Bachelor Thesis

Test Result Viewer

Semester: Spring 2023

Version: 1.0
June 14, 2023

Students: Olivier Lischer
Luzia Kündig

Advisor: Thomas Corbat

School of Computer Science
OST - Eastern Switzerland University of Applied Sciences

Abstract

Sonova AG is a company based in Stäfa, Switzerland, and is known for the development of high quality hearing aids and other audio-related products. Their development division currently uses several custom made tools for the visualization of automated test results. They suffer from various deficiencies and should be replaced by one single platform. In existing off-the-shelf solutions it is often the assumption that detailed information about test results is only relevant if there were any errors during execution. In case of a successful run, no details are presented to a developer except the final result "all passed". Therefore, it has been decided that a new and custom platform should be built to replace the existing tools.

This platform receives test result files in different XML-based formats along with separate JSON metadata, which will be provided by the current infrastructure at Sonova. The Data Consumer component parses and transformes these files into a unified schema. To create such a schema required the definition of what information is relevant for the visualizations. The results are then saved to a central data store and displayed by a frontend application. Elaborate requirements of what visualizations the platform should support have been provided by Sonova in advance. We have evaluated different tools for the components and according to our own research, some recommendations and experience present at Sonova, we chose Python to implement the Data Consumer, Elasticsearch to store the data and Angular to implement the frontend.

The resulting product of this project serves as a prototype kind of implementation to be extended by Sonova in the future. The platform we have created contains two main visualizations as well as additional pages that display important metadata. It focuses on extensibility so that it supports implementing new requirements in existing as well as additional views.

Management Summary

Problem description

Software tests are an important tool to improve software quality and robustness when changes have to be made. In most software projects, automated tests can be run locally on the developer's machine or centrally using continuous integration, for example when pushing changes to a central repository.

However, these benefits can only be leveraged if the results of these tests are presented in a way that actively supports the development process.

In existing solutions where test results are represented graphically, it is often the assumption that test results are only relevant if something went wrong. In case of a successful execution, no details whatsoever are presented to a developer except the final result "all passed".

In the particular case at Sonova AG, running and evaluating tests is more complex than this. There are tests that depend on specific hardware to run, tests that run up to 100 times in one test execution, tests that only run during the night or on weekends because of their duration.

All these different combinations require the use of a more sophisticated tool to quickly grasp and evaluate the current status of a given codebase. Currentyly, several tools try to address this problem (see Figure 1 for an example), however the information is distributed between them, they are hard to maintain and do not handle new requirements well. The goal of this thesis is to create a single tool that combines all test results and displays them in a way that is relevant for the intended users.

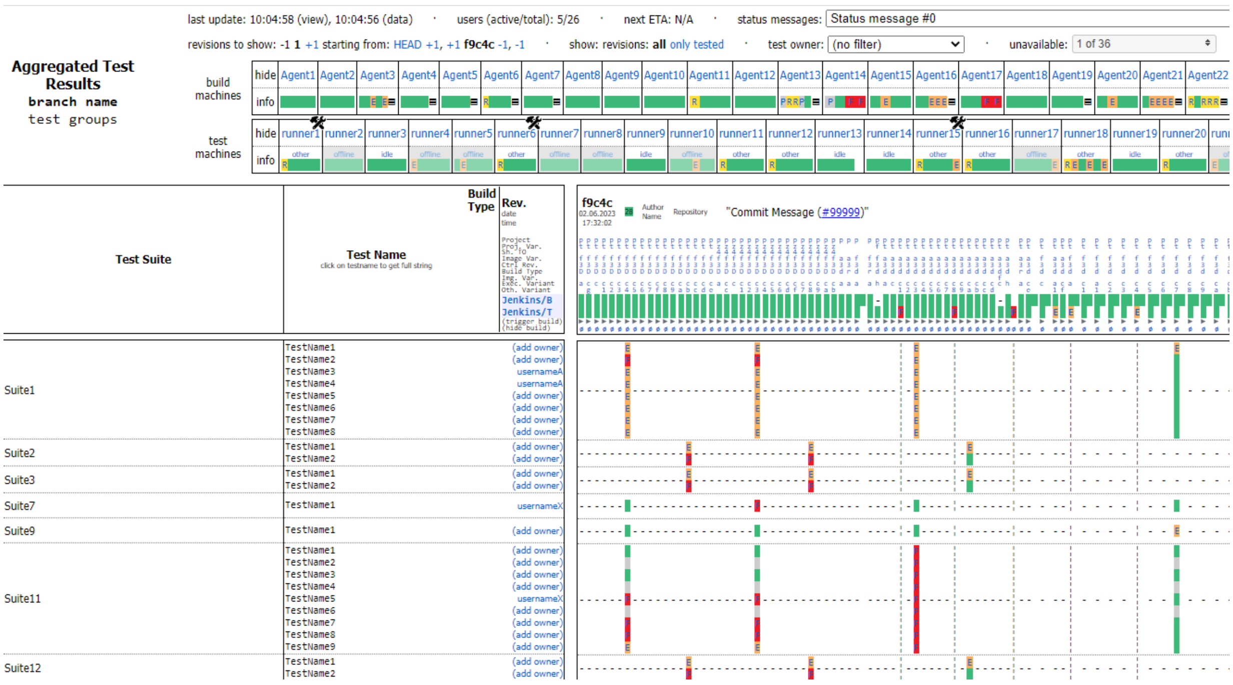

last update: 10:04:58 (view), 10:04:56 (data)
users (active/total): 5/26
next ETA: N/A
status messages: Status message #0
revisions to show: -1 1 +1 starting from: HEAD +1, +1 f9c4c -1, -1
show: revisions: all only tested
test owner: (no filter)
unavailable: 1 of 36
Aggregated Test Results
branch name
test groups
build machines
test machines
hide
info
Agent1 Agent2 Agent3 Agent4 Agent5 Agent6 Agent7 Agent8 Agent9 Agent10 Agent11 Agent12 Agent13 Agent14 Agent15 Agent16 Agent17 Agent18 Agent19 Agent20 Agent21 Agent22
runner1 runner2 runner3 runner4 runner5 runner6 runner7 runner8 runner9 runner10 runner11 runner12 runner13 runner14 runner15 runner16 runner17 runner18 runner19 runner20
Test Suite
Test Name
Build Type
Rev.
f9c4c
02.06.2023
17:32:02
"Commit Message (#99999)"
Suite1
Suite2
Suite3
Suite7
Suite9
Suite11
Suite12
TestName1
(add owner)
usernameA
usernameX

Figure 1.: Screenshot of one existing tool

Preconditions and approach

When developing this application for an existing ecosystem at Sonova, we were presented with some recommendations about technologies to use. Since the final result should integrate into their environment as seamlessly as possible, we often prioritized the products they use over othes. Still we conducted a short analysis of possible alternatives for each component and made our own recommendations if we found another solution to be more suitable.

