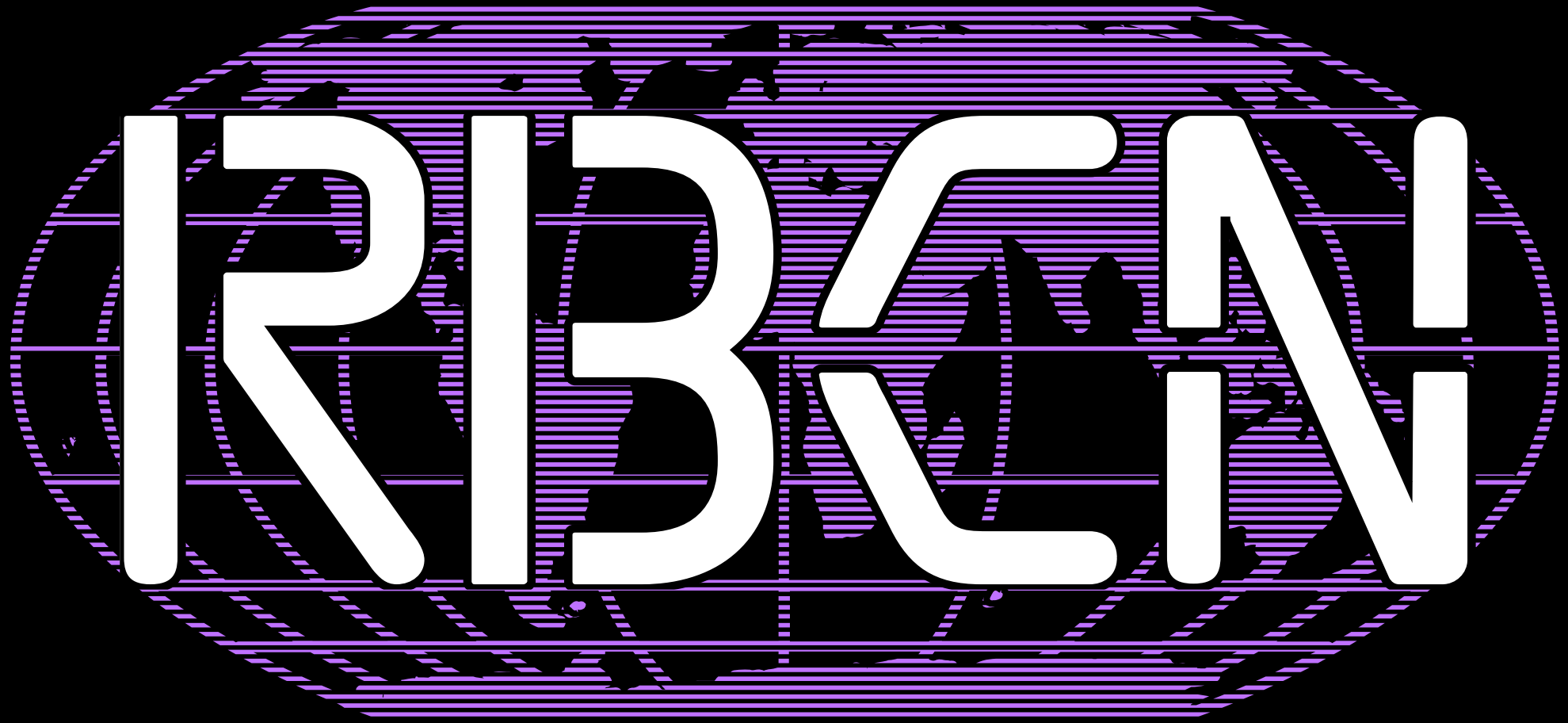
Helsinki:
8-9/February-24
Online:
28-29/February-24

FULL EVENT
PROGRAM VISIT
robocon.io

THE 7th ANNUAL
ROBOT FRAMEWORK
CONFERENCE



Robot
Framework
Foundation

RBCN24

ENHANCING TEST INSIGHTS

A DEEP DIVE INTO ROBOT FRAMEWORK REPORTING

Many Kasiriha

@Many
Join #reporting

“Imagine putting your heart into a masterpiece that no one sees.

That's what testing feels like without meaningful reports”

-ChatGPT

WHAT KIND OF REPORTS DO WE HAVE?
DO WE NEED MORE?

report.html	log.html	xunit.xml	output.xml	output.json
• Overview of test execution results	• Detailed test log of executed tests	• test execution summary in xUnit compatible format	• All execution results in machine readable XML format	• All execution results in JSON format

WHAT THE STANDARD REPORTS DON'T SAY (OR ONLY WITH SOME INVESTIGATION)

- What were the results last time?
- Was the run faster or slower than before?
- Were the tests overlapping ?
- Is it 10 failures or 10 times the same failure?
- Is that a flaky test?
- Which tests take really long?

AGENDA

1. How to use the Robot API for Reporting (30 min)
   1. Create Markdown Report for GH Actions
   2. Send MS Teams/Slack Notification
2. Using Grafana for Test Result Visualization (30 min)
   1. Reporting (with PostgreSQL)
   2. Monitoring (with InfluxDB)
3. Allure for Robot Framework Reporting (15 min)
4. ReportPortal (15 min)
5. General Reporting ideas and other RF Reporting Projects
6. Q&A and discussions

WORKSHOP SETUP

https://github.com/manykarim/robot-framework-reporting

What you need

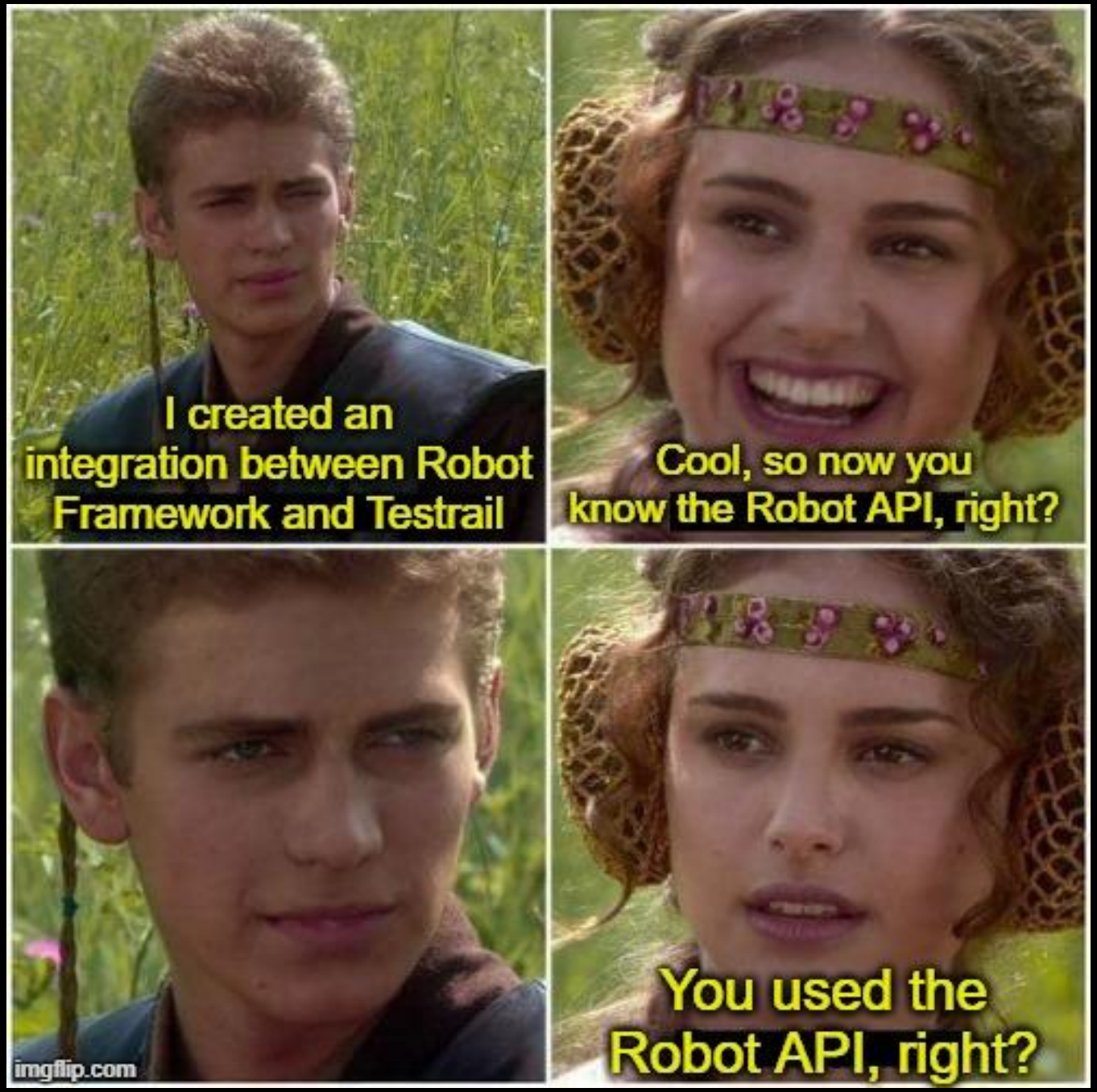
- GitPod Account (Free account is ok)
  or
- an environment with
  - Python 3.8+
  - Nodejs (for Browser Library)
  - Docker for Grafana and DBs
    (or anything else which
    can run Dockerimages)
  - Java 8+ for Allure

https://bit.ly/rf-reporting

ROBOT API

(to access test results)

01_resultmodel/



RBCN24

WHY YOU SHOULD KNOW THE ROBOT API

There might be a time when you want to

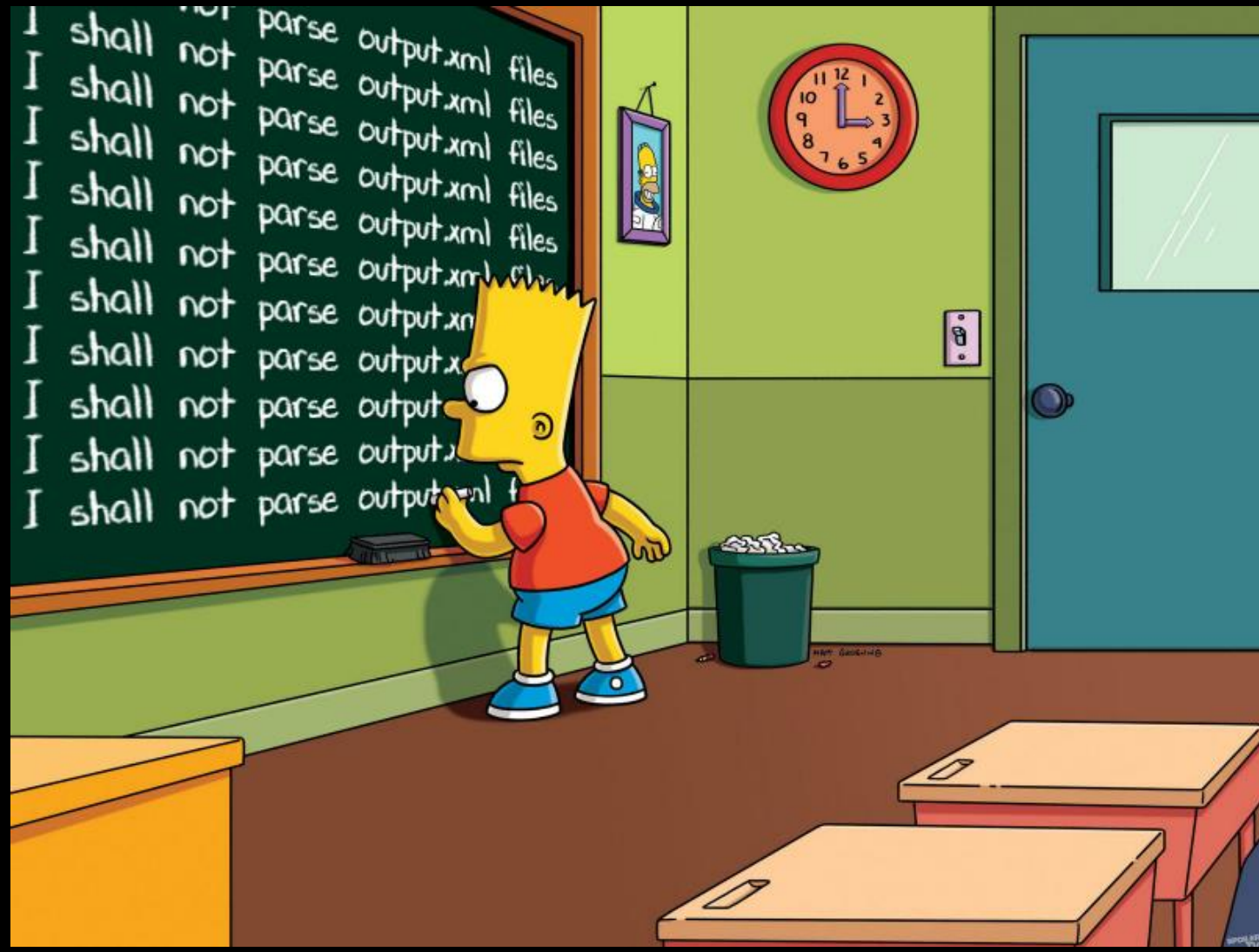
- Send your results to another system
- Create a custom report
- Collect your result data for further analysis
- Modify your test results
- ...

