint End Date**
4. **Story points completed**
5. **Total Story points planned**

**Fig 1.0 displays the report from the WebApp.**



**Fig 1.0**



**Fig 2.0**

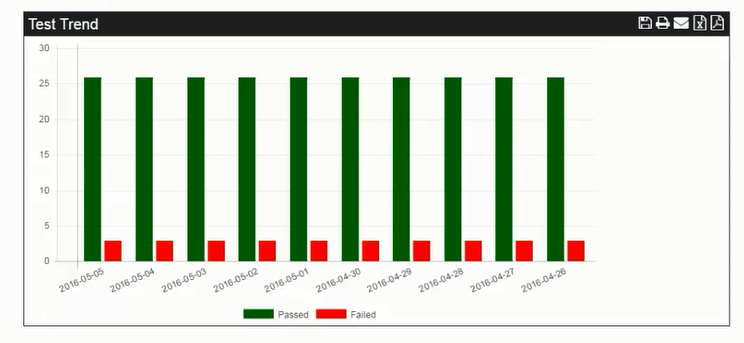
While fetching the results you have to pass rapidviewid and sprintid as parameters to the URL, i.e. “rapidViewId=$RAPIDVIEWID&sprintId=$SPINTID”. For ex: the URL will be http://172.19.82.13:8050/rest/greenhopper/1.0/rapid/charts/sprintreport?rapidViewId=2&sprintId=3. **Fig 2.0** displays the sample data.

* **Report Name: Test Trend:**

**This report displays below details in the report**

1. **Date**
2. **Passed count**
3. **Failed count**

**Fig 3.0 displays the report from the WebApp.**



**Fig 3.0**

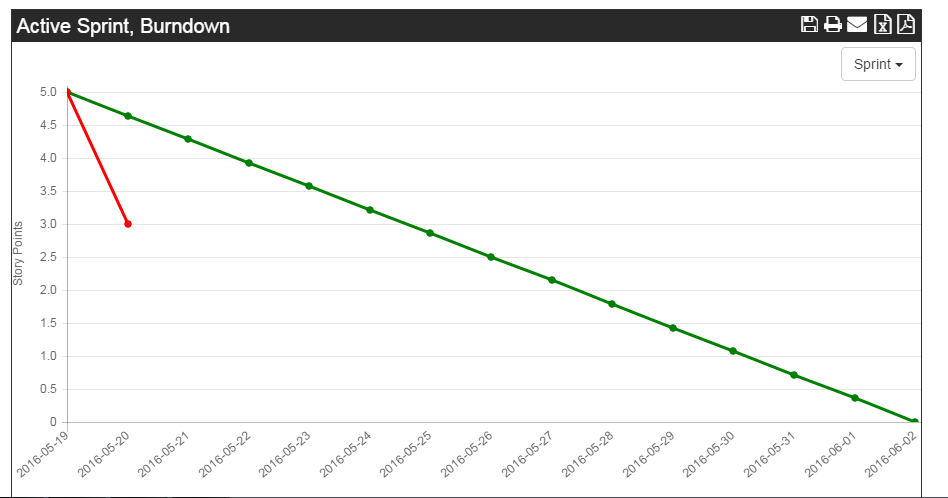
**Test trend report not done.**

* **Report Name: Sprint Burn down:**

**This report displays below details in the report**

1. **Sprint Start date-End date**
2. **Open Estimates between start date-end date**
3. **Ideal line**

**Fig 5.0 displays the report from the WebApp.**



**Fig 5.0**



**Fig 6.0**

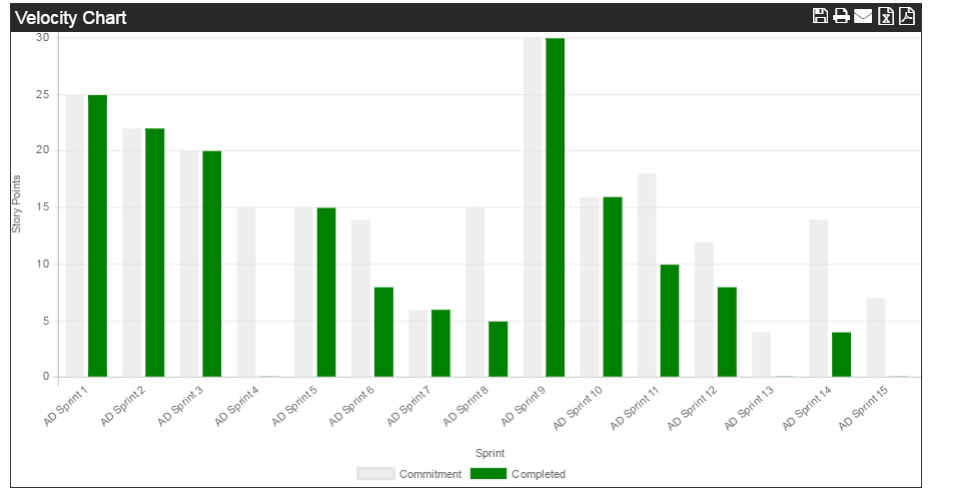
The parameters to pass are same as “Sprint report”. **Fig 6.0** displays the sample data.