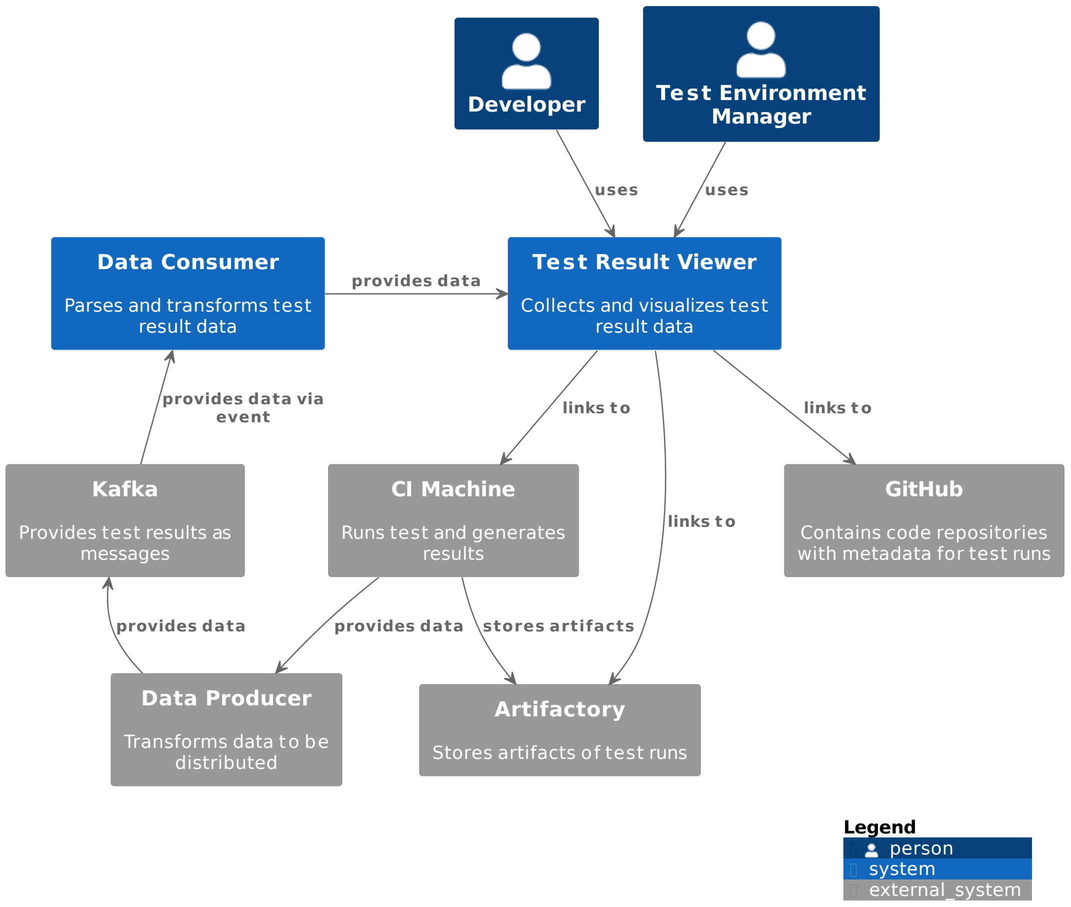

Developer
Test Environment Manager
uses
uses
Data Consumer
Parses and transforms test result data
provides data
Test Result Viewer
Collects and visualizes test result data
provides data via event
links to
links to
links to
Kafka
Provides test results as messages
CI Machine
Runs test and generates results
GitHub
Contains code repositories with metadata for test runs
provides data
provides data
stores artifacts
Data Producer
Transforms data to be distributed
Artifactory
Stores artifacts of test runs
Legend
person
system
external_system

Figure 2.: Architecture Goal

Test results in different formats along with relevant metadata in a seperate file will be produced by the current infrastructure at Sonova. These are the systems colored in grey in Figure 2. The results and metadata should be received, parsed and transformed into a unified schema using the Data Consumer component. Then the data is saved and displayed by the Test Result Viewer. According to our research, some recommendations and experience present at Sonova we chose Python for the Data Consumer and Elasticsearch in combination with Angular to implement the Test Result Viewer.

Implementation

The main focus for the realization of this project is parsing the input files to create a database schema as well as the creation of some basic frontend visualizations. The final implementation of the product and its integration with several other systems at Sonova is presented in Figure 2. Figure 3 again shows the flow of data and the extent of this thesis.

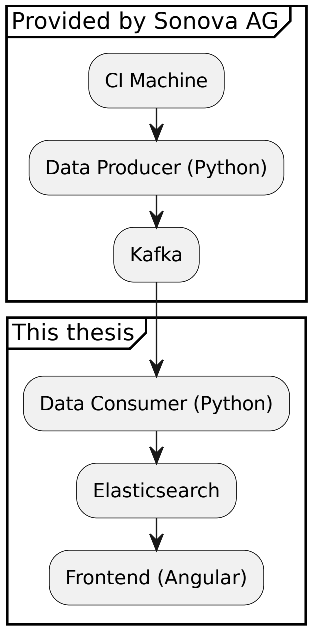


Figure 3.: General flow of data

Database Schema

The first set of sample data we received from Sonova follows the NUnit test result format [41], therefore we implemented a parser for this format first. As the tool should support different kinds of result formats, we focused on creating a modular structure. This was helpful when adding the functionality of generating random test results to add some load to the Test Result Viewer. Later in the project we added support for parsing JUnit test results as well.

In the database schema of a test result we used the fields necessary to display all relevant information of a result, together with an additional field to distinguish its format.

Frontend Visualizations

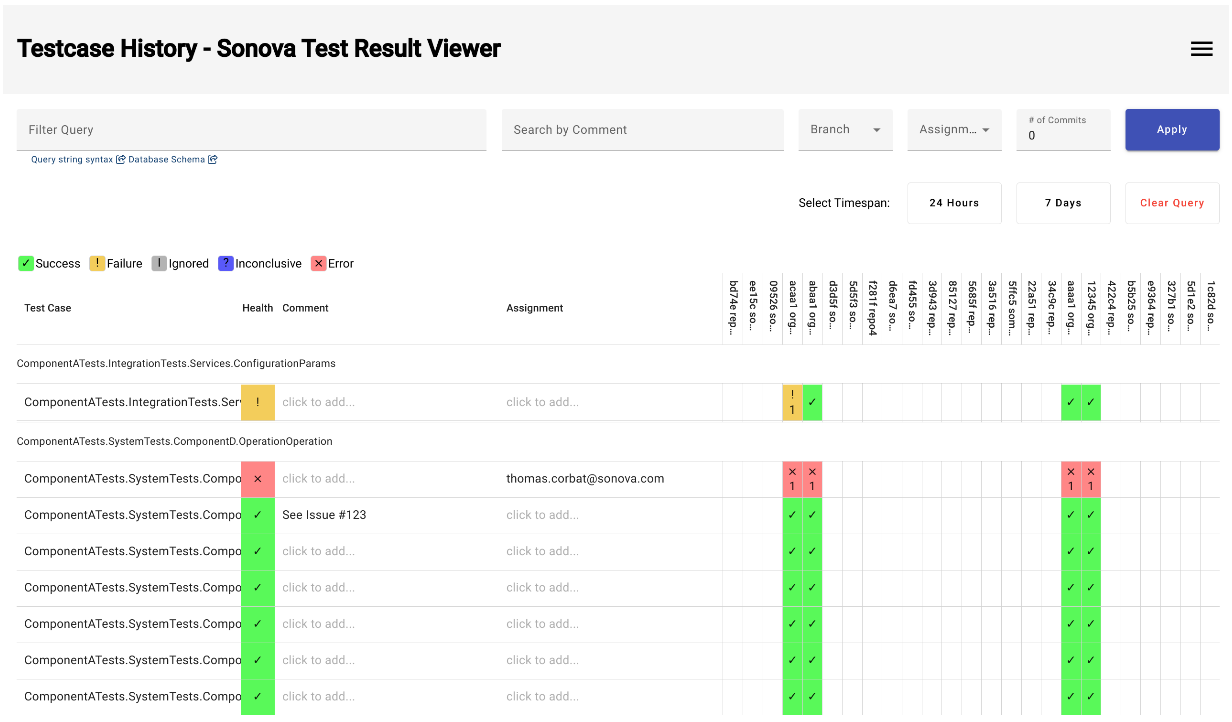
To create the visualizations that Sonova requires, we analyzed the data structure and created complex transformations. The addition of comments and assignees had to be supported by the data structure in Elasticsearch as well.

We implemented different views for the two different user personas; the developer and the test environment manager. These views combine multiple use cases and offer filtering based on the common query language KQL, also used by Kibana.

Conclusion

The database schema we created largely depends on the data that is available in the different result formats and the data that is needed by the users of this tool. By using a separate file to provide metadata it is be easy to add more information in the future. The visualizations we created require good knowledge of the Elasticsearch querying tools and response formats. The abstraction of these specifics inside the frontend was another focus we set for ourselves. Figure 4 shows the view created for the developer user persona.

The current state of the implementation allows developers at Sonova to use it productively, but since it acts as a prototype many improvements can be made in the future, as demonstrated by the extensive list of requirements.