...and you realize that there is no existing Python Package for that

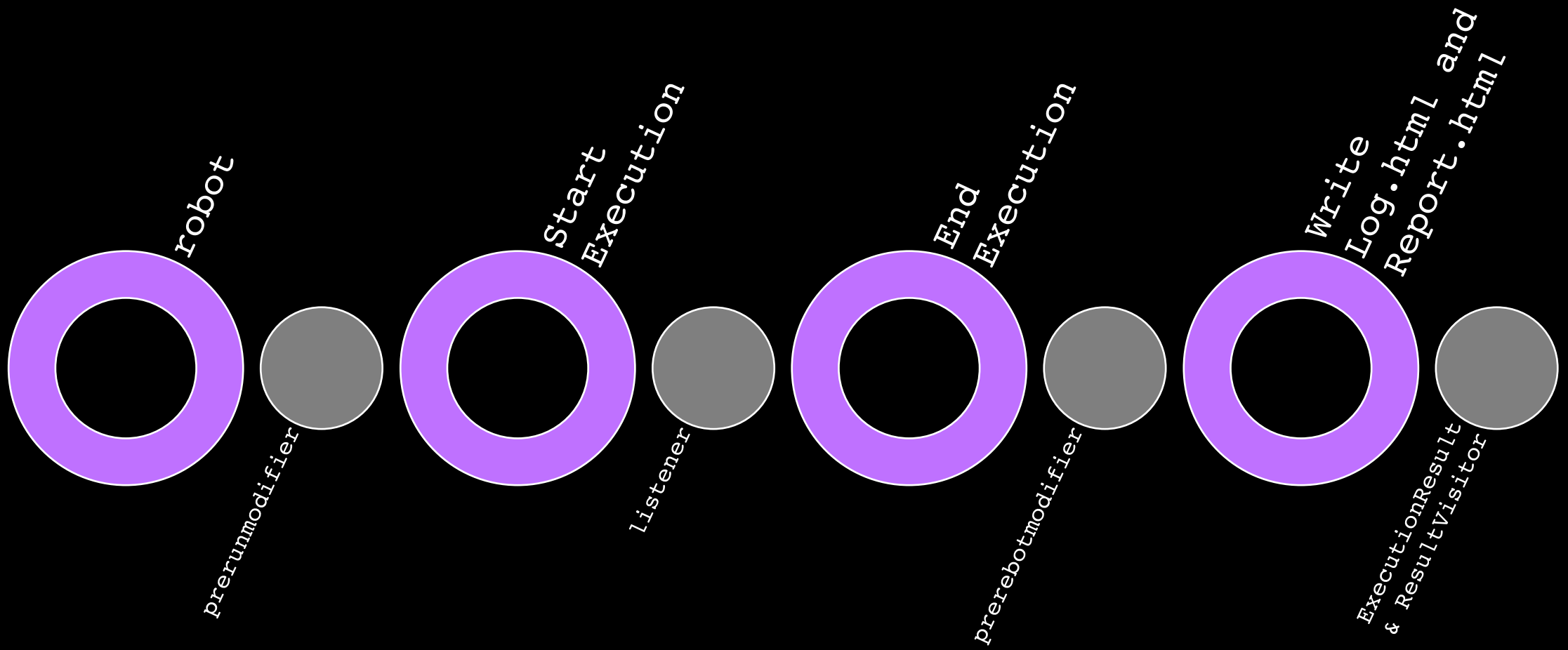
WHEN READING RESULTS

- Don‘t parse the .xml !!

- Use the Robot API to read the results programmatically from Python



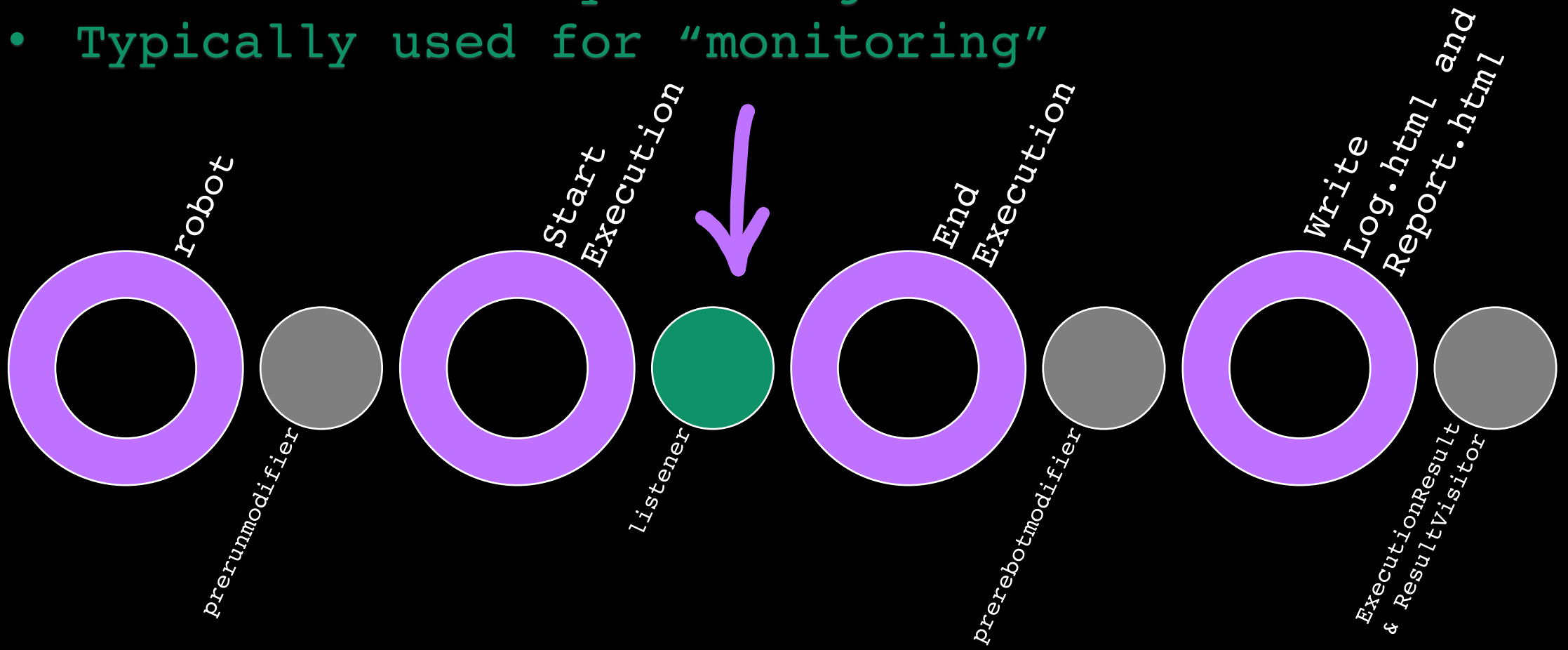
A ROBOT FRAMEWORK RUN



RBCN24

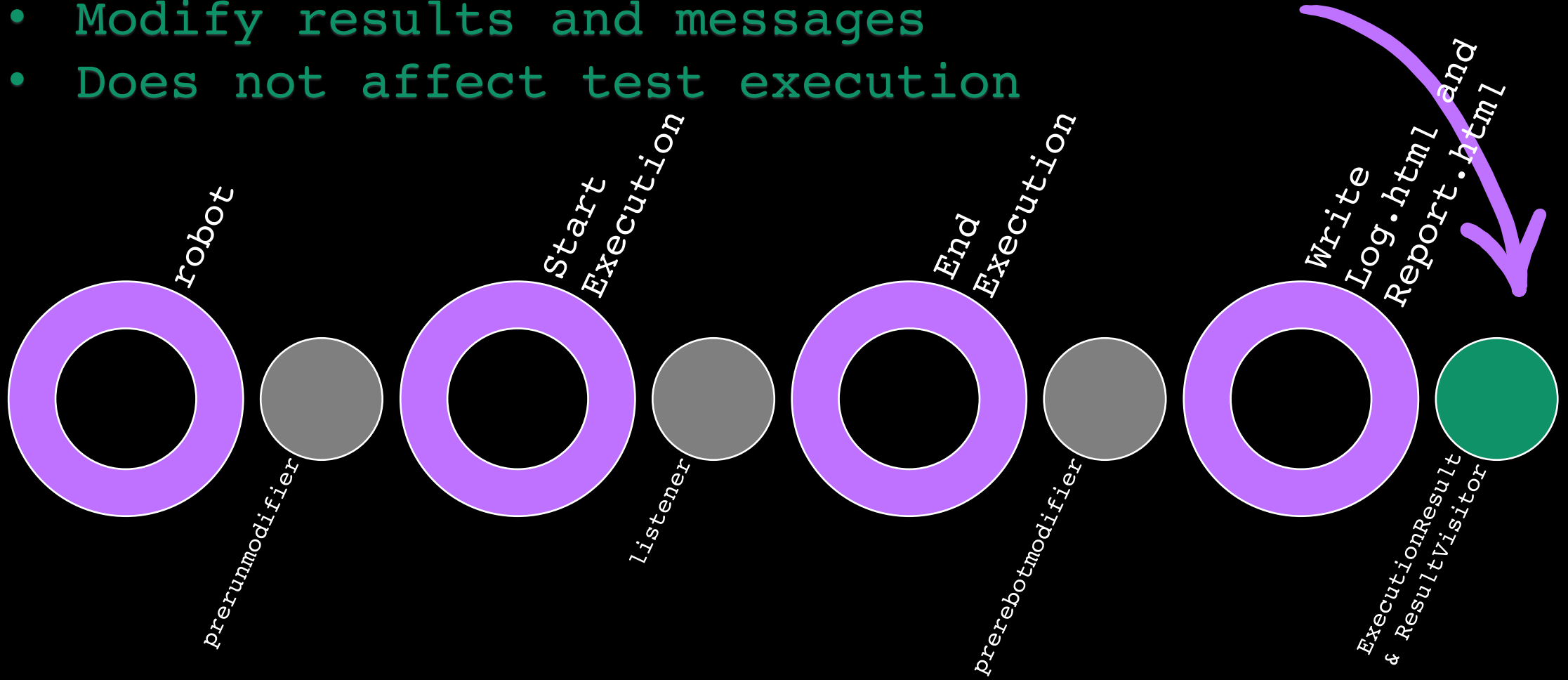
Hook into execution and react on Events

- Start/End Test/Suite/Keyword/..
- Access and modify running model
- Typically used for “monitoring”

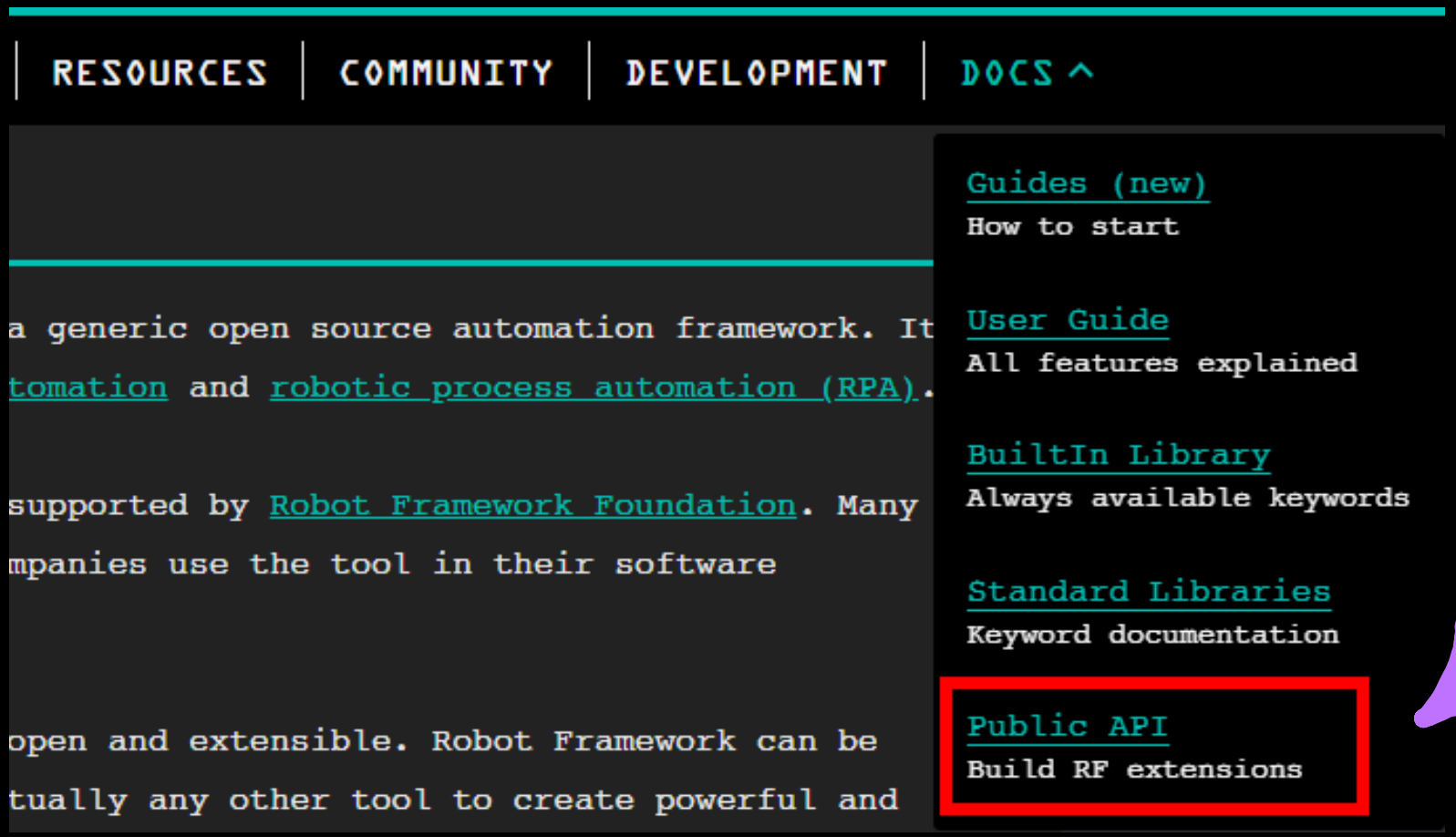


Access and modify Test Results after execution

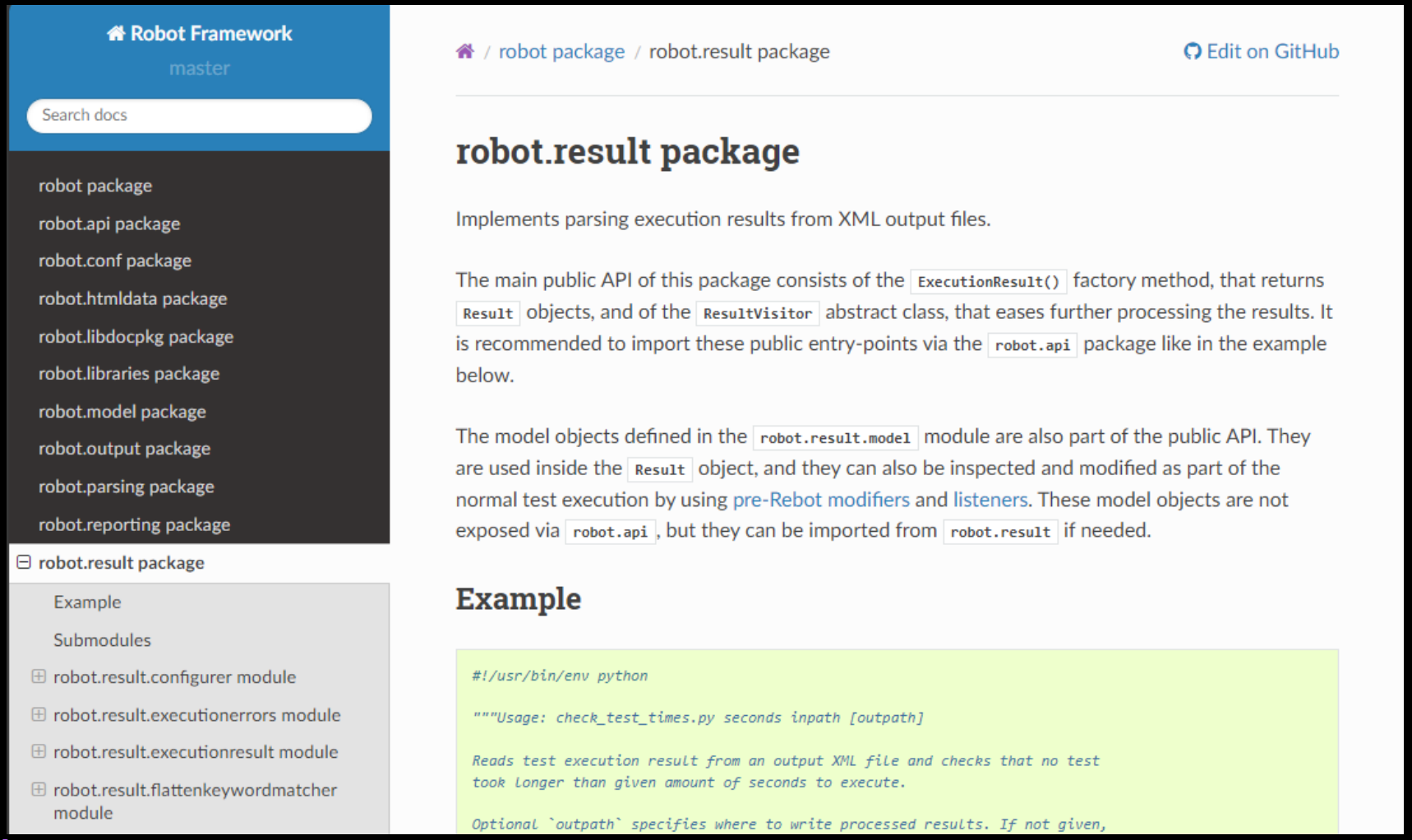
- Report to other tools
- Modify results and messages
- Does not affect test execution