* **Report Name: Velocity report:**

**This report displays below details in the report**

1. **Sprint Name**
2. **Total story points planned**
3. **Story points completed**

**Fig 7.0 displays the report from the WebApp.**



**Fig 7.0**

The valued for this report is same as values got from “sprint report”

**CODE FLOW:**

**JS file path:** WebContent/js/dash/projetcburndown.js

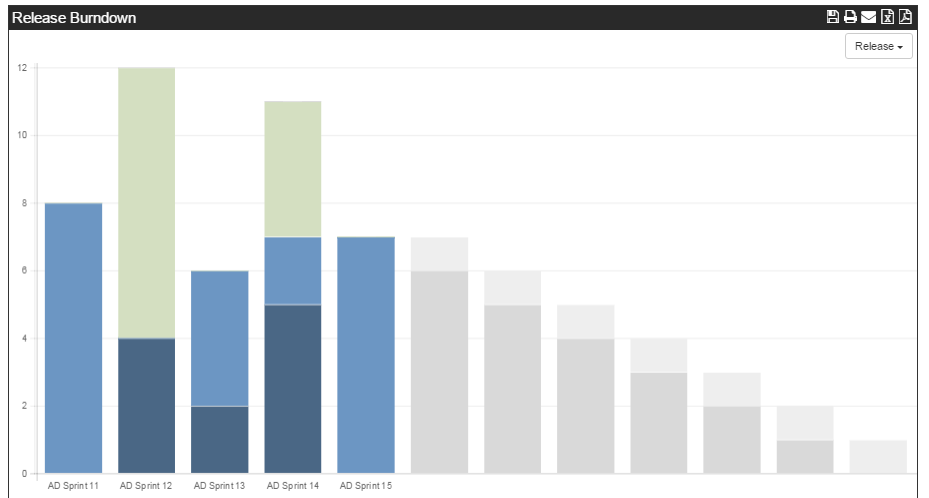
1. From projectburndown.js a get call is made as following, $.get('sprintreport', drawBarChart).
2. This will hit the ‘sprintreport’ servlet. Then ProjectReportsController calls the projectReportsService.getSprintDetails() method;
3. Then getSprintDetails method will call getSingleSprintDetails method for each sprint ids
4. Then in getSingleSprintDeatils method an API call is made to <http://172.19.82.13:8050/rest/greenhopper/1.0/rapid/charts/sprintreport?rapidViewId=2&sprintId=3>
5. Here we can get the following values: completedissuestimatesum, issuenotcompletedestimatesum, startdate, enddate, sprintname
6. Then a Json object is constructed with the above values.

* **Report Name: Release burn down:**

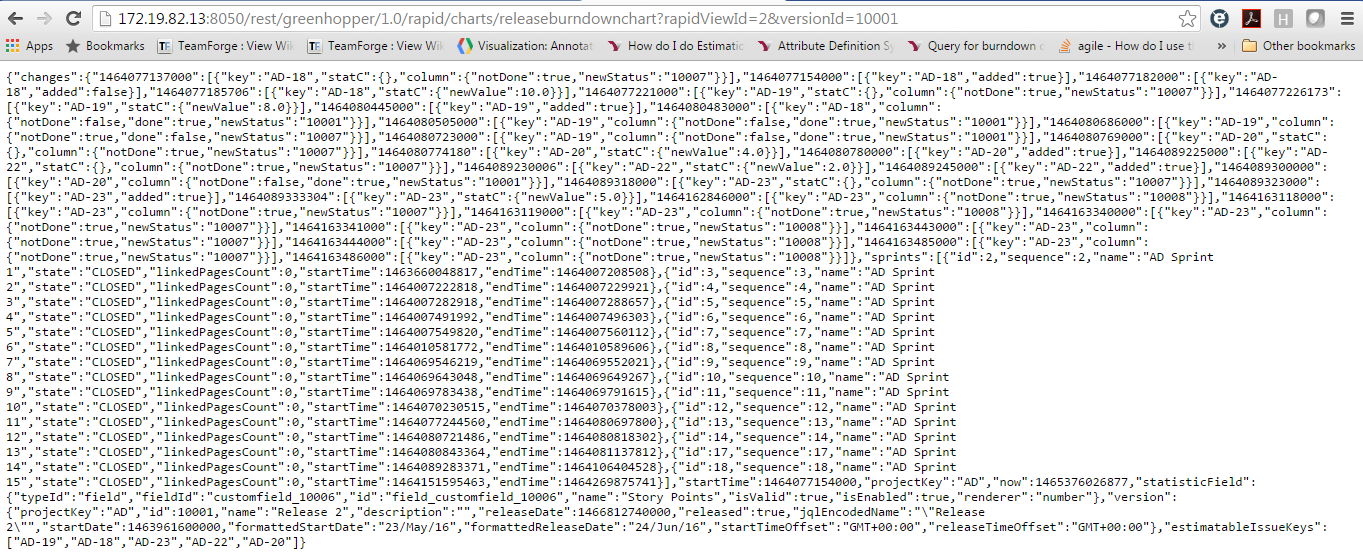
**This report displays below details in the report**

1. **Sprint Name**
2. **Work added in a sprint**
3. **Work completed in a sprint**
4. **Work remaining in a sprint**

**Fig 8.0 displays the report from the WebApp.**



**Fig 8.0**



**Fig 9.0**

While fetching the results you have to pass rapidviewid and versionid as parameters to the URL, i.e. “rapidViewId=$RAPIDVIEWID&versionId=$RELEASEID”. For ex: the URL will be http://172.19.82.13:8050/rest/greenhopper/1.0/rapid/charts/releaseburndownchart?rapidViewId=2&versionId=10001. **Fig 9.0** displays the sample data.