Testcase History - Sonova Test Result Viewer

Filter Query | Search by Comment | Branch | Assignm... | # of Commits 0 | Apply

Query string syntax · Database Schema

Select Timespan: 24 Hours | 7 Days | Clear Query

✓ Success ! Failure I Ignored ? Inconclusive × Error

Test Case	Health	Comment	Assignment	bd74e rep...	ee15c so...	09526 so...	acaa1 org...	abaa1 org...	d3d5f so...	5d5f3 so...	f281f repo4	d6ea7 so...	fd455 so...	3d943 rep...	85127 rep...	5685f rep...	3a516 rep...	5ffc5 som...	22a51 rep...	34c9c rep...	aaaa1 org...	12345 org...	422c4 rep...	b5b25 so...	e9364 rep...	327b1 so...	5d1e2 so...	1c82d so...
ComponentATests.IntegrationTests.Services.ConfigurationParams																												
ComponentATests.IntegrationTests.Ser	!	click to add...	click to add...				! 1	✓													✓	✓						
ComponentATests.SystemTests.ComponentD.OperationOperation																												
ComponentATests.SystemTests.Compo	×	click to add...	thomas.corbat@sonova.com				× 1	× 1													× 1	× 1						
ComponentATests.SystemTests.Compo	✓	See Issue #123	click to add...				✓	✓													✓	✓						
ComponentATests.SystemTests.Compo	✓	click to add...	click to add...				✓	✓													✓	✓						
ComponentATests.SystemTests.Compo	✓	click to add...	click to add...				✓	✓													✓	✓						
ComponentATests.SystemTests.Compo	✓	click to add...	click to add...				✓	✓													✓	✓						
ComponentATests.SystemTests.Compo	✓	click to add...	click to add...				✓	✓													✓	✓						
ComponentATests.SystemTests.Compo	✓	click to add...	click to add...				✓	✓													✓	✓						

Figure 4.: Screenshot Test Result Viewer

Acknowledgements

We have given a lot of effort to this project over its entire duration. However, without the help of others it would have not been possible to achieve as much as we have done. This is why we want to express our sincere gratitude to

- **Thomas Corbat**, our Thesis Advisor at OST and at the same time a future user of our product in his role as Embedded Software Developer at Sonova
- **Dariusz Danilko**, Product Owner and a future user of our product at Sonova

for their constant and detailed feedback on the progress of our work, support in any questions that we had along the way and their patient explanations about the situation and problems that we have attempted to solve.

To Sonova as a company we are grateful for the opportunity to work on a project in the industry that has very good chances of being adopted after the conclusion of this thesis.

No less important, we have been lucky to count on the support of

- **Thomas Kälin, Marco Agostini, Dominik Ehrle, Jan Untersander, Tsigereda Nebai Kidane and Stefan Morf**, our lecturer, peer students and friends for their valuable time and knowledge in giving feedback regarding our documentation and product and challenging our ideas to become even better. Thank you.

And of course the patience and understanding of our friends and families when we were unavailable for yet another day. :-)

Contents

Part I.

Product Documentation

1. Vision

Before a daily stand-up meeting at Sonova can start, each developer has to gather all test results from various sources and check the reason for any potential failures. In a central platform to gather and display test result data, the primary focus of the development team would be the current status and the history of all test cases, as well as the potential introduction of regressions.

The infrastructure team on the other hand has to monitor and maintain multiple test environments, comprised of numerous different hardware test devices and servers. Some devices have a limited lifetime, as for example the number of read and write operations on the memory is limited. so it has to be replaced after a certain amount of time. For this infrastructure team any graphical overview should focus on the executions per hardware, regardless of which test case was executed.

Many Continuous Integration (CI) systems provide some kind of test result visualizations out of the box. However most of them, if not all, are not capable of visualizing thousands of tests results and attaching some metadata to each test run.

It is the goal of this bachelor thesis to create a platform consisting of the following components:

- **Data Consumer:** Parses and inserts test data into a data store
- **Data Store:** Stores processed test data
- **Frontend:** Displays and visualizes test results

This platform should support the developers and infrastructure maintainers in getting an overview over the current situation of their respective field of interest. It should also allow them to attach additional metadata like references to items on other related systems, including bug reports or build jobs.

2. Introduction

In this chapter we briefly introduce Sonova AG, our partner company for the project. Section 2.1 describes the current situation and section 2.2 the problems they are facing.

2.1. Current Situation

On their website, Sonova provides the following introduction:

> *Sonova is a global leader in innovative hearing care solutions: from personal audio devices and wireless communication systems to audiological care services, hearing aids and cochlear implants. The Group was founded in 1947 and is headquartered in Stäfa, Switzerland [44].*

Test Execution

To run and support Sonova's high quality hardware products, teams of developers create the necessary firmware. To ensure software quality, there are many automated tests that are executed on a regular basis.

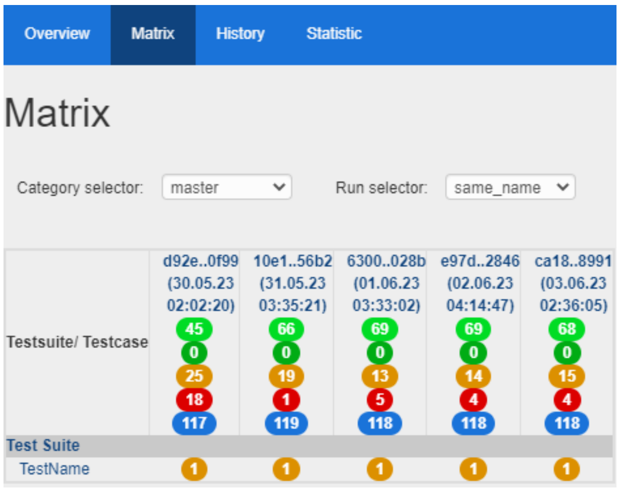
Ideally, code commits and the related test executions would be close together in time, with every test run being finished before a new commit is made. For Sonova, this is not possible since there are three different schedules for tests. They can be run *per commit*, which would be the ideal case, *nightly* and *on weekends*. The reason for this is the complexity of the hardware devices and also their related systems. A regular hearing aid consists of several internal components like CPU, Digital Signal Processors (DSPs), power management chip and wireless communication chip. Additionally, for each product the tests have to be executed with different configurations, like the execution on development versions of the real hardware or on Field Programmable Gate Arrays (FPGAs).

Daily Scrum Meetings

In an agile software development style when using the well known Scrum framework, there is a short meeting every morning where every developer communicates their current status of work to the group. This is called the daily scrum. To prepare for this meeting, a detailed overview of the previous test runs is necessary to see their current status and see if some previous work might have had a negative impact [49].

Existing Tools

Many CI systems display the results of a test job with minimal information only, e.g. success or failure. This seems to be the standard and generally enough for a quick overview. Sonova tried to address their special requirements regarding test results in the past and created several tools to display more detailed information.



Matrix

Category selector: master

Run selector: same_name

Testsuite/ Testcase	d92e..0f99 (30.05.23 02:02:20)	10e1..56b2 (31.05.23 03:35:21)	6300..028b (01.06.23 03:33:02)	e97d..2846 (02.06.23 04:14:47)	ca18..8991 (03.06.23 02:36:05)
	45	66	69	69	68
	0	0	0	0	0
	25	19	13	14	15
	18	1	5	4	4
	117	119	118	118	118
Test Suite					
TestName	1	1	1	1	1

Figure 2.1.: Screenshot of an existing tool with minimal information

2.2. Problem

With the situation described previously, a developers daily routine would include checking many different dashboards and job executions to get a complete overview of the current status of all test runs.

As stated by our advisors at Sonova, their existing tools suffer from various deficiencies and need replacement. The main problems of the current solutions are the following.

Loading time is often too long. In some tools the test results are stored in XML files, which might create this problem. What other factors could affect loading times has not been analyzed in detail.

The code base is badly maintained. Database and viewer component of these tools were custom implementations instead of established third party tools. These components were created by developers that are mainly required to write embedded code. The time available for maintaining and extending such internal projects is limited.

The tools are hard to change and extend. As the technologies used have aged, fewer developers know and are able to support them.

2.3. Expected Results

The problems described in the previous section lead to the task we have been set for this thesis:

Our goal is to create a new platform based on existing frameworks that allows a complete view of the recent test results.

It should address all issues mentioned of the old solution while leveraging the functionality of modern tools for storing and visualizing the data.

2.4. Conditions

This project is realized as a bachelors' thesis at the OST - Eastern Switzerland University of Applied Sciences over the course of one semester. The duration of the project is from the 20 of

February to the 16 of June. It includes a budget of 360h per student which will result in 12 ECTS points each.

3. Requirements

This chapter describes user personas, functional and non-functional requirements that have been defined together with our advisors at Sonova. Additionally, it contains the description of a Minimum Viable Product (MVP) as well as our prioritization of further use cases in the scope of this thesis and the reasoning behind it.

3.1. User Personas

The following user personas were provided by Sonova as part of the initial project description. The platform we built targets two main groups of users, th developer and the test environment manager.

3.1.1. Developer

The user interested in test results. Configures, schedules and/or starts test runs, analyzes and reports test results, acts on failures by updating the source code in question.

3.1.2. Test Environment Manager

The user managing the hardware present for tests to be run on. He is responsible that the test environment is ready and able to run the tests. He addresses any problems such as low disk space or general performance issues by updating or replacing the hardware as needed.