THE ROBOT FRAMEWORK API AND THE RESULT MODEL



THE ROBOT FRAMEWORK API AND THE RESULT MODEL



RBCN24

RESULT OBJECT

- statistics
  - Numbers of passed, failed and skipped tests

- suite
  - Root TestSuite object

- visit(visitor)
  - Method to walk through results using a ResultVisitor

```
from robot.api import ExecutionResult

result = ExecutionResult('output.xml')

stats = result.statistics

result_dict =   {
    "Passed": stats.total.passed,
    "Failed": stats.total.failed,
    "Skipped": stats.total.skipped,
    "Total": stats.total.total
                }
print(result_dict)
root_suite = result.suite
```

TESTSUITE OBJECT

- name/full_name
- 📄 all_tests[]
  - Yields all tests of this suite and its children
- 📄 tests
  - Tests of this test suite
- ⏱ elapsed_time
- ☑/✕ status

```
from robot.api import ExecutionResult

result = ExecutionResult('output.xml')
root_suite = result.suite

for test in root_suite.all_tests:
    test_dict = {
        "Test": test.name,
        "Suite": test.parent.name,
        "Status": test.status
                }
    print(test_dict)
```

RBCN24

TESTCASE OBJECT

- name/full_name

- parent
  - Test Suite to which Test belongs

- tags

- ⏱ elapsed_time

- ☑/✕ status

```
from robot.api import ExecutionResult

result = ExecutionResult('output.xml')
root_suite = result.suite

for test in root_suite.all_tests:
    test_dict = {
        "Test": test.name,
        "Suite": test.parent.name,
        "Status": test.status
                }
    print(test_dict)
```

RBCN24

RESULTVISITOR
ABSTRACT CLASS

Can implement:
- start_result
- start_test
- start_suite
- start_keyword
- start_message
- ....

Also end_ and visit_ (instead of start_) can be implemented

```
from robot.api import ExecutionResult, ResultVisitor

class SuitesWithTestsVisitor(ResultVisitor):

    def start_suite(self, suite):
      if suite.tests:
        print(f"{suite.name} contains tests")

    def start_test(self, test):
      print(f"{test.name} status is {test.status}")

result = ExecutionResult('output.xml')
suite_visitor = SuitesWithTestsVisitor()
result.visit(suite_visitor)
```

LET'S CREATE A REPORT IN MARKDOWN AND SHOW IT IN A GITHUB ACTION

- Collect overall statistics
- Collect Test Suites (with tests) and their statistics
- Collect Test Names and Status of executed tests
- Write everything to a markdown file

Test Status

Test Suite	Passed ✅	Failed ❌	Skipped 🔴	Total	Elapsed Time ⏱️
Tests.Booker.Restful Booker	3	0	0	3	0.340416
Tests.Booker.Restful Booker Json	1	0	0	1	0.215507
Tests.Login.Mfa Login	4	0	0	4	3.144582
Tests.Todo.Todo	0	3	0	3	4.321990

Tests.Booker.Restful Booker

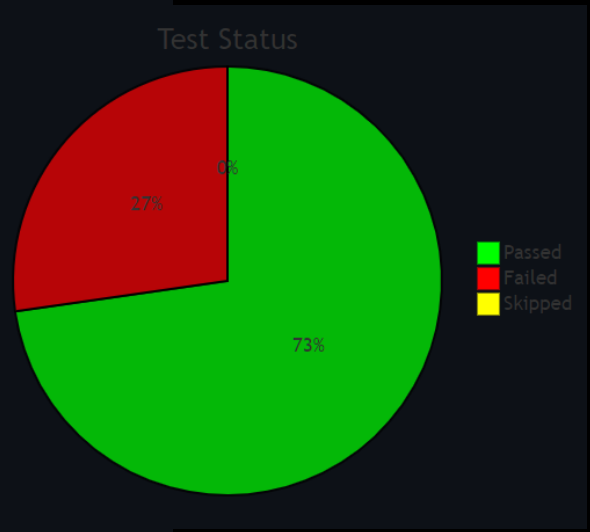
Test Case	Status	Elapsed Time ⏱️
Create Booking with Low Level Keywords	✅ PASS	0.068398
Create a Booking with High Level Keywords	✅ PASS	0.061478
Delete Booking	✅ PASS	0.136063

Tests.Booker.Restful Booker Json

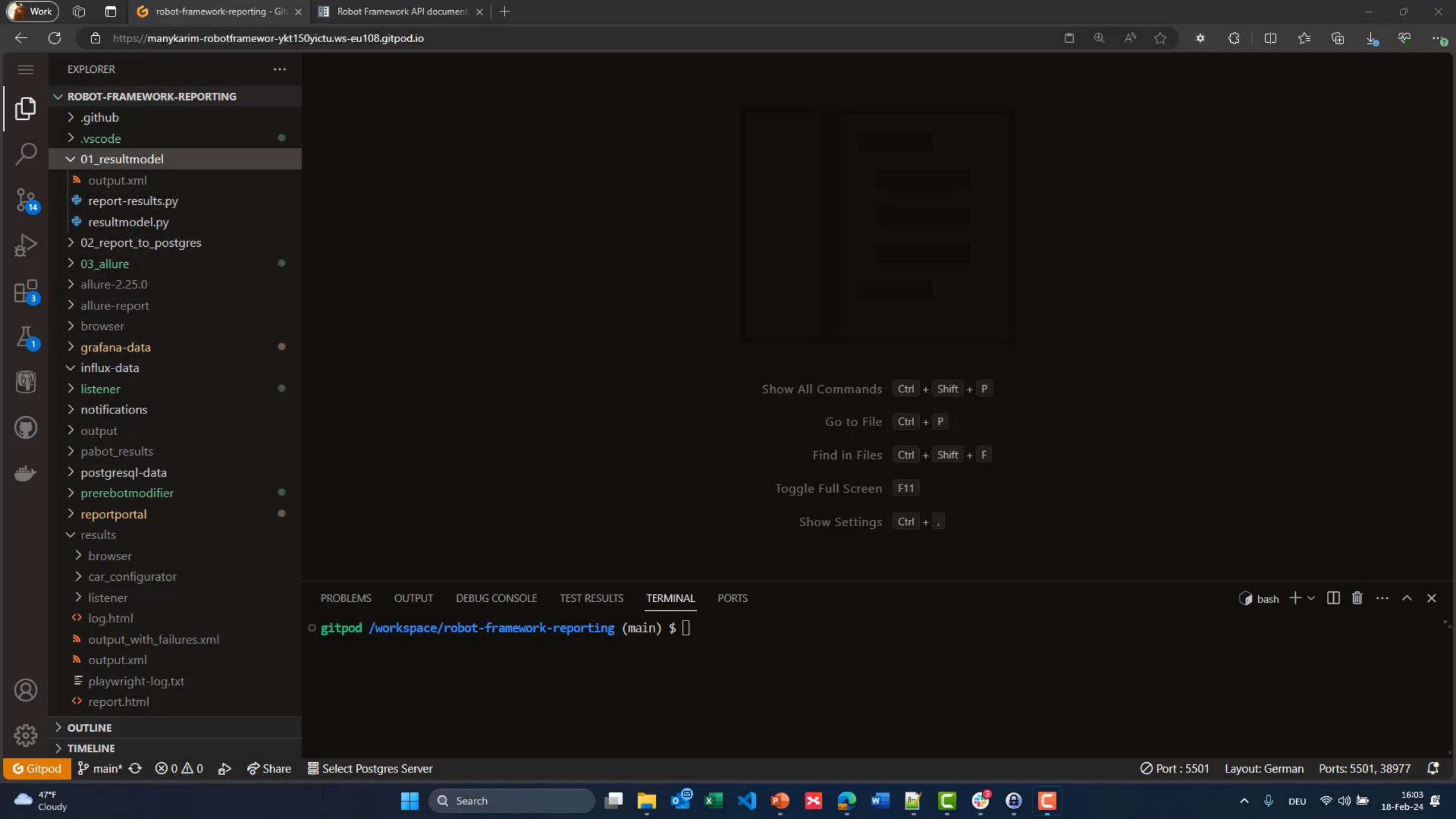
Test Case	Status	Elapsed Time ⏱️
Create a Booking	✅ PASS	0.165168

Tests.Login.Mfa Login

Test Case	Status	Elapsed Time ⏱️
Login with valid username, password and totp	✅ PASS	1.061443
Login with invalid username	✅ PASS	0.531396
Login with invalid password	✅ PASS	0.549314
Login with invalid totp	✅ PASS	0.550671



RBCN24


Work
robot-framework-reporting - Git
Robot Framework API document
https://manykarim-robotframewor-ykt150yictu.ws-eu108.gitpod.io
EXPLORER
ROBOT-FRAMEWORK-REPORTING
.github
.vscode
01_resultmodel
output.xml
report-results.py
resultmodel.py
02_report_to_postgres
03_allure
allure-2.25.0
allure-report
browser
grafana-data
influx-data
listener
notifications
output
pabot_results
postgresql-data
prerebotmodifier
reportportal
results
browser
car_configurator
listener
log.html
output_with_failures.xml
output.xml
playwright-log.txt
report.html
OUTLINE
TIMELINE
Show All Commands Ctrl + Shift + P
Go to File Ctrl + P
Find in Files Ctrl + Shift + F
Toggle Full Screen F11
Show Settings Ctrl + ,
PROBLEMS
OUTPUT
DEBUG CONSOLE
TEST RESULTS
TERMINAL
PORTS
bash
gitpod /workspace/robot-framework-reporting (main) $
Gitpod
main*
Share
Select Postgres Server
Port : 5501
Layout: German
Ports: 5501, 38977
47°F
Cloudy
Search
DEU
16:03
18-Feb-24

GITHUB ACTION SUMMARY WITH MARKDOWN

- There is an env variable **$GITHUB_STEP_SUMMARY**

- To create a Report Summary in Markdown, we can just write Markdown code **$GITHUB_STEP_SUMMARY**

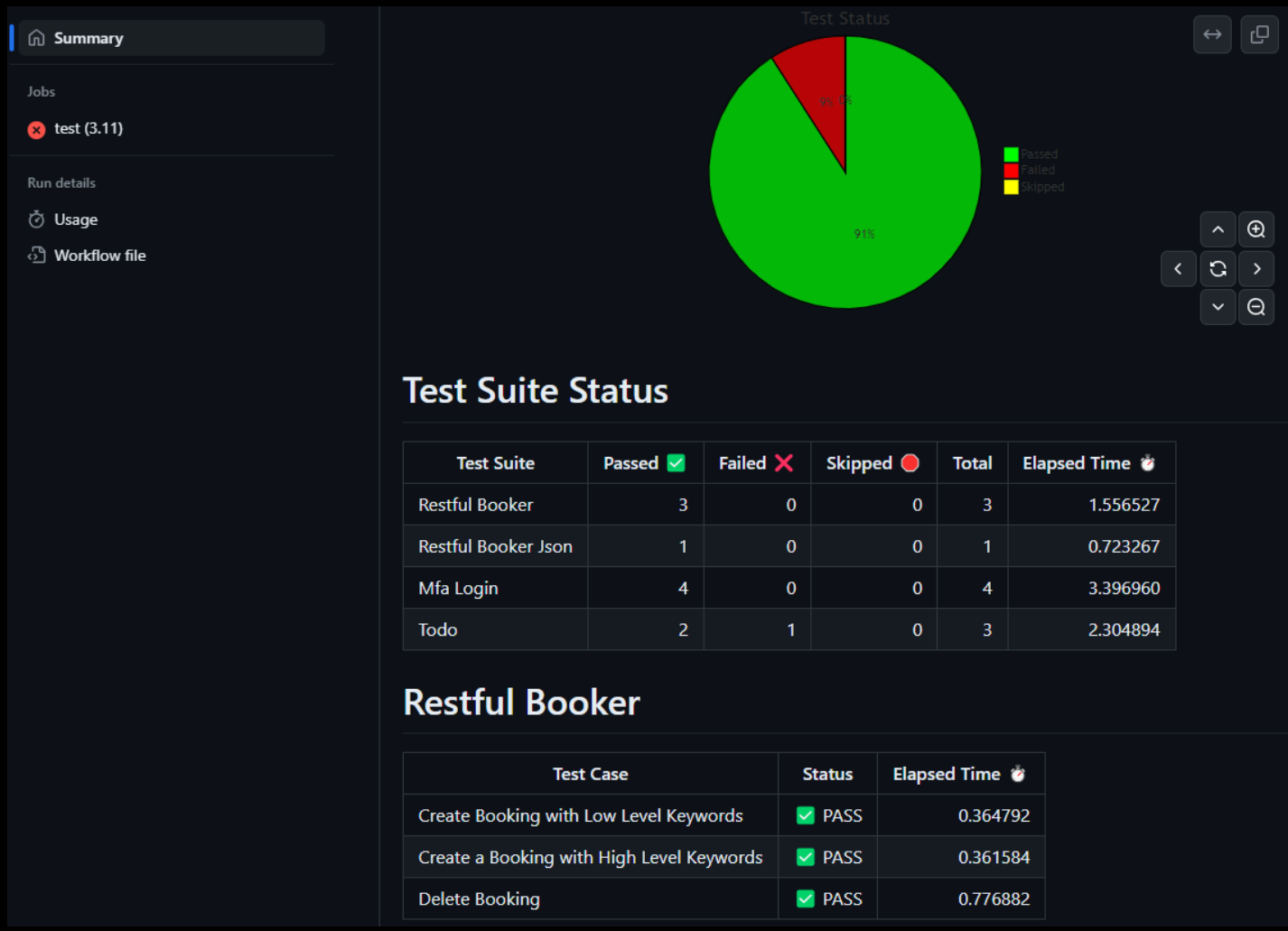
Bash

```
echo "### Hello world! :rocket:" >> $GITHUB_STEP_SUMMARY
```

example summary

Hello world! 🚀

Job summary generated at run-time



GITHUB ACTION

- That's how it looks like in the GitHub Action

```yaml
- name: Test with Robot Framework
  run: |
    robot -d results --exclude car-configurator tests
- name: Store Artifact
  uses: actions/upload-artifact@v3
  if: success() || failure()
  with:
    name: test-results ${{ matrix.python-version }} # Choose a descriptiv
    path: |
      results
- name: Report results
  if: success() || failure()
  run: |
    pip install -r utils/requirements.txt
    python utils/gha_reporter.py results/output.xml $GITHUB_STEP_SUMMARY
```