**CODE FLOW**

**JS file path:** WebContent/js/dash/projetcburndown.js

1. From projectburndown.js a get call is made as following, $.get('releaseburndown', {releasename: "Release 2"}, drawReleaseBurndownChart). Since filtering is not added, “Release 2” is hard coded.
2. Then releaseburdown servlet is called i.e ProjectReportsController class.
3. Controller calls”projectReportsService.releaseBurndown(releaseId);”.
4. In releaseBurndown method we will call the following API <http://172.19.82.13:8050/rest/greenhopper/1.0/rapid/charts/releaseburndownchart?rapidViewId=2&versionId=10001>.

Which gives the key value pair of date in ms , work added and sprint details.

1. After this we call two methods calWorkAdded () and getOutput ().
2. calWorkAdded method calculates the work added between the particular sprint start date and end date. For ex: If “sprint 1” start date and end date is “2016-06-15 23:45:23” and “2016-06-30 22:42:15” respectively, in this method we are calculating how much work is added between these dates.
3. Also, in calWorkAdded method if there is a work added between previous sprint end date and the next sprint start date the “workadded” is considered for the previous sprint and added to the previous sprint workadded.
4. Next we call getOuput method, where we calculate how much work was (is) completed and how much work is remaining. For this we will use the following api <http://172.19.82.13:8050/rest/greenhopper/1.0/rapid/charts/sprintreport?rapidViewId=2&sprintId=3>
5. From the above two methods we will be getting two maps objects

**From calWorkAdded:** {null=25, AD Sprint 9=16, AD Sprint 13=0, AD Sprint 1=22, AD Sprint 12=0, AD Sprint 2=0, AD Sprint 3=0, AD Sprint 11=0, AD Sprint 4=0, AD Sprint 10=0, AD Sprint 5=24, AD Sprint 6=0, AD Sprint 15=0, AD Sprint 7=0, AD Sprint 14=0, AD Sprint 8=20}

**From getOutput:** {3={completedIssuesEstimateSum=22, issuesNotCompletedEstimateSum=0, sprintName=AD Sprint 2}, 2={completedIssuesEstimateSum=25, issuesNotCompletedEstimateSum=0, sprintName=AD Sprint 1}, 10={completedIssuesEstimateSum=30, issuesNotCompletedEstimateSum=0, sprintName=AD Sprint 9}, 7={completedIssuesEstimateSum=8, issuesNotCompletedEstimateSum=6, sprintName=AD Sprint 6}, 6={completedIssuesEstimateSum=0, issuesNotCompletedEstimateSum=0, sprintName=AD Sprint 5}, 5={completedIssuesEstimateSum=0, issuesNotCompletedEstimateSum=0, sprintName=AD Sprint 4}, 4={completedIssuesEstimateSum=0, issuesNotCompletedEstimateSum=0, sprintName=AD Sprint 3}, 17={completedIssuesEstimateSum=0, issuesNotCompletedEstimateSum=0, sprintName=AD Sprint 14}, 18={completedIssuesEstimateSum=0, issuesNotCompletedEstimateSum=0, sprintName=AD Sprint 15}, 9={completedIssuesEstimateSum=0, issuesNotCompletedEstimateSum=10, sprintName=AD Sprint 8}, 8={completedIssuesEstimateSum=6, issuesNotCompletedEstimateSum=0, sprintName=AD Sprint 7}, 13={completedIssuesEstimateSum=0, issuesNotCompletedEstimateSum=0, sprintName=AD Sprint 12}, 14={completedIssuesEstimateSum=0, issuesNotCompletedEstimateSum=0, sprintName=AD Sprint 13}, 11={completedIssuesEstimateSum=16, issuesNotCompletedEstimateSum=0, sprintName=AD Sprint 10}, 12={completedIssuesEstimateSum=0, issuesNotCompletedEstimateSum=0, sprintName=AD Sprint 11}}

1. Once we get these data, we will call calculateReleaseBurndown (Map<String, Integer> map, Map<String, Map<String, String>> issuemap) by passing the map objects.
2. In calculateReleaseBurndown we will get the work added in a sprint and get the work completed (completedIssuesEstimateSum) and work not completed (issuesNotCompletedEstimateSum) for the particular sprint and build a json object having these values.
3. Work remaining is calculated using the below formula

workremaining = workadded in previous sprint + workremaining in previous sprint – worknotcompleted in current sprint.