3.2. Functional Requirements: Use Cases

At the start of this project, Sonova has provided an extensive list of functional requirements (see Appendix A). It is a collection of ideas as to what the tool could be used for in the future. All the requirements together far exceed the scope of what is possible to implement in this project. For this reason, a shorter list of main use cases has been provided as well (Appendix B).

These main use cases describe what is expected for the product to be capable of in order to be adopted by the customer and possibly extended in the future. Based on this, we created our own list with a prioritization and definition of the MVP (Table 3.1) and confirmed them in our weekly meetings with Sonova.

The **Prototype** priority does not directly apply to requirements by Sonova but serves as examples to verify different functionality of the tools used and for the developers to get used to applying them.

The components mentioned that implement certain use cases always refer to the specific implementation inside the Angular application.

Use Case	Priority	Completed
Database schema must support at least NUnit and Junit (3.2.3)	very high	W03
Overview page of last test run results (Figure 3.1)	Prototype	W04
Overview of all test case names	Prototype	W04
Overview of test environments grouped by result (3.7)	Prototype	W04
Tabular view of commits and test cases	Prototype	W05
Test result of a single test case at a specific time (3.2.5)	very high	W07
History of test results from a single test case (3.2.6)	very high	W07
History of test results on a specific test environment (3.2.7)	very high	W07
Assigning a label to a test result (3.2.8)	very high	W08
Hierarchical test order (3.2.9)	high	W09
Detection of regression of a particular test case (3.2.10)	middle	W07 (partially)
Detection of flickering tests (3.2.11)	low	W07
Detection of test environment problems (3.2.12)	low	-
Detection of similar/related test failures (3.2.13)	very low	-
Attach a comment to a test case (3.2.14)	middle	W12
Assign a Developer to a test case (3.2.8)	middle	W13

Table 3.1.: Use Cases with Priority

3.2.1. Minimum Viable Product

All use cases with a priority of **very high** make up the MVP.

3.2.2. Mockups

We have created different visualization mockups for some requirements at the beginning of the project. The facts they represent are not guaranteed to be accurate for the final state of the product. Figure 3.1 presents a Mockup there was no requirement for, but creating the page served as a starting point to get familiar with the technologies.

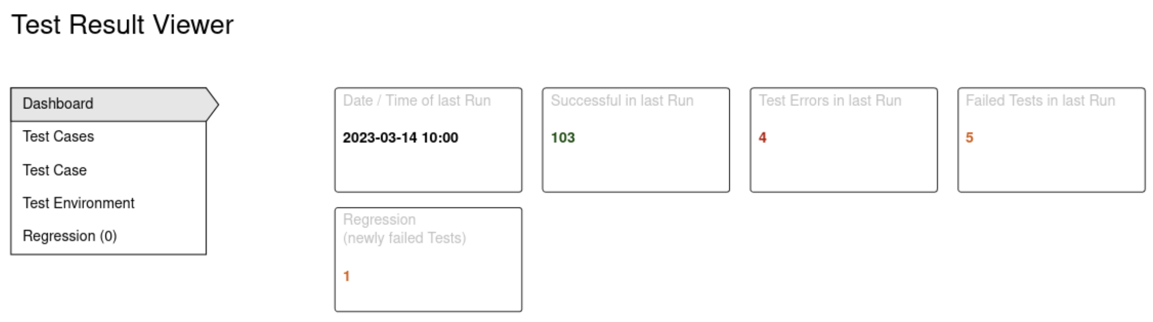


Figure 3.1.: Mockup: Dashboard page

3.2.3. Database schema must support at least NUnit and Junit

- **As a** developer
- **I want to** import the test results from various testing frameworks
- **so that** I can use the same platform to display different formats of test results.

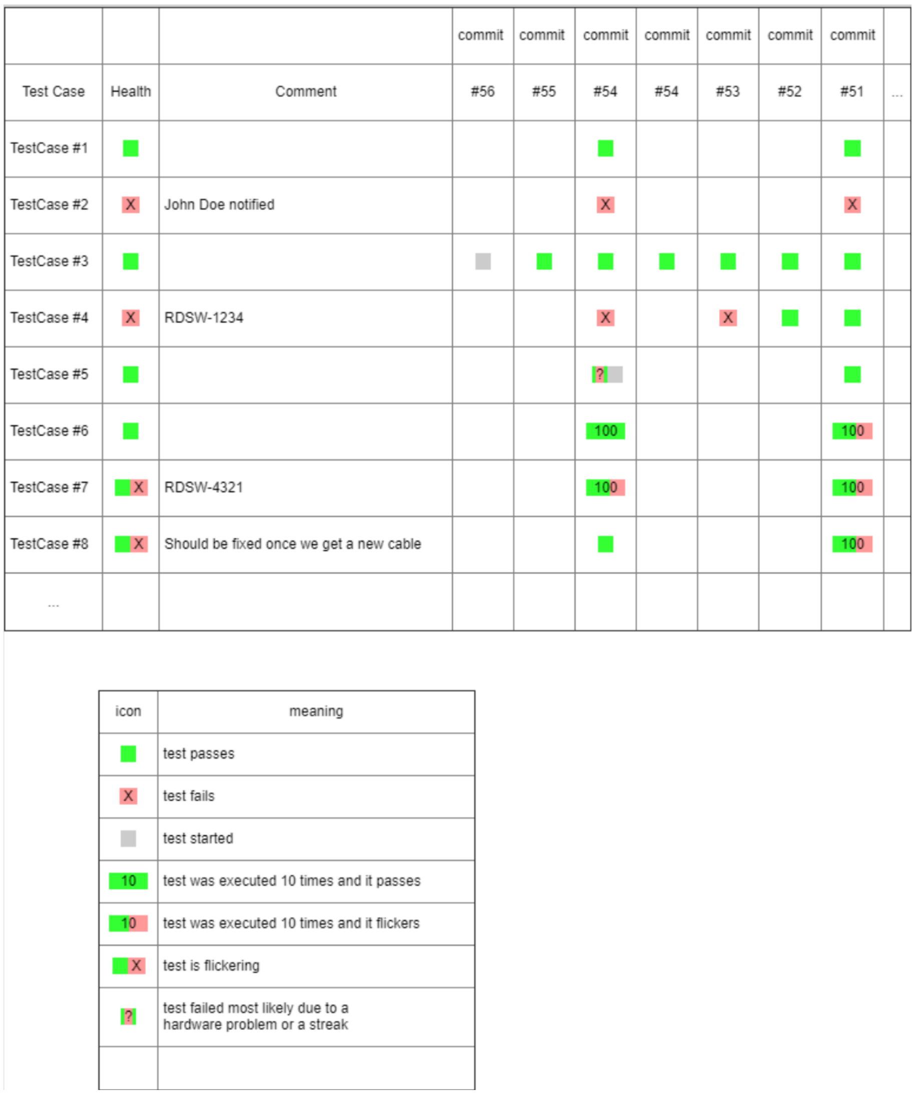
This use case is implemented in the consumer and the structure of the database schema.

3.2.4. Tabular view of commits and test cases

This use case has been taken from a sample UI sketch provided by Sonova (Figure 3.2) and should give direction for a view that supports several other use cases.

- **As a** developer
- **I want to** get an overview of the last commit results, separated by product configuration and test case
- **so that** I can see the status of the latest code updates at one glance.

This use case is implemented in the component *Testcase History*.



			commit	commit	commit	commit	commit	commit	commit	
Test Case	Health	Comment	#56	#55	#54	#54	#53	#52	#51	...
TestCase #1	🟩				🟩				🟩	
TestCase #2	X	John Doe notified			X				X	
TestCase #3	🟩		⬜	🟩	🟩	🟩	🟩	🟩	🟩	
TestCase #4	X	RDSW-1234			X		X	🟩	🟩	
TestCase #5	🟩				?⬜				🟩	
TestCase #6	🟩				100				100	
TestCase #7	🟩X	RDSW-4321			100				100	
TestCase #8	🟩X	Should be fixed once we get a new cable			🟩				100	
...										

icon	meaning
🟩	test passes
X	test fails
⬜	test started
10	test was executed 10 times and it passes
10	test was executed 10 times and it flickers
🟩X	test is flickering
?	test failed most likely due to a hardware problem or a streak

Figure 3.2.: Mockup: Tabular view by Sonova

3.2.5. Test result of a single test case at a specific time

- **As a** developer
- **I want to** get an overview of a test result from a single test case at a specific time
- **so that** I can follow up what happened at a specific time.

Original requirement: Req-1.2.1: History of a test case and Req-2.3: Filters.
UI sketch: Figure 3.3.