GHA_REPORTER

Check out the examples
- utils/gha_reporter.py

- 01_resultmodel/
  report-result.py

```
result = ExecutionResult(output_file)


# Write PieChart as Markdown
stats = result.statistics
f.write("```mermaid\n")
f.write("%%{init: {'theme': 'base', 'themeVariables': { 'pie1': '#00FF00', 'pie2': '#FF0000', 'pie3': '#FFFF00'}}}
f.write("pie title Test Status\n")
f.write(f'    "Passed" : {stats.total.passed}\n')
f.write(f'    "Failed" : {stats.total.failed}\n')
f.write(f'    "Skipped" : {stats.total.skipped}\n')
f.write("```\n")

# Get all suites with Tests
suite_visitor = SuitesWithTestsVisitor()
result.visit(suite_visitor)
suites_with_tests = suite_visitor.suites_with_tests

suite_results = []
for suite in suites_with_tests:
    suite_results.append([suite.name, suite.statistics.passed, suite.statistics.failed, suite.statistics.skipped,

# Write Table with Results for each Suite as Markdown
table_columns = ["Test Suite", "Passed ✅", "Failed ❌", "Skipped 🔴", "Total", "Elapsed Time ⏱️"]
writer = MarkdownTableWriter(table_name= "Test Suite Status", headers = table_columns, value_matrix = suite_result
writer.stream = f
writer.write_table()
f.write("\n")

for suite in suites_with_tests:
    tests_in_suite = []
    for test in suite.tests:
        status = lambda test: "✅ PASS" if test.status == "PASS" else ("❌ FAIL" if test.status == "FAIL" else "
        tests_in_suite.append([test.name, status(test), test.elapsed_time.total_seconds()])

    # Write Table with Results for each Test Case for a Suite as Markdown
    table_columns = ["Test Case", "Status", "Elapsed Time ⏱️"]
    writer = MarkdownTableWriter(table_name= suite.name, headers = table_columns, value_matrix = tests_in_suite)
    writer.stream = f
    writer.write_table()
    f.write("\n")
```

RBCN24

TASK

SEND NOTIFICATION TO MS TEAMS OR SLACK

05_notifications/

NOTIFICATIONS
TO MS TEAMS OR SLACK

RobotFramework 9:00 PM

Test run completed with 6 failures

Run Overview

Pass Rate	Passed ✅	Failed ❌	Skipped 🔴	Total	Elapsed Time ⏱
94.00%	94	6	0	100	621.239

Test Status

Test Suite	Passed ✅	Failed ❌	Skipped 🔴	Total	Elapsed Time ⏱
Tests.Car Configurator.Firstrun	94	6	0	100	621.152

Failed and skipped tests in Tests.Car Configurator.Firstrun

Test Case	Status	Message	Elapsed Time ⏱
Car 002	❌ FAIL	Error: page.goto: Timeout 30000ms exceeded. Call log: [2m- navigating to "https://vsr.testbench.com/", waiting until "load"[22m Tip: Use "Set Browser Timeout" for increasing the timeout or double check your locator as the targeted element(s) couldn't be found.	31.506
Car 003	❌ FAIL	Connection reset Also parent suite teardown failed: ConnectionError: Playwright process has been terminated with code 134	11.050
Car 063	❌ FAIL	Text '23.999,00 €' (str) should be '23.489,00 €' (str)	34.041
Car 064	❌ FAIL	Text '23.973,00 €' (str) should be '23.659,00 €' (str)	32.609
Car 065	❌ FAIL	Text '26.489,00 €' (str) should be '25.359,00 €' (str)	31.984
Car 070	❌ FAIL	Text '26.458,00 €' (str) should be '26.622,00 €' (str)	17.085

see less

Reply

Test run completed with 8 failures

Run Overview

Pass Rate: 33.33%

Passed ✅: 4

Failed ❌: 8

Skipped 🛑: 0

Total: 12

Elapsed Time ⏱️: 4.084

Failed Tests

* Login with valid username, password and totp
* Login with invalid username
* Login with invalid password
* Login with invalid totp
* Add Two ToDos And Check Items
* Add Two ToDos And Check Wrong Number Of Items
* Add ToDo And Mark Same ToDo
* Add ToDo And Mark Non Existing ToDo

Show less

Apprise | Today at 10:22 PM

NOTIFICATIONS WITH APPRISE

https://github.com/caronc/apprise

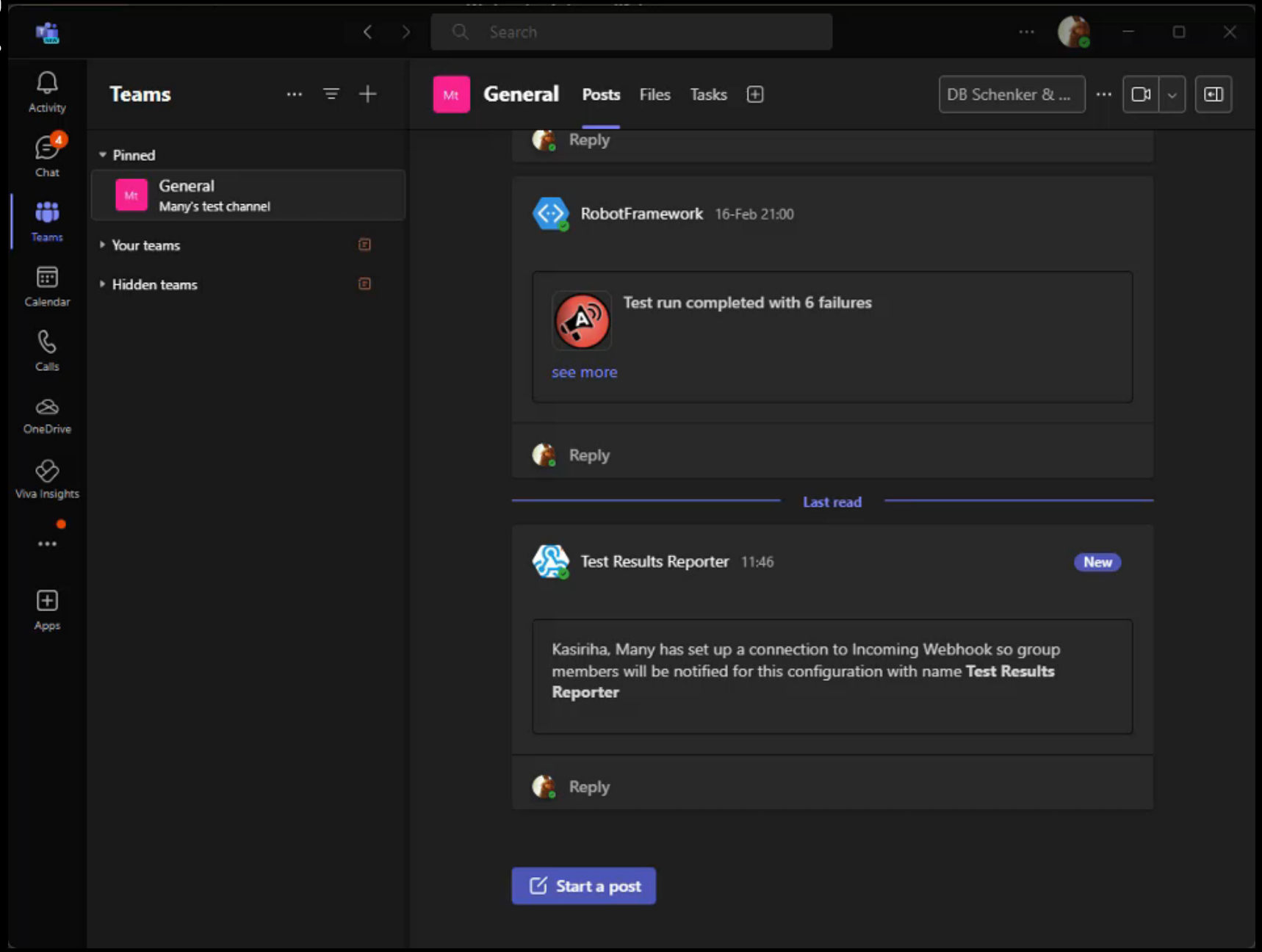
Sends notifications to most services:
- Slack, Teams, Discord, WhatsApp, ...

Usage via CLI or API

```
pip install apprise
```

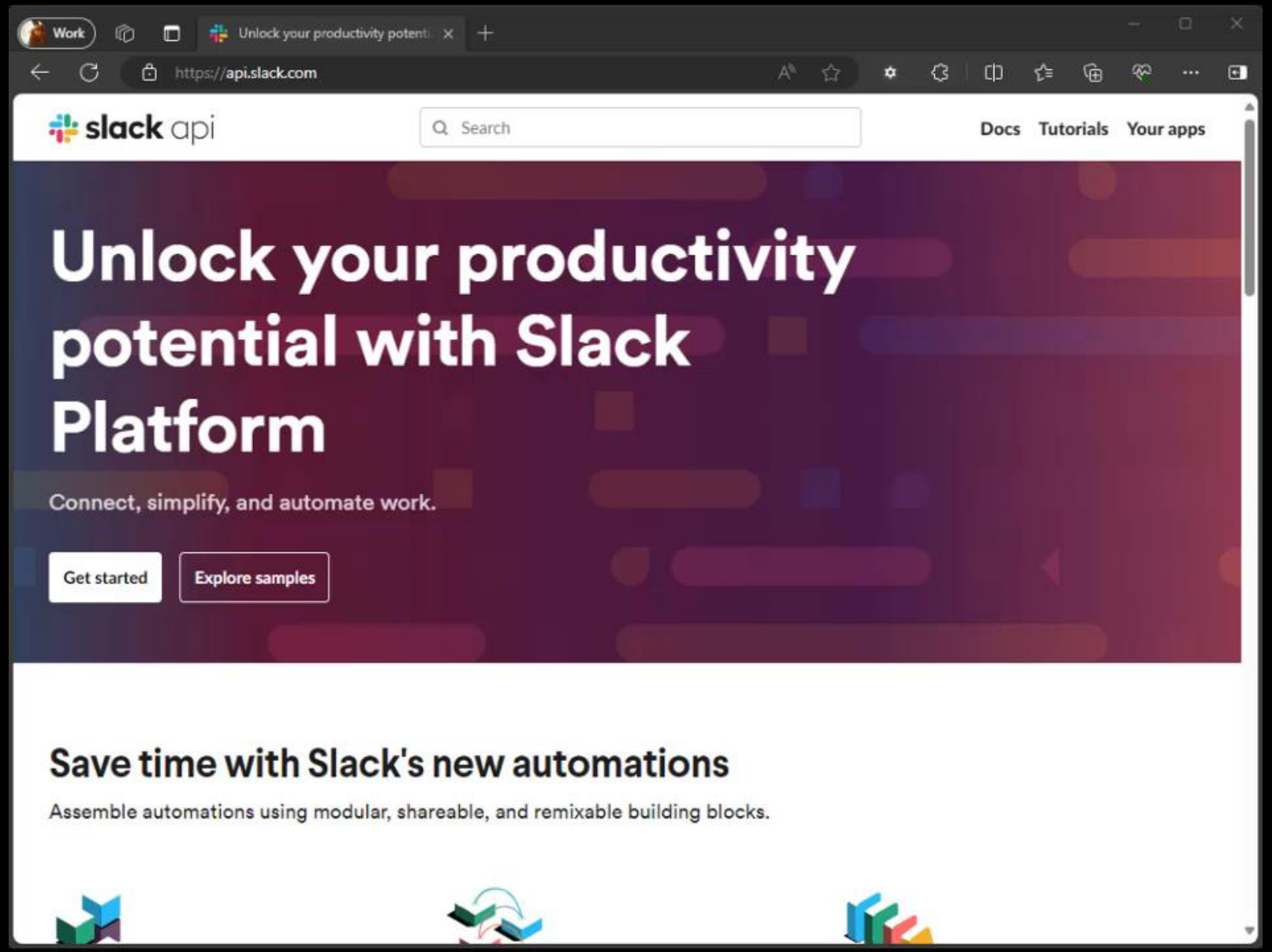
```python
run_overview = open('run_overview.md', "r").read()
suite_overview = open('suite_overview.md', "r").read()
results = open('test_results.md', "r").read()

import apprise
from apprise import NotifyType

apobj = apprise.Apprise()

apobj.add('msteams://<teamName>/<tokenA>/<tokenB>/<tokenC>')

if stats.total.failed == 0:
    title_message = "Test run completed successfully"
    body_message = run_overview + suite_overview
    apobj.notify(body=body_message, title=title_message, notify_type=NotifyType.SUCCESS)
else:
    title_message = f"Test run completed with {stats.total.failed} failures"
    body_message = run_overview + suite_overview + results
    apobj.notify(body=body_message, title=title_message, notify_type=NotifyType.FAILURE)
```