This use case is implemented in the component *Testcase History*, using the query feature to filter by test case name and time.

3.2.6. History of test results of a single test case

- **As a** developer
- **I want to** investigate the history of a single test case

- **so that** I can detect regression and / or flickering tests.

Original requirement: Req-1.2.1: History of a test case.
UI sketch: Figure 3.3.

This use case is implemented in the component *Testcase History*, using the query feature to filter for a single test case name.

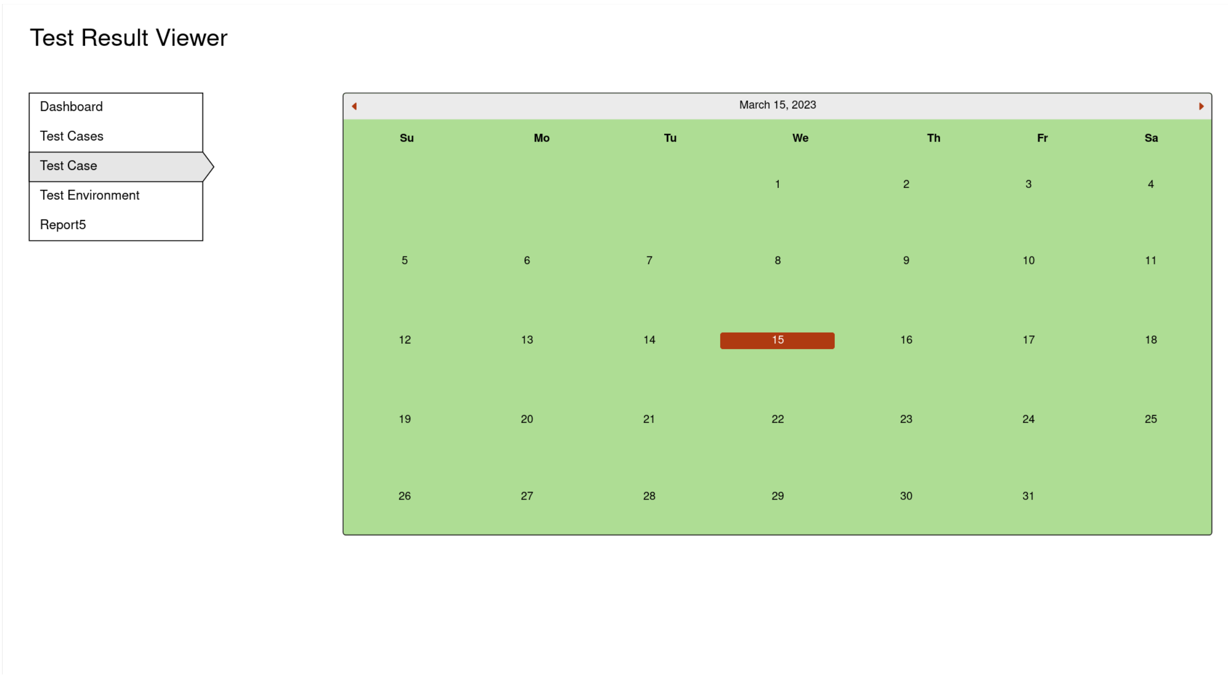


Figure 3.3.: Mockup: Test result of a single test case on a timeline

3.2.7. History of test results on a specific test environment

- **As a** test environment manager
- **I want to** see the history of test results grouped by test environment
- **so that** I can follow up on any issues that are apparent.

Original requirement: Req-1.2.2: History of a test environment
UI sketch: Figure 3.7

This requirement is implemented in the component *Test Environment*, using the query feature to filter for a single test environment.

3.2.8. Assigning a label to a test result

- **As a** developer
- **I want to** attach various labels to a test result
- **so that** I can add more metadata *after* the test run.

Examples what can be achieved using labels:

- mark a test result as irrelevant (e.g. because the test environment was down)
- mention a related task/bug
- mention someone who should look into a specific failure

Original requirement: Req-2.1: Labels
This requirement is implemented in the component *Testresult Detail*, accessible from any tabular view via the colored result box.

3.2.9. Hierarchical test order

- **As a** developer
- **I want to** have collapsible test suits
- **so that** I can choose to display only relevant data.

Original requirement: Req-1.6: Hierarchical test order
UI sketch: Figure 3.4

This requirement is implemented in the component *Testcase History*. Rows for test suite names can be expanded and collapsed.

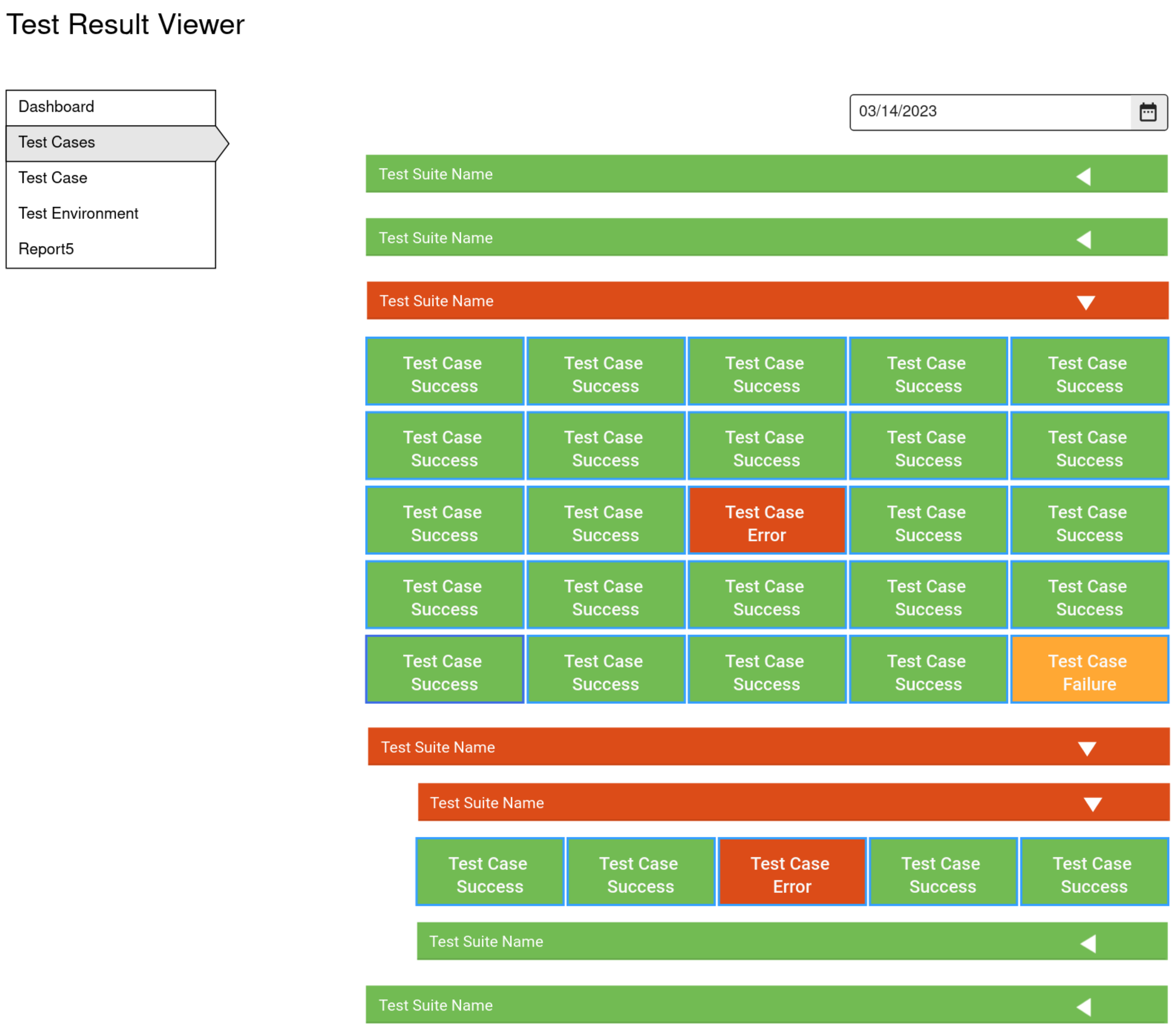


Figure 3.4.: Mockup: Test results of all test cases

3.2.10. Detection of regression of a particular test case

- **As a** developer
- **I want to** see if the latest commit introduced a regression for a particular test case
- **so that** I can limit the bug to a few lines of code.