WEBHOOK FOR MS TEAMS

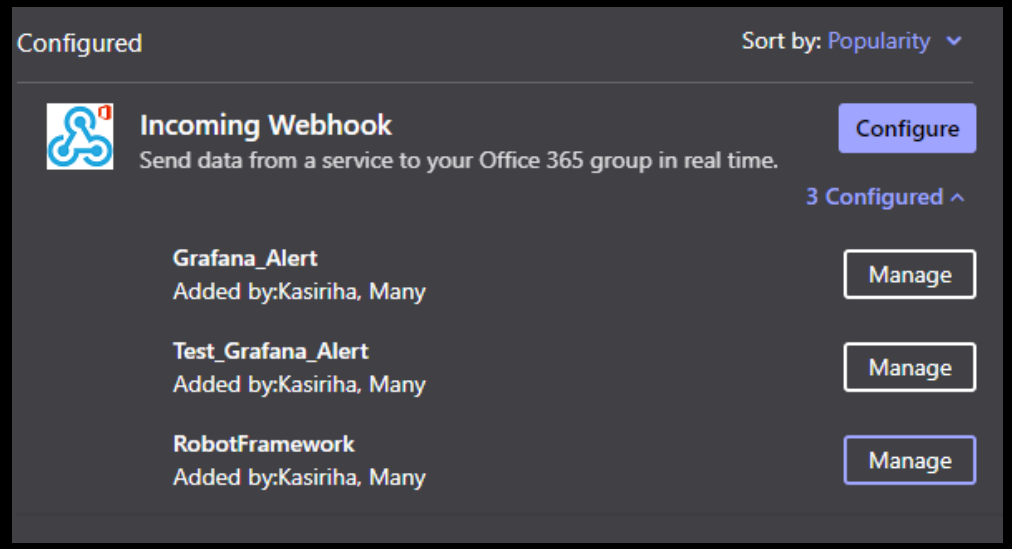


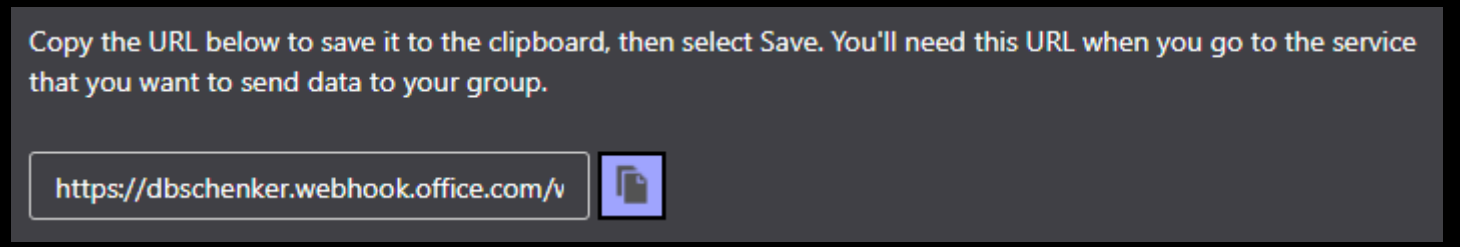
RBCN24

WEBHOOK FOR SLACK



WEBHOOK URLS IN APPRISE





https://mycompany.webhook.office.com/webhookb2/
2ca1e245-2c0e-1253-a754-f222fdd23750@c2d1e235-e4b2-42bf-23ff-42f23453e0a2/
IncomingWebhook/
123ebd5d63456e12b5d543e234d5fd23/
57d234f4-c6ad-51f3-aa14-f120301f0145

https://{team}.office.com/webhook/{tokenA}/IncomingWebhook/{tokenB}/{tokenC}

```python
apobj = apprise.Apprise()

apobj.add('msteams://<teamName>/<tokenA>/<tokenB>/<tokenC>')

if stats.total.failed == 0:
    title_message = "Test run completed successfully"
    body_message = run_overview + suite_overview
    apobj.notify(body=body_message, title=title_message, notify_type=NotifyType.SUCCESS)
```

RBCN24

SHORT BREAK
AND TIME FOR QUESTIONS

TASK FOR THE BREAK

USE ROBOT.API TO READ
MESSAGES OF FAILED TESTS

SEND A NOTIFICATION TO
YOUR MS TEAMS OR SLACK CHANNEL

TASK

WRITE TEST RESULTS TO A DATABASE AND VISUALIZE WITH GRAFANA

02_report_to_postgres/
04_listener_influxdb/

RBCN24

GRAFANA

It‘s an Open-source visualization and monitoring tool.

Features

- Dynamic dashboards
- Supports multiple data sources
  - (e.g. Prometheus, InfluxDB, PostgreSQL, ..)
- Real-time alerts

Use Cases

- Time series analytics
- Application monitoring
- Operational intelligence

RBCN24

A QUICK ROUNTRIP THROUGH  GRAFANA

Work | Gitpod: Always ready-to-code. | start_grafana.sh - robot-framew

https://manykarim-robotframewor-ykt150yictu.ws-eu108.gitpod.io

DOCKER

CONTAINERS
- Individual Containers
  - influxdb:latest influx - Ex...
  - postgres postgres - Exite...
  - grafana/grafana-enterp...

IMAGES
- docker.elastic.co/elastics...
- docker.io/bitnami/rabbit...
- docker.io/grafana/grafan...
- docker.io/minio/minio
- docker.io/reportportal/m...
- docker.io/reportportal/se...
- docker.io/reportportal/se...

REGISTRIES
- Connect Registry...

NETWORKS

VOLUMES

CONTEXTS

HELP AND FEEDBACK
- Read Extension Documentati...
- Watch Extension Tutorial Vid...
- Get Started with Docker Tuto...
- Open Docker Extension Walk...
- Review Issues

start_grafana.sh

```
$ start_grafana.sh
1  #!/bin/bash
2  docker run -d -p 3000:3000 --name=grafana  --user "$(id -u)"  --volume "$PWD/grafana-data:/var/lib/grafana"  --network=robot grafana/grafana-enterprise
```

PROBLEMS | OUTPUT | DEBUG CONSOLE | TEST RESULTS | TERMINAL | PORTS

bash

```
gitpod /workspace/robot-framework-reporting (main) $
```

Gitpod | main* | 0 0 | Share | Select Postgres Server | Ln 2, Col 109 | Spaces: 4 | UTF-8 | LF | Shell Script | Go Live | Layout: German | Ports: 36571, 41045

48°F Cloudy | Search | DEU | 20:57 19-Feb-24

SOME THOUGHTS BEFORE WE START
TO STORE AND VISUALIZE STUFF

1. Think about WHAT you want to visualize, e.g.
   - Test Status ☑ and Elapsed Time ⏱
   - Overall Test Run Statistics 📊
   - Test Suite Statistics 📊 and Status ☑
   - ..
2. Create a simple Table Model in your Database
3. Send your data to a Database using robot api
4. Add Database to Grafana as Data Source
5. Create Dashboard
6. Add Panels to Dashboard using SQL Queries to retrieve Data

PROPOSAL FOR A DB MODEL

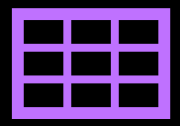


Table: **Test Run Overview**
-Columns: Test Suite, Passed, Failed, Skipped, Total, ElapsedTime, Starttime, Endtime, Metadata

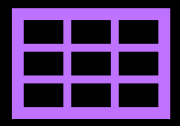


Table: **Test Suite Results**
-Columns: Test Suite, Passed, Failed, Skipped, Total, ElapsedTime, Starttime, Endtime, Metadata

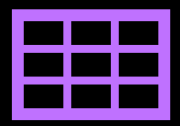


Table: **Test Results**
-Columns: Test Name, Status, ElapsedTime, Starttime, Endtime, Test Suite, Metadata

SIDE NOTE
POSTGRESQL HAS A JSON COLUMN TYPE

- Flexible Data Model thanks to JSON support in PostgreSQL

Index	testname	status	elapsed Time	metadata
1	Add Two ToDos	PASS	1.5	{„project“: „ToDo“, „version“:1.0}
2	Mark ToDo As Done	FAIL	12.8	{„project“: „ToDo“, „version“:1.0}
3	Add 200 ToDos	SKIP		{„project“: „ToDo“, „version“:1.0}
4	Delete Completed ToDo	PASS	5.4	{„project“: „ToDo“, „version“:1.0}

```
SELECT metadata->>‘project‘ from testresults
WHERE metadata->>‘version‘=‘1.0‘
```

HOW WE WILL UPLOAD DATA TO DB USING PANDAS AND SQLALCHEMY

- **pandas**
  „pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool”

- **sqlalchemy**
  „A Python SQL toolkit and Object Relational Mapper”

A PANDAS DATAFRAME

„A **DataFrame** is a 2-dimensional labeled data structure with columns of potentially different types. You can think of it like a spreadsheet or SQL table"

Index	Test Name	Status	Elapsed Time	Metadata
1	Add Two ToDos	PASS	1.5	{„project": „ToDo", „version":1.0}
2	Mark ToDo As Done	FAIL	12.8	{„project": „ToDo", „version":1.0}
3	Add 200 ToDos	SKIP		{„project": „ToDo", „version":1.0}
4	Delete Completed ToDo	PASS	5.4	{„project": „ToDo", „version":1.0}

SOME SNIPPETS

```
# Store some metadata in a dictionary
run_metadata = {"test_run": run, "environment": "dev", "branch": "main",
                "version": version, "build": "1.0.0", "project": project}

# Set up DB connection (we will use that later)
engine = create_engine('postgresql://postgres:robotframework@localhost:5432/robotframework_json')

# List of columns names
test_results_columns = ["test", "status", "elapsedtime", "testsuite",
                        "metadata", "starttime", "endtime"]

# Add details for every test into a nested list
for test in suite.tests:

    test_results.append([test.name, test.status, test.elapsed_time.total_seconds(),
                         suite.full_name, run_metadata, test.start_time, test.end_time])

# Create Pandas Dataframe from test results and columns
testresults_df = pd.DataFrame(test_results, columns=test_results_columns)


# Publish results to Database, columns types will be assigned automatically

test_run_results_df.to_sql('testrun_results', engine, if_exists='append',
                           index=True, dtype={'metadata': JSONB})
```

```python
test_results_columns = [
    "test",
    "status",
    "elapsedtime",
    "testsuite",
    "metadata",
    "starttime",
    "endtime",
]
```

```python
# Some metadata in a dictionary (I will later use that for filtering in Grafana)
run_metadata = {
    "test_run": run,
    "environment": "dev",
    "branch": "main",
    "version": version,
    "build": "1.0.0",
    "project": project,
}

# Set up DB connection
engine = create_engine(
    "postgresql://postgres:robotframework@localhost:5432/robotframework_json"
)
```

```python
for test in suite.tests:
    test_results.append(
        [
            test.name,
            test.status,
            test.elapsed_time.total_seconds(),
            suite.name,
            run_metadata,
            test.start_time,
            test.end_time,
        ]
    )
```

```python
testresults_df = pd.DataFrame(test_results, columns=test_results_columns)
```

```python
testresults_df.to_sql(
    "test_results",
    engine,
    if_exists="append",
    index=True,
    dtype={"metadata": JSONB},
)
```

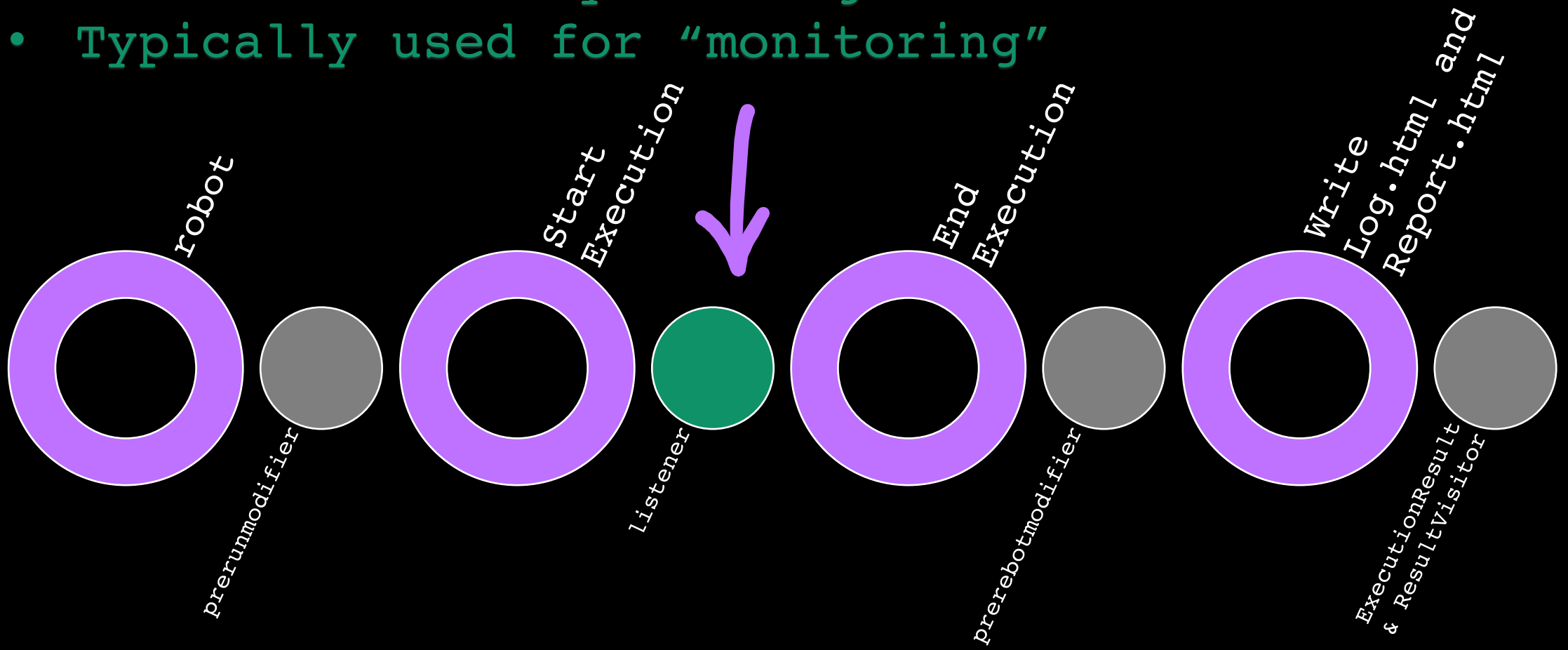
TASK

REAL TIME REPORTING

04_listener_influxdb/

Hook into execution and react on Events

- Start/End Test/Suite/Keyword/..
- Access and modify running model
- Typically used for “monitoring”



INFLUXDB

It‘s an Open-source time-series database

Features
- High Performance
- Scalable
- Flexible Query Language, InfluxQL („SQL-like“)

Use Cases
- Monitoring
- Internet of Things
- Real-Time Analytics

Works well with Grafana for data visualization

INFLUXDB KEY CONCEPTS

Data is organized into **buckets**
- They have a retention-period (e.g. 30 days)

Instead of “columns” we have:
- **Tags:** Indexed key-value pairs
  - e.g. tag(“project”:, “car-configurator”)
- **Fields:** Non-indexed measurement values
  - e.g. field(“status”:, test.status)
- **_time:** Timestamp for each data point
  - e.g. time(test.end_time)

LISTENER WITH INFLUXDB

```
class InfluxListener:

    ROBOT_LIBRARY_SCOPE = "GLOBAL"
    ROBOT_LISTENER_API_VERSION = 3

    def __init__(self):
        self.write_client = influxdb_client.InfluxDBClient(url=influxdb_url,
                                           token=influxdb_token, org=influxdb_org)
        self.bucket="results"
        self.write_api = self.write_client.write_api(write_options=SYNCHRONOUS)

    def end_test(self, test, result):
        point = (
            Point("testresult")
            .tag("name", result.name)
            .tag("suite", result.parent.name)
            .field("elapsedtime", result.elapsed_time.total_seconds())
            .field("status", result.status)
            .field("message", result.message)
            .time(result.end_time)
        )
        self.write_api.write(bucket=self.bucket, org="robotframework", record=point)
```

RBCN24

SHORT BREAK
AND TIME FOR QUESTIONS

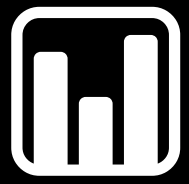
TASK FOR THE BREAK

EXPLORE THE GRAFANA DASHBOARDS

(Public Link will be provided)

User: robocon Password: robocon

WHAT INFORMATION WOULD YOU LIKE TO SEE THERE?



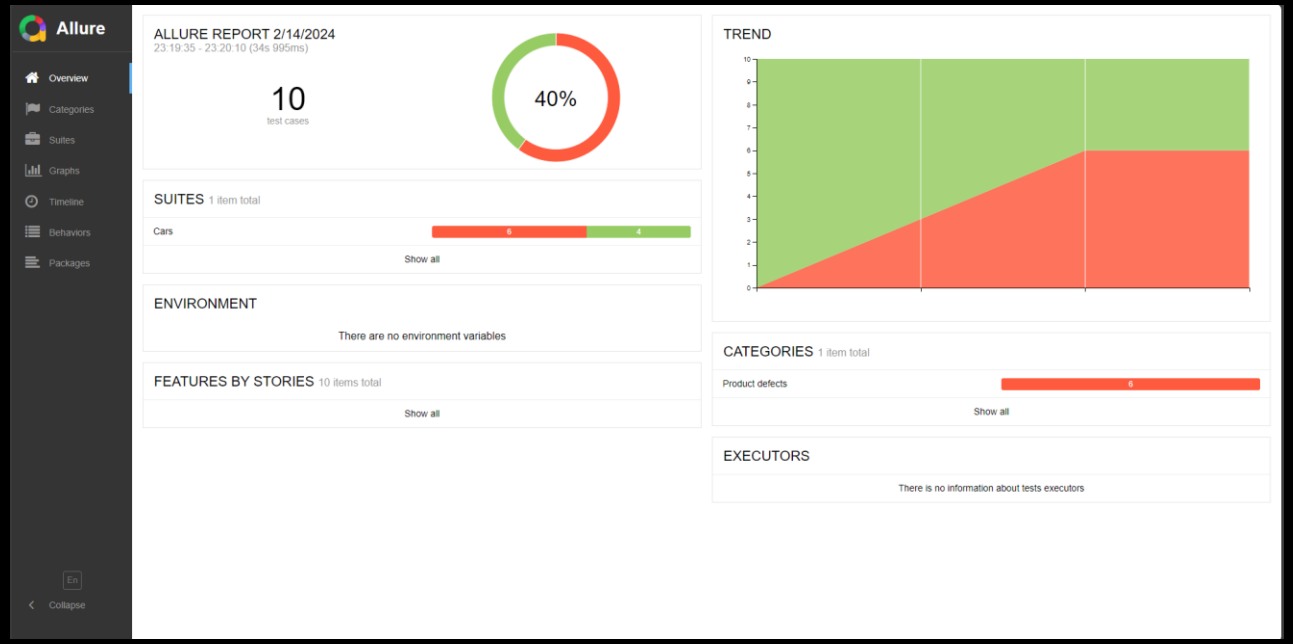
REPORTING WITH ALLURE

03_allure/

RBCN24

WHAT'S ALLURE?
A TEST AUTOMATION REPORTING TOOL

- Language and Framework agnostic
- Visualization of Status, Duration, History, Failures
- Open Source
- Report consists of .html , .js and .json files (can be shared)



INSTALLATION AND USAGE

Installation

```
# Download
wget https://github.com/allure-framework/allure2/releases/download/2.25.0/allure-2.25.0.tgz

tar -xzf allure-2.25.0.tgz

# Install Listener
pip install allure-robotframework
```

Usage

```
# Run Tests with Listener
robot --listener allure_robotframework ./my_robot_test

# Generate Report
allure generate output/allure

# Open Report in Browser
allure open allure-report
```

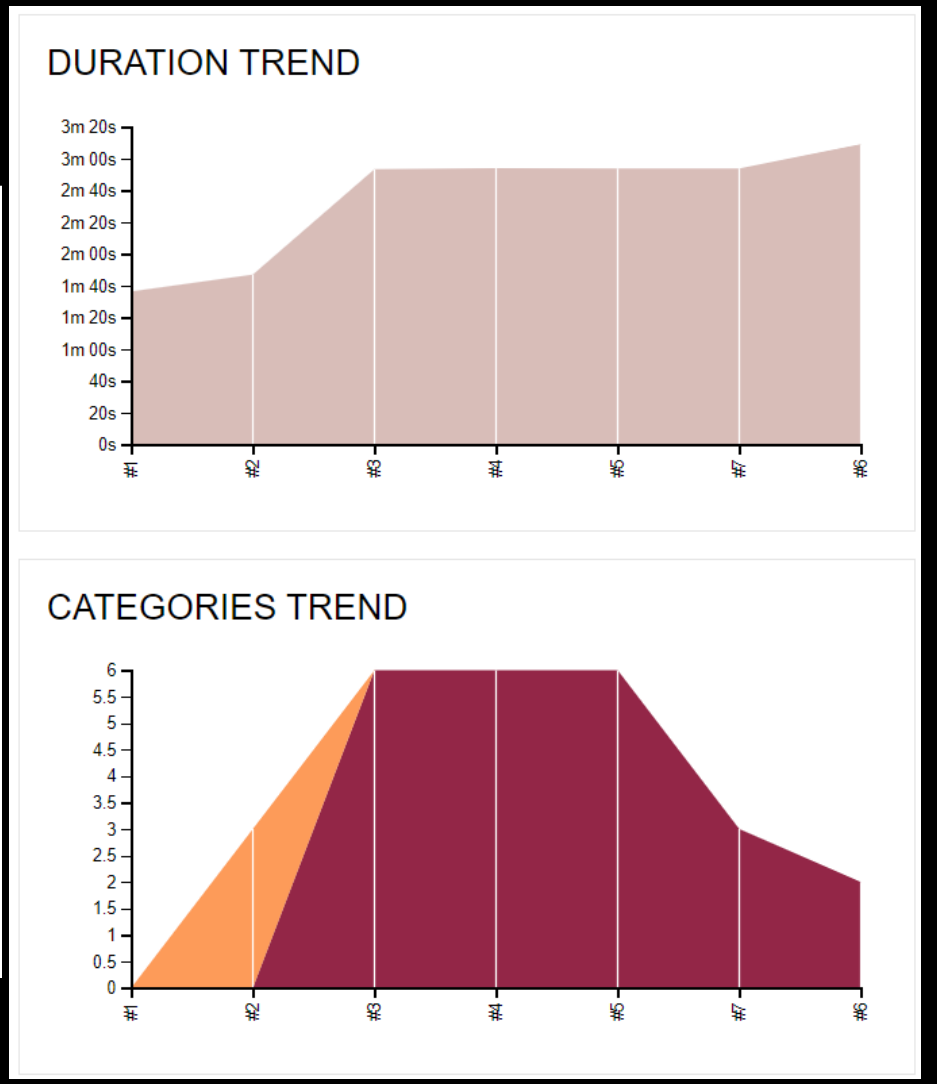
A QUICK ROUNTRIP THROUGH  ALLURE

TRENDS AND HISTORY

Allure can show results from previous runs

TRENDS AND HISTORY

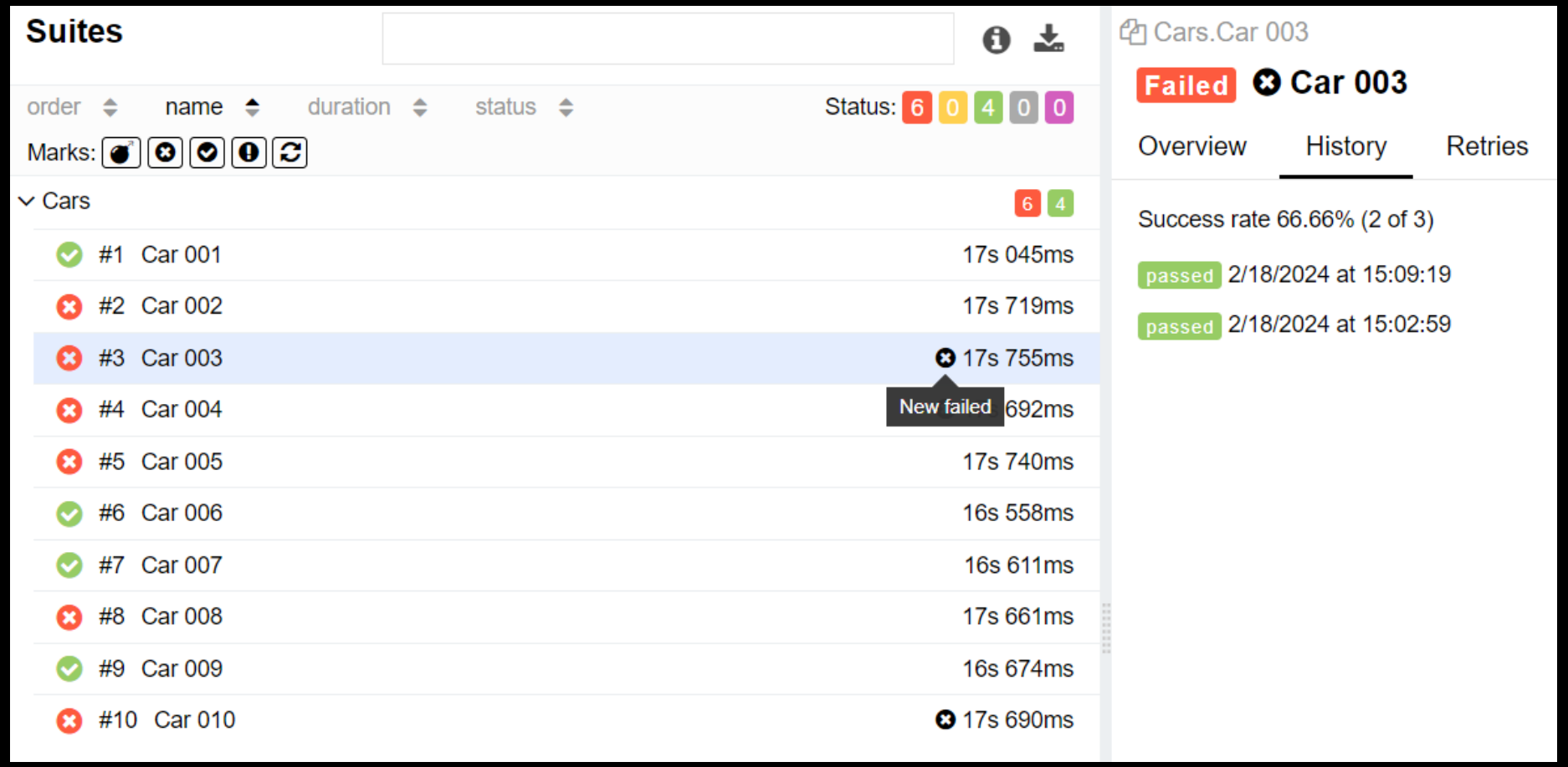
- Allure can show results from previous runs