Original requirement: Req-3.1: Detection of regression
UI sketch: Figure 3.5

This requirement is implemented in the component *Regression*. The current status of this is unfinished, further work on this use case has been stopped to prioritize the comment and assignment feature (May 15, Weekly Meeting, see requirements at subsection 3.2.14 and subsection 3.2.8). Using the filter form in the *Testcase History* component, it is possible to find regressions manually.

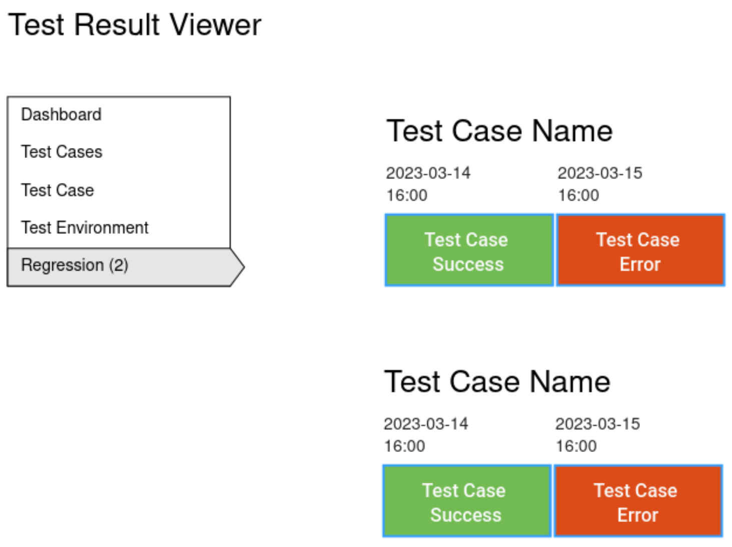


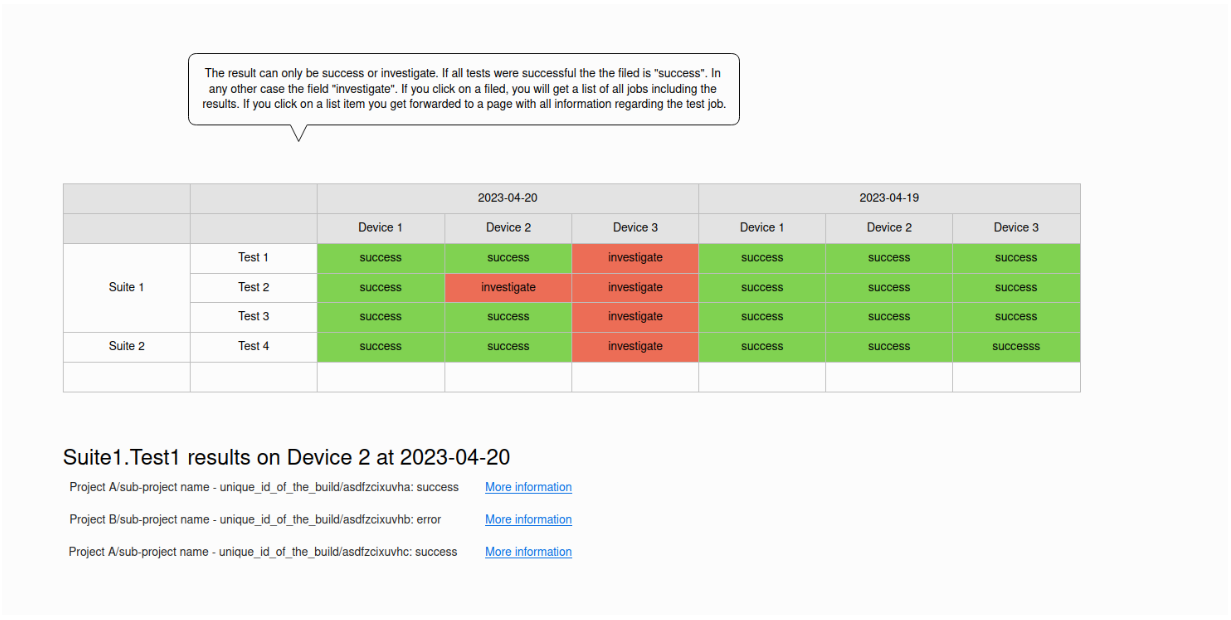
Figure 3.5.: Mockup: Regression display

3.2.11. Detection of flickering tests

- **As a** test environment manager
- **I want to** see if tests are producing differing results when run several times in sequence
- **so that** I can investigate possible hardware issues.

Original requirement: Req-3.2: Detection of flickering tests
UI sketch: Figure 3.6

This requirement is implemented in the component *Test Environment* as a history view for the different hardware devices. It uses a gradient color to indicate multiple test runs produced different results.



The result can only be success or investigate. If all tests were successful the the filed is "success". In any other case the field "investigate". If you click on a filed, you will get a list of all jobs including the results. If you click on a list item you get forwarded to a page with all information regarding the test job.

		2023-04-20			2023-04-19		
		Device 1	Device 2	Device 3	Device 1	Device 2	Device 3
Suite 1	Test 1	success	success	investigate	success	success	success
	Test 2	success	investigate	investigate	success	success	success
	Test 3	success	success	investigate	success	success	success
Suite 2	Test 4	success	success	investigate	success	success	successs

Suite1.Test1 results on Device 2 at 2023-04-20

Project A/sub-project name - unique_id_of_the_build/asdfzcixuvha: success More information

Project B/sub-project name - unique_id_of_the_build/asdfzcixuvhb: error More information

Project A/sub-project name - unique_id_of_the_build/asdfzcixuvhc: success More information

Figure 3.6.: Mockup: History for Test Environments

3.2.12. Detection of test environment problems

- **As a** test environment manager
- **I want to** have a quick overview of error results on specific hardware environments
- **so that** I can investigate possible hardware issues.

Original requirement: Req-3.2: Detection of flickering tests
UI sketch: Figure 3.7

This requirement is implemented in the component *Dashboard*, using a chart to display results by test environment.

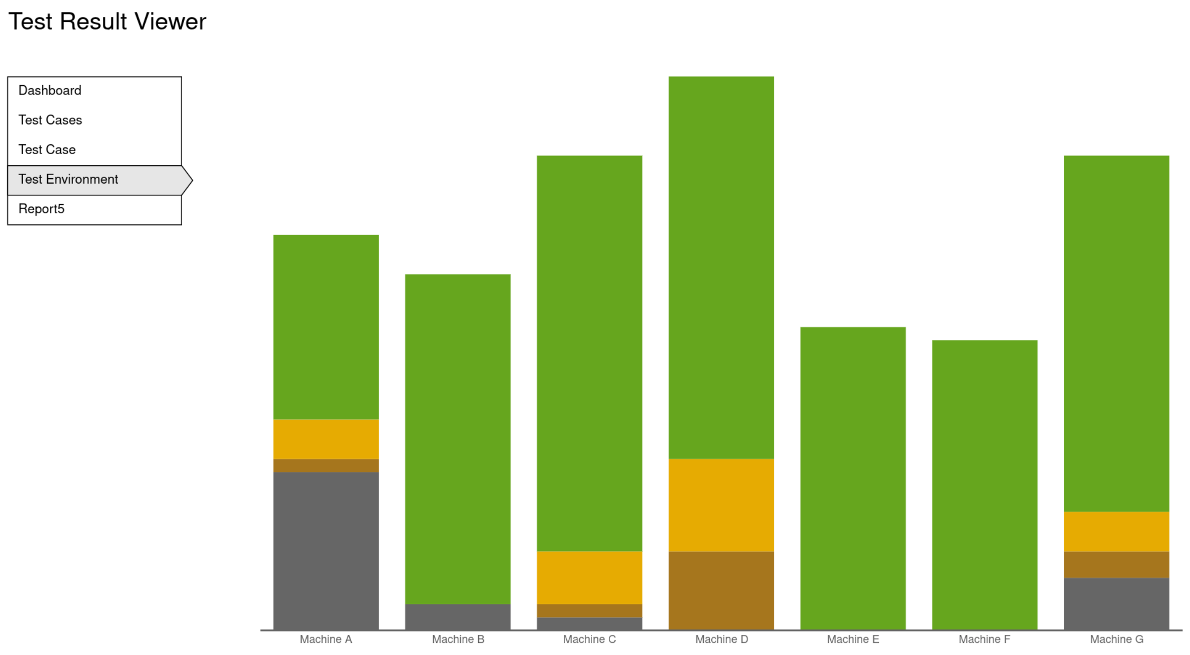


Figure 3.7.: Mockup: Test result of all test environments

3.2.13. Detection of similar/related test failures

- **As a** developer
- **I want to** see hints to test failures that might be related to each other in some meaningful way
- **so that** I can investigate possible problems that affect several test cases in a way that isn't obvious.