RBCN24

TRENDS AND HISTORY

- It can also show the history per test



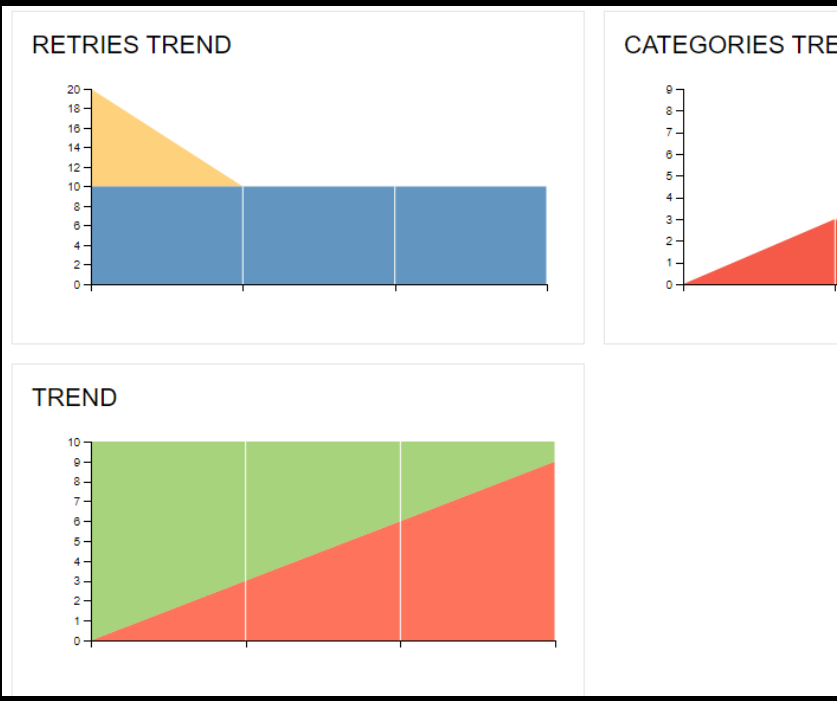
HOW TO SET UP HISTORY

- Every generated Allure Report contains a folder History

- Run your new Tests with Allure Listener

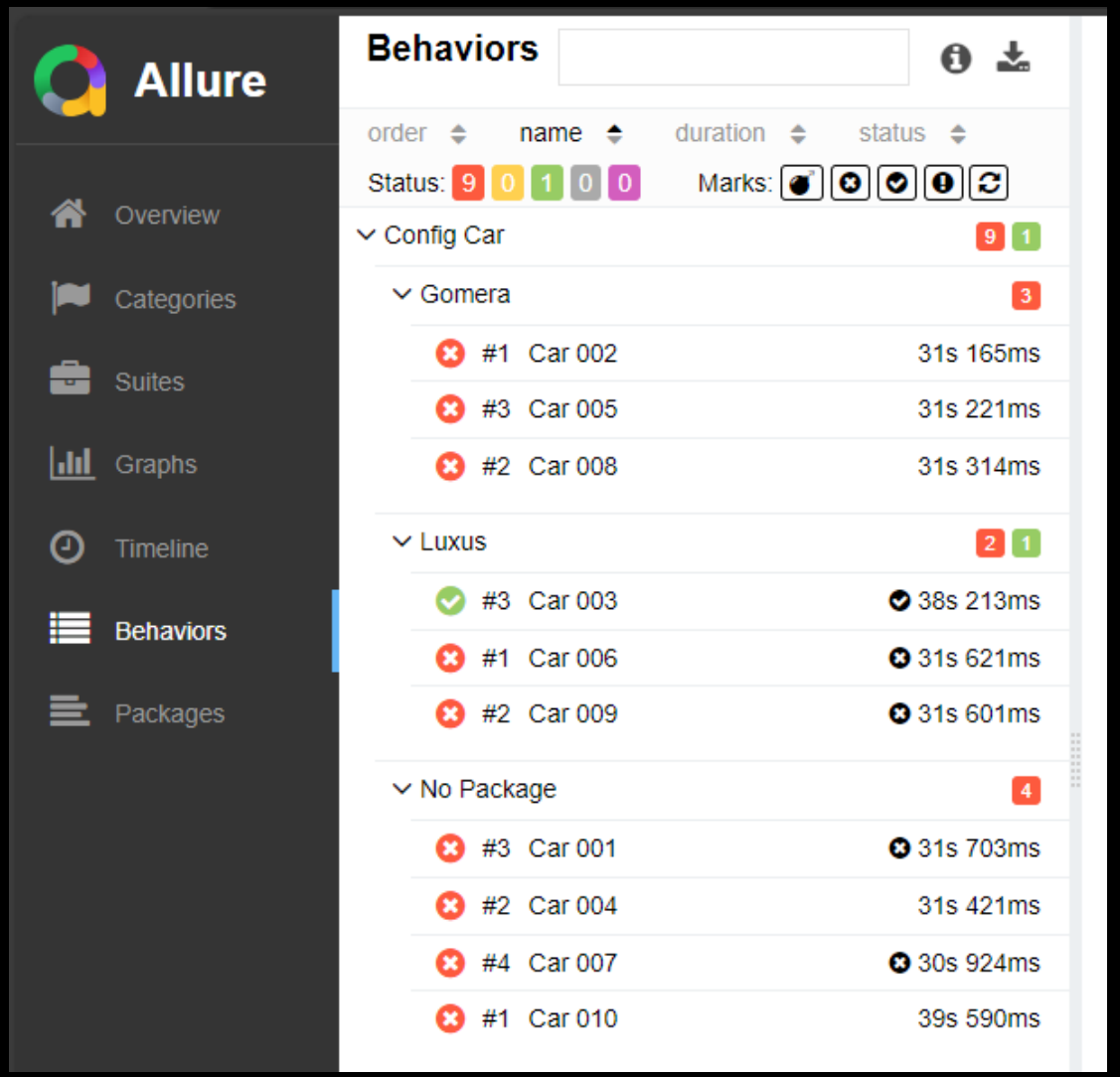
- Copy that History folder to your Allure Results, **before generating** the next report

- Generate new Allure Report



USE LABELS FOR STORIES/FEATURES

```
*** Settings ***
Library     Browser     timeout=30
Library     OperatingSystem
Library     String
Test Template     Configure Car
Suite Setup     New Browser     browser=chromium     headless=True
Suite Teardown     Close Browser
Test Tags  allure.feature:Config Car


*** Variables ***
${4SPACES}     ${SPACE}${SPACE}${SPACE}${SPACE}

*** Test Cases ***     {MODEL}=0     ${ENGINE}=0     ${PACKAGE}=0
Car 001     0     0     0     0     31.900,00 €
    [Tags]     allure.story:No Package
Car 002     0     1     1     1     32.823,00 €
    [Tags]     allure.story:Gomera
Car 003     0     2     2     2     36.498,99 €
    [Tags]     allure.story:Luxus
Car 004     0     3     0     3     31.246,00 €
    [Tags]     allure.story:No Package
```



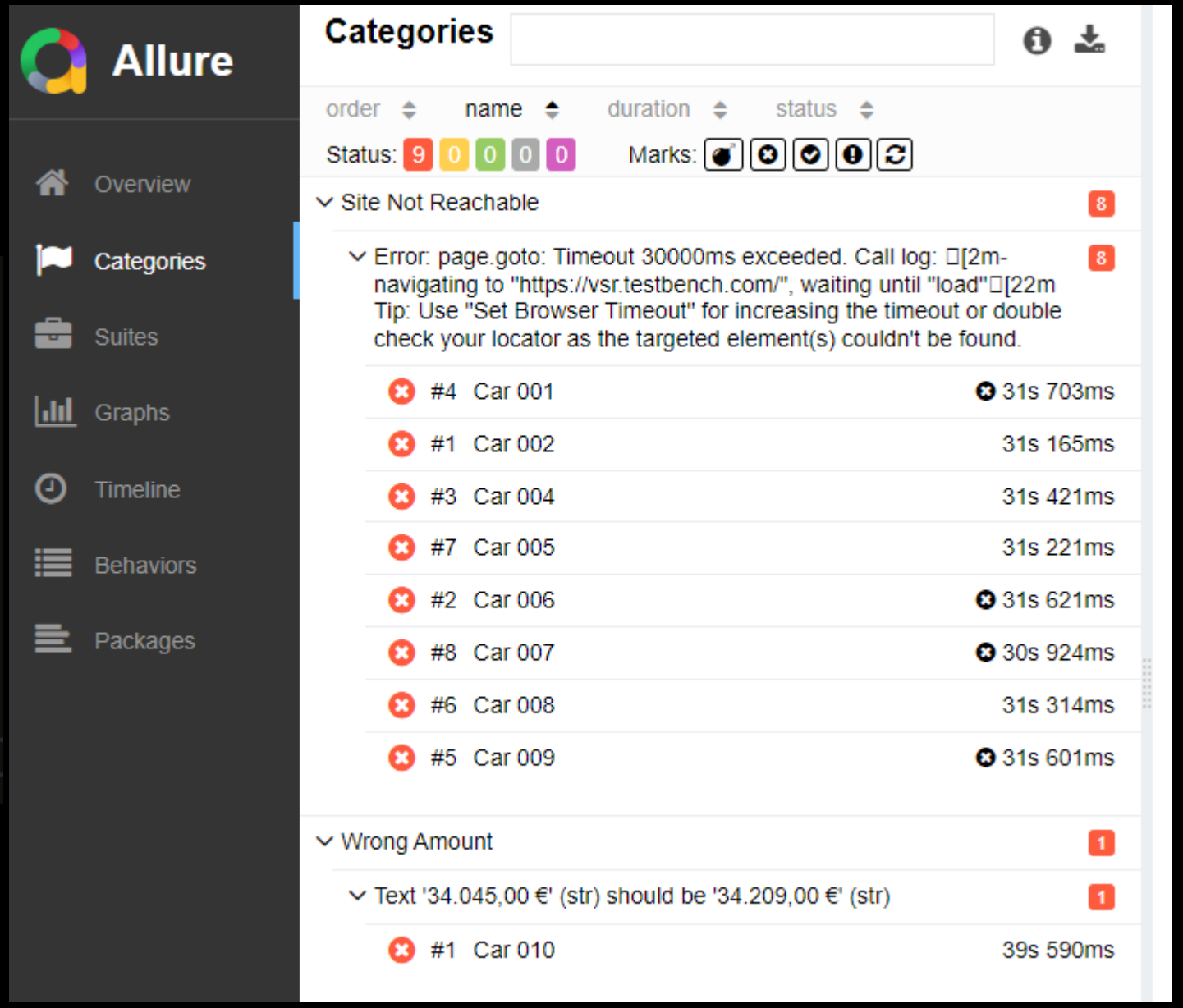
RBCN24

CATEGORIES VIA CATEGORIES.JSON

- Copy file into output/allure directory

03_allure > {} categories.json > ...

```
1  [
2      {
3          "name": "Wrong Amount",
4          "messageRegex": "Text.*should be.*",
5          "matchedStatuses": ["failed"]
6      },
7      {
8          "name": "Site Not Reachable",
9          "messageRegex": "Error:.*Timeout.*",
10         "matchedStatuses": ["failed"]
11     }
12 ]
```



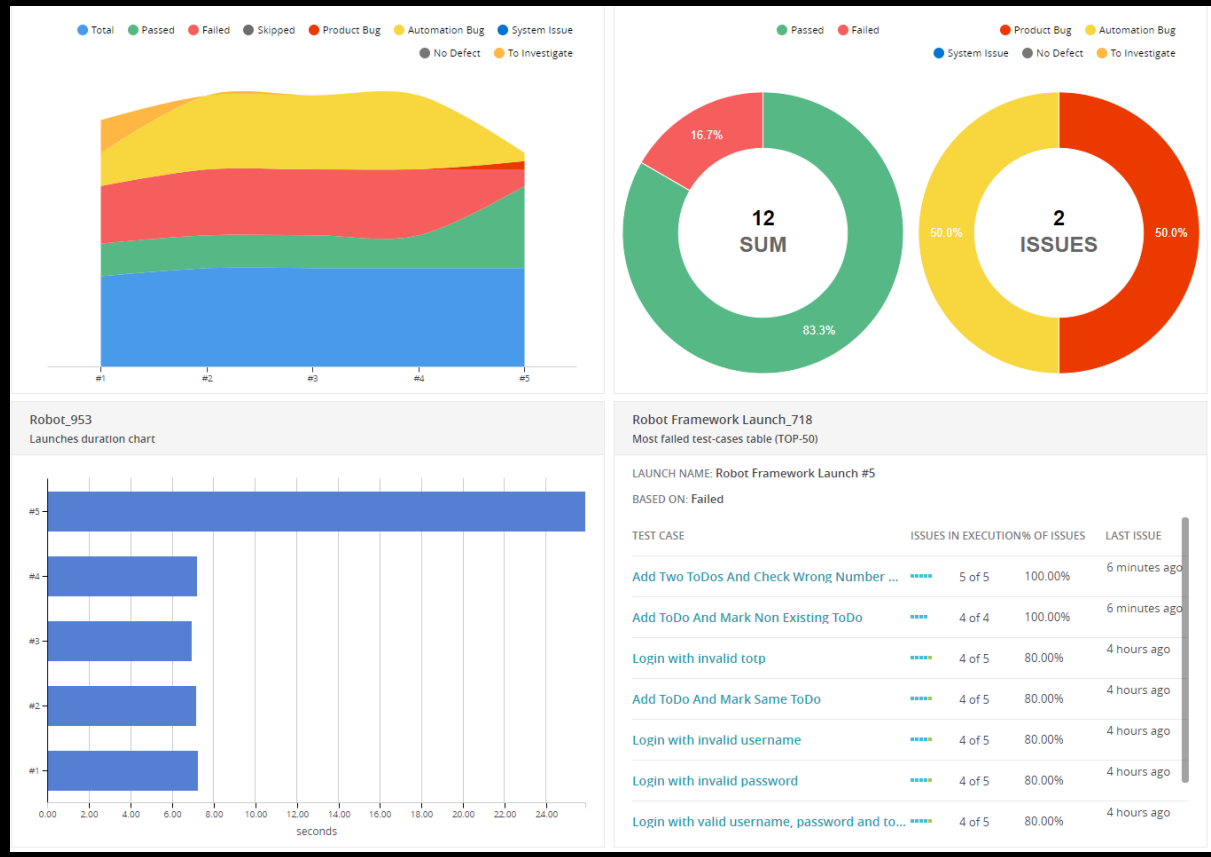
RBCN24

REPORT PORTAL

06_reportportal/

REPORTPORTAL FEATURES

- Customizable Dashboard
- Automated Failure analysis via AI and text pattern matching
- Unique Error analysis (10 errors or 1 error 10 times)
- RF Logs are fully available
- API exposed
- Integrated via Listener or robot api
- Self-hosted or SaaS



A QUICK ROUNTRIP THROUGH  REPORT PORTAL

GENERAL REPORTING HINTS

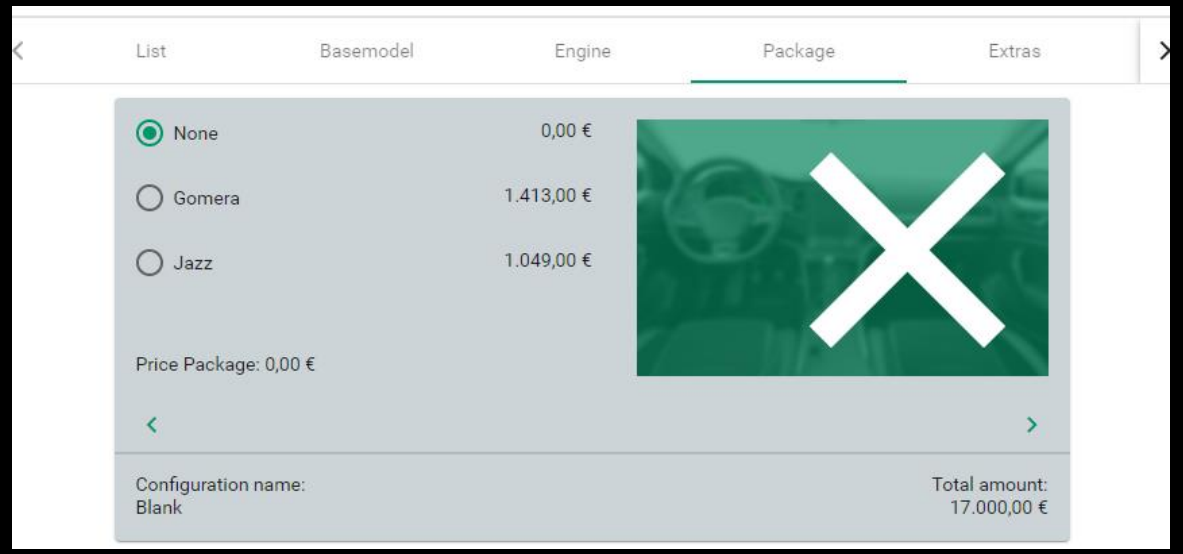
REVISIT THE STANDARD REPORTS AND DO SOME TWEAKS

- Use [TAGS] to
  - Categorize Tests
  - Categorize Failures
  - Mark known Defects
- Use --removekeywords to get rid of unneded log entries
- Use --flattenkeywords to reduce log tree structure
- Use xunit.xml reports for reporting to CI tools

ADD TAGS DURING RUNTIME TO CATEGORIZE TESTS

```
Select Package
    [Arguments]    ${PACKAGE}
    Click    mat-radio-button >> nth=${PACKAGE}    # 0-2
    Set Tag From Element    mat-radio-button.mat-radio-checked    Package
    Click    id=arrow_next
```

```
Set Tag From Element
    [Arguments]    ${locator}    ${prefix}
    ${value}    Get Text    ${locator}
    ${value}    Strip String    ${value}
    Set Tags    ${prefix}=${value}
```