Original requirement: Req-3.4: Advanced detection of test failures

This requirement is not implemented due to lack of time and low prioritization.

3.2.14. Attach a comment to a test case

- **As a** developer
- **I want to** comment on a test case execution
- **so that** I can provide additional information.

Original requirement: Req-4.2: Comments on test case execution

This requirement is implemented in the component *Testcase History*, using a specific column in the table.

3.2.15. Assign a Developer to a test case

- **As a** developer
- **I want to** assign a developer to a test case
- **so that** I know who is in charge of a specific test case.

Original requirement: Req-4.1: Test owner

This requirement is implemented in the component *Testcase History*, using a specific column in the table.

3.3. Non-Functional Requirements

The following non-functional requirements have either been suggested by Sonova at the start of the project or added by the developers to reflect discussions that took place at the weekly meetings. Borrowed from the well known SMART notation for defining goals, we define the relevant traits of these requirements as follows:

- **Specific**: Which part of the system is affected / has to fulfill the requirement?
- **Measurable**: How can be decided if the requirement is met or not?
- **Agreed upon**: Do both parties involved (developers and stakeholders) agree on the requirement and the wording used?
- **Relevant**: What reason is there for this requirement? What user personas benefit from fulfilling it?
- **Time-bound**: When should the requirement meet the proposed measure?

This specific use of the terms is based on the materials provided at the Application Architecture course taught at OST, written by Olaf Zimmermann [54]. Instead of the definition *realistic* we chose *relevant* to provide the main reason for each requirement. The fact that a requirement has been proposed should in our case confirm that it is in fact realistic.

ID	NFR-1 (Performance)
Requirement	Response time should be kept low to keep the application usable. See Req-8.1: Query response time
Specification	Frontend, every application page
Measure(s)	Page loading time **must not** be longer than 2.0 seconds when simple queries to the database are involved. Loading animations **should** be used to indicate data being fetched.
Agreement	Requirement placed by Sonova
Relevance	Waiting time when using the product will decrease adoption
Timeframe	Requirement must be met on project completion
Additional information	*Simple queries* meaning the requested data is not older than 24 hours **or** is limited to the details of a specific record (test case, revision, commit). For Data characteristics, see Appendix C

Table 3.2.: NFR-1, Performance

ID	NFR-2 (Confidentiality)
Requirement	The application will only be available in an internal network at Sonova. Security measures should be implemented according to this usage criteria.
Specification	All application components
Measure(s)	Communication between Client and Application as well as between application components **must** be encrypted using TLS.
Agreement	Requirement proposed by the developers
Relevance	Basic security should protect sensitive data
Timeframe	Requirement must be met on project completion
Additional information	Even with internal use only, security breaches can happen. Basic security measures should be applied.

Table 3.3.: NFR-2, Confidentiality

ID	NFR-3 (Browser Support)
Requirement	The application should support the use of different web browsers. Although the goal is for the application to be browser-agnostic, we will use the following prioritization for different browser products, based on internal use at Sonova. 1. Chromium (Microsoft Edge) 2. Mozilla Firefox
Specification	Frontend, every application page
Measure(s)	Chrome based browsers **must** support all application features. Firefox **should** support all application features, with workarounds to avoid losing information. Any other browsers (e.g. Safari) will not be tested for feature support.
Agreement	Requirement proposed by the developers
Relevance	Browser support should be aligned with effective browser usage to avoid overengineering
Timeframe	Requirement must be met on project completion

Table 3.4.: NFR-3, Browser Support

4. Analysis

This chapter contains considerations for the information we were given as well as the choice of technology for each component. The reasoning behind the design decisions is summarized in section 5.6 to provide a brief overview for later reference. First, section 4.1 focuses on the data structure, the following sections provide information about the Data Consumer (4.2), Data Store (4.3), Frontend (4.4), Backend (4.6) and Reverse Proxy (4.7) components. Figure 4.1 shows our planned components in the project context.

Although Sonova has often expressed their own preferences about what technologies they would use for each component, we performed our own analysis to ensure there were no obvious drawbacks to using their preferred solution.

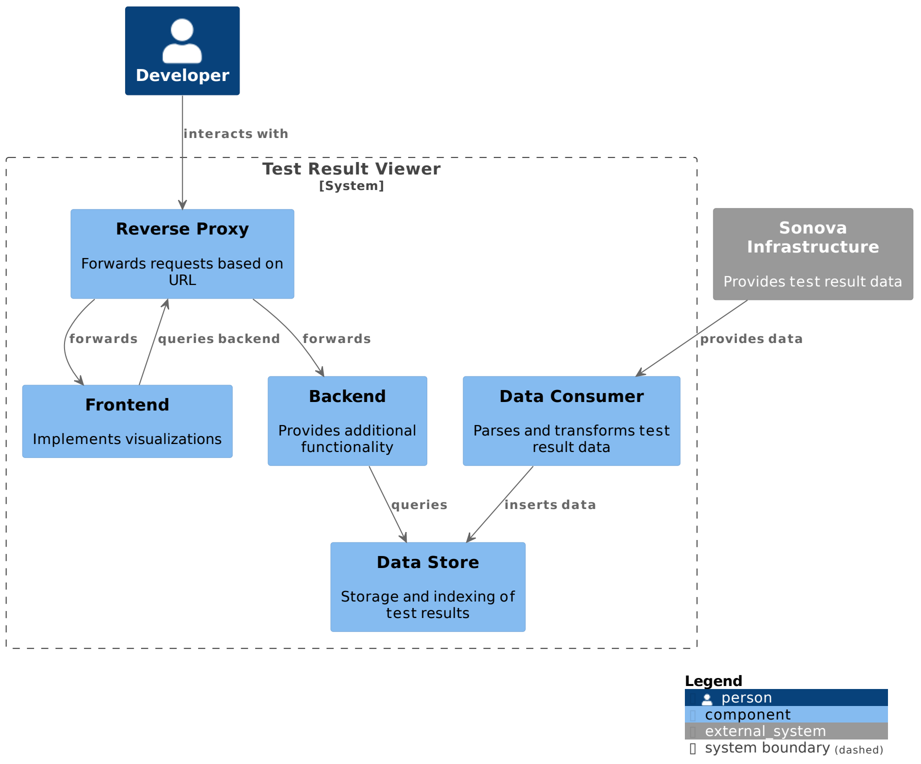


Figure 4.1.: Overview of planned Components

4.1. Data Structure

The data structure is the foundation of this platform. To support many different requirements the data structure must be as complete but also as flexible as possible. Therefore, it was the first task to find a suitable solution.

The final structure combines two sources of information: a report from a testing framework and a JSON file with additional metadata. We have been provided with two sample files of a test result following the NUnit Standard XML Format Version 2.0 [42] and one following the JUnit Format [28]. The JSON file contains additional data from several systems of Sonovas existing infrastructure and is constructed and delivered by them. As discussed in our weekly meetings, it is their responsibility to provide all information relevant for any use case they require and ours to consume and process the data.

4.1.1. Known Limitations

The following limitations are relevant at the time of this project and should be evaluated again in case of an upgrade of the test framework version used by Sonova.

Missing end time in NUnit: At the time of writing Sonova uses NUnit Version 2. The XML schema used by this version contains neither the time when the test run as a whole ended nor when a single test case ended. It only provides the information when the test run was started and how long the execution of each test suite and test case took [42]. NUnit Version 3 provides information about start and end time of each test run, test suite and test case [41].

Missing start / end time in JUnit: Similar to NUnit Version 2, JUnit does not provide exact information about when each test case execution has started. It only provides information about the time the test suite has started and how long the execution of each test case has lasted [28].

4.1.2. Domain Model

Figure 4.2 defines the relationship between the different data sources and the information they provide. The figure is based on NUnit. However, for this project JUnit acts as a subset of NUnit so it is not displayed separately.