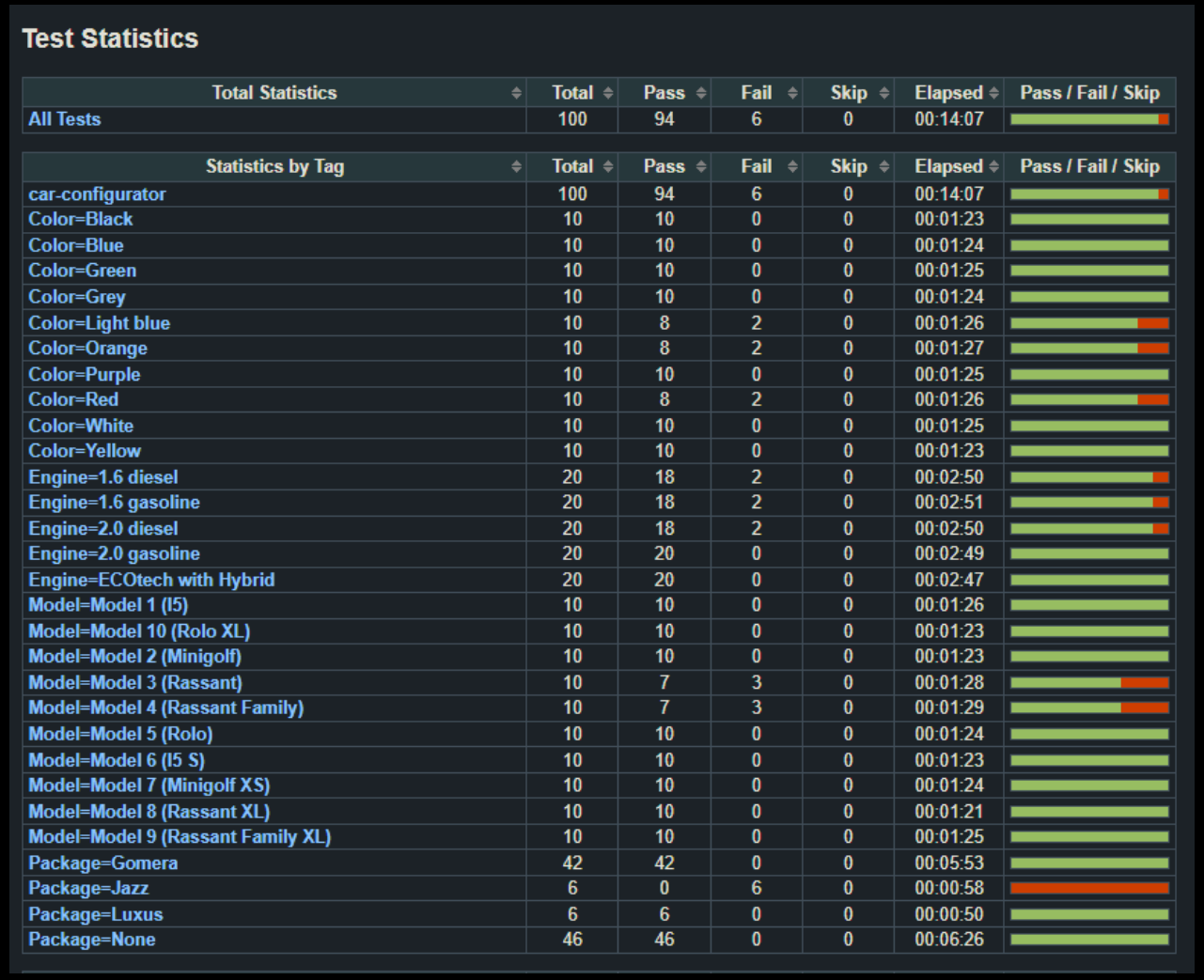


RBCN24

WHO IS THE TROUBLEMAKER?

6 out of 100 are failed

But why?



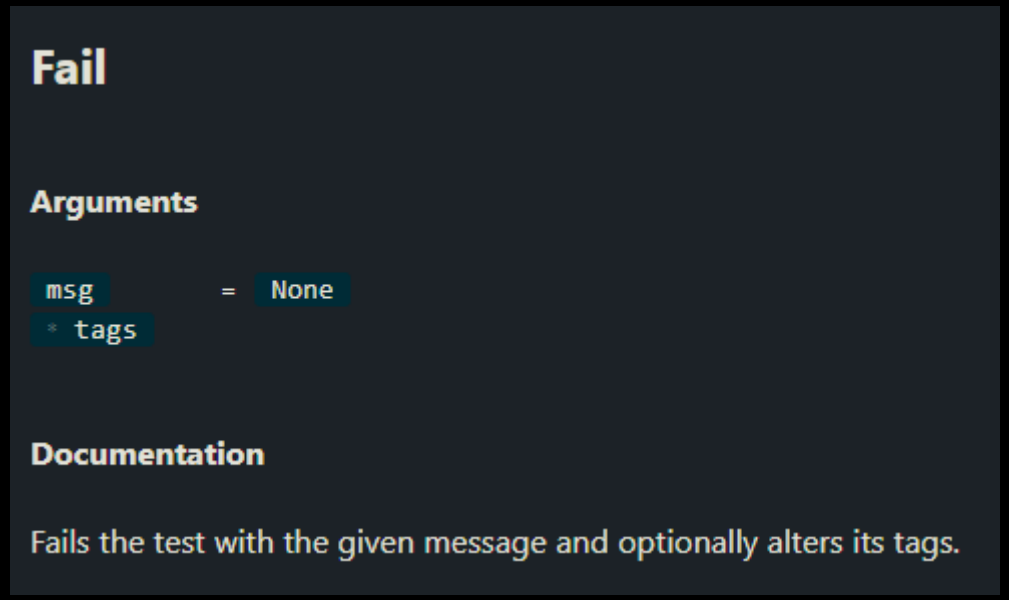
Test Statistics

Total Statistics	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
All Tests	100	94	6	0	00:14:07	

Statistics by Tag	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
car-configurator	100	94	6	0	00:14:07	
Color=Black	10	10	0	0	00:01:23	
Color=Blue	10	10	0	0	00:01:24	
Color=Green	10	10	0	0	00:01:25	
Color=Grey	10	10	0	0	00:01:24	
Color=Light blue	10	8	2	0	00:01:26	
Color=Orange	10	8	2	0	00:01:27	
Color=Purple	10	10	0	0	00:01:25	
Color=Red	10	8	2	0	00:01:26	
Color=White	10	10	0	0	00:01:25	
Color=Yellow	10	10	0	0	00:01:23	
Engine=1.6 diesel	20	18	2	0	00:02:50	
Engine=1.6 gasoline	20	18	2	0	00:02:51	
Engine=2.0 diesel	20	18	2	0	00:02:50	
Engine=2.0 gasoline	20	20	0	0	00:02:49	
Engine=ECOtech with Hybrid	20	20	0	0	00:02:47	
Model=Model 1 (I5)	10	10	0	0	00:01:26	
Model=Model 10 (Rolo XL)	10	10	0	0	00:01:23	
Model=Model 2 (Minigolf)	10	10	0	0	00:01:23	
Model=Model 3 (Rassant)	10	7	3	0	00:01:28	
Model=Model 4 (Rassant Family)	10	7	3	0	00:01:29	
Model=Model 5 (Rolo)	10	10	0	0	00:01:24	
Model=Model 6 (I5 S)	10	10	0	0	00:01:23	
Model=Model 7 (Minigolf XS)	10	10	0	0	00:01:24	
Model=Model 8 (Rassant XL)	10	10	0	0	00:01:21	
Model=Model 9 (Rassant Family XL)	10	10	0	0	00:01:25	
Package=Gomera	42	42	0	0	00:05:53	
Package=Jazz	6	0	6	0	00:00:58	
Package=Luxus	6	6	0	0	00:00:50	
Package=None	46	46	0	0	00:06:26	

KNOW YOUR ENEMIES
TAG FOR DEFECT TYPES

- Use the **Fail** Keyword to modify [Tags] in case of failure




```
TRY
    Check Price    ${PRICE}
EXCEPT    .*should be.*    AS    ${error}
    Fail    {error}    Wrong Price
END
```

- TAG for linked Defect
- TAG to skip tests
- Add Link for TAG
- TAG error types, e.g. via TRY/EXCEPT and FAIL

REDUCE OUTPUT.XML SIZE

REPROCESS YOUR RESULTS WITH REBOT

- Use --removekeywords to get rid of unneded log entries
  - Remove PASSED Tests Log Details

- Use --flattenkeywords to reduce log tree structure
  - Flatten deeply nested Keywords with many layers

```
rebot --removekeywords passed --output removed.xml output.xml
robot --removekeywords passed tests.robot
```

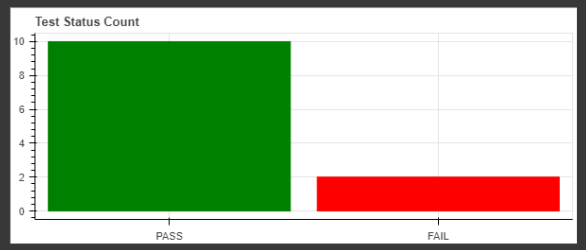
```
rebot --flattenkeywords name:HugeKeyword --output flattened.xml output.xml
robot --flattenkeywords name:HugeKeyword tests.robot
```

LOOK INTO PANDAS AND BOKEH

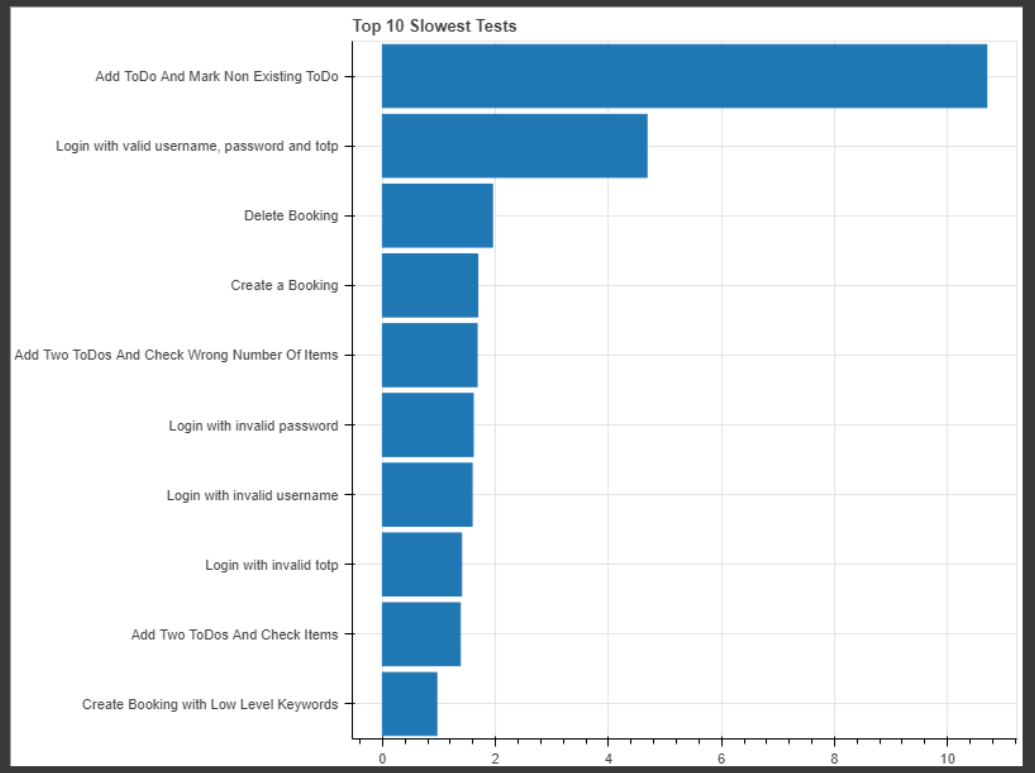
(or other packages for data analysis and visualization)

CHECK OUT PANDAS AND BOKEH
FIND EXAMPLES IN MY NOTEBOOK

```
[18] # Create a bar chart from the df and output it to a html using bokeh
     # Count the number of tests per status and create a bar chart
     # PASS shall be green, FAIL shall be red, SKIP shall be yellow

     from bokeh.plotting import figure
     from bokeh.io import output_file, show
     from bokeh.models import ColumnDataSource

     # Count the number of tests per status
     test_status_count = testresults_df['Test Status'].value_counts()

     # Create a bar chart
     output_file("results/bokeh_bar_chart.html")

     teststatus_bar = figure(x_range=test_status_count.index.tolist(), height=250, title="Test Status Count",
                 toolbar_location=None, tools="")

     # Set the color of the bars
     # PASS shall be green, FAIL shall be red, SKIP shall be yellow

     teststatus_bar.vbar(x=test_status_count.index[test_status_count.index == 'PASS'].tolist(), top=test_status_count[test_status_count.index ==
     teststatus_bar.vbar(x=test_status_count.index[test_status_count.index == 'FAIL'].tolist(), top=test_status_count[test_status_count.index ==
     teststatus_bar.vbar(x=test_status_count.index[test_status_count.index == 'SKIP'].tolist(), top=test_status_count[test_status_count.index ==

     output_notebook()
     show(teststatus_bar)
```




```
# Sort the df by Test Elapsed Time in descending order
testresults_df = testresults_df.sort_values(by='Test Elapsed Time', ascending=False)


slowest_tests_bar = figure(y_range=testresults_df['Test Name'].head(10)[::-1], width=800, title="Top 10 Slowest Tests",
                           toolbar_location=None, tools="")
slowest_tests_bar.hbar(y=testresults_df['Test Name'].head(10)[::-1], right=testresults_df['Test Elapsed Time'].head(10)[::-1], height=0.9)

show(slowest_tests_bar)
```



RBCN24

SOME OTHER REPORTING TOOLS AND OTHER RF ECOSYSTEM PROJECTS

- RF Report Modifier
  https://github.com/MarketSquare/robotframework-reportmodifier
  (Presented at RoboCon 2024, Helsinki)
- RF xunit modifier
  https://github.com/rikerfi/robotframework-xunitmodifier
- RF GH Action Reporter
  https://github.com/rasjani/robotframework-ghareports
- Testarchiver
  https://github.com/salabs/TestArchiver
- Robot Framework Metrics
  https://github.com/adiralashiva8/robotframework-metrics
- Robot Framework Jenkins Plugin
  https://plugins.jenkins.io/robot/

CHECK OUT ROBOCON 2024 TALKS ABOUT REPORTING AND VISUALIZATION

- Beyond log files: Elevating test analysis with data visualization
- Robot Framework as Compliance Enabler
- Test reporting@UnionInvestment

RBCN24

THANKS!
QUESTIONS?

Many Kasiriha

@Many
Join #reporting