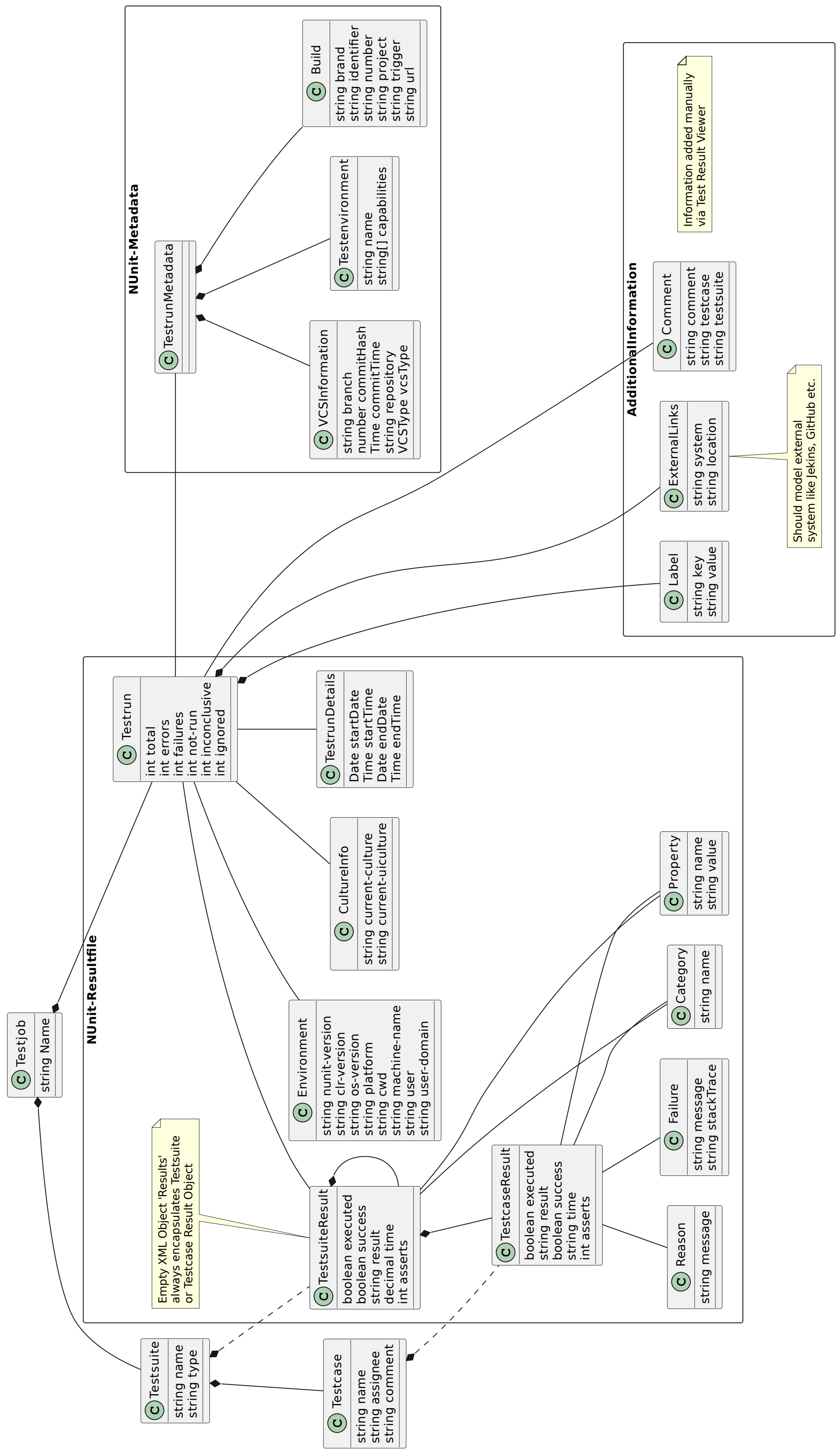


Figure 4.2.: Domain Model: NUnit XML with added metadata

4.2. Data Consumer

The data consumer component will be responsible for parsing and transforming input data for test results to a format that corresponds to the data structure. Data will be retrieved from a Kafka Topic that will be set up by Sonova in the future. Currently it is necessary to parse two separate files for the XML result and the JSON metadata. Also, the automated and randomized generation of sample test results based on the files given has been advised to create a realistic amount of data for testing.

4.2.1. Decision

At Sonova, many utilities used in the CI/CD area are written using Python, so it is proposed that we do the same. A template package can be provided as well. Because an easy integration into the existing environment is valued higher than potential advantages of other tools (mainly: better performance of compiled over interpreted languages), we decided to use Python without any further analysis.

4.3. Data Store

The data store component stores test result information including metadata. Additionally, it must be possible to query and aggregate the data inside the store. The data store will be populated by the data consumer.

Sonova proposed Elasticsearch as a primary solution. They did not investigate any alternatives for lack of time. To ensure we do not miss any better solutions we decided to evaluate possible alternatives. The technology should be able to fulfill the following requirements:

1. easy to scale horizontally
2. easy to add new fields to schema
3. short time for search and query operations

Common Relational Database Management Systems (RDBMS) normally have a strict database schema which makes it difficult to change the data structure later [7, 30]. Also, these systems are not optimized for horizontal scaling. Because of these reasons we focused on NoSQL databases only.

> *While SQL is valued for ensuring data validity, NoSQL is good when it's more important that the availability of big data is fast. It's also a good choice when a company will need to scale because of changing requirements. NoSQL is easy-to-use, flexible and offers high performance [7].*

After some research we chose the following technologies for a more detailed comparison:

- RavenDB (see 4.3.1)
- CouchDB (see 4.3.2)
- Elasticsearch (see 4.3.3)
- MongoDB (see 4.3.4)

4.3.1. RavenDB

RavenDB is a document-oriented NoSQL database, and in contrast to other NoSQL databases it is fully ACID compatible [3]. In the current scope of this project, writing to the database is not as important as fast querying. Therefore, ACID as a property is useful but not crucial.

For a full-text search, RavenDB uses Lucene as its search backend [24]. A critical function is aggregating data over a large set of data in a short time. To perform such operations fast, RavenDB provides Map-Reduce indexes [25]. These are special views which are calculated during indexing and updated during insert, update or delete operations. Therefore querying such an index is fast, with the drawback that it has to be defined at development and can not easily be extended at runtime [25].

4.3.2. CouchDB

CouchDB is a NoSQL database which focuses on the *Availability* and *Partition tolerance* aspects of the CAP theorem. Its strengths are horizontal scaling using master-slave or master-master replication [11].

According to existing comparisons, MongoDB has a better read performance [13, 52]. The documentation also states that performance issues might occur in a huge data set:

> *With up to tens of thousands of documents you will generally find CouchDB to perform well no matter how you write your code. Once you start getting into the millions of documents you need to be a lot more careful [1].*

4.3.3. Elasticsearch

Elasticsearch is not a traditional database but a search engine based on Lucene. It is thus focused on low query time at the cost of higher inserting time using inverted indexes [48]. However, Elasticsearch can store documents and is therefore an option for this project.

> *Elasticsearch is the distributed search and analytics engine at the heart of the Elastic Stack. It provides near real-time search and analytics for all types of data. Whether you have structured or unstructured text, numerical data, or geospatial data, Elasticsearch can efficiently store and index it in a way that supports fast searches. Elasticsearch provides a REST API that enables you to store data in Elasticsearch and retrieve it. The REST API also provides access to Elasticsearch search and analytics capabilities.*
>
> *Data in: Elasticsearch is a distributed document store. Instead of storing information as rows of columnar data, Elasticsearch stores complex data structures that have been serialized as JSON documents. When you have multiple Elasticsearch nodes in a cluster, stored documents are distributed across the cluster and can be accessed immediately from any node.*
>
> *Information out: While you can use Elasticsearch as a document store and retrieve documents and their metadata, the real power comes from being able to easily access the full suite of search capabilities built on the Apache Lucene search engine library [17].*

In comparison to other NoSQL databases it has a much higher insertion time but outperforms others in terms of searching and aggregating data [23].

For this project, modifying operations are not critical and can take longer. In case of performance issues with modifying operations, it is an option to use another database as data store and use Elasticsearch on top of that using connectors [16]. All operations are performed using an HTTP API.

4.3.4. MongoDB

MongoDB is one of the most common NoSQL databases [34]. It stores the documents in BSON format.

Similar to Elasticsearch, MongoDB is fast in searching data and performing aggregations. However, as soon as the search queries are more complex, Elasticsearch performs better than MongoDB [15]. Other comparisons produced a similar result: Elasticsearch is faster in searching. However in inserting, updating, deleting and storing, MongoDB performs better [35].

The solution using MongoDB as a database and Elasticsearch for searching and analyzing the data is described in the paper "Massive Semi-structured Data Platform based on Elasticsearch and MongoDB" [15]. As it is mentioned there, MongoDB on its own is not capable of quickly calculating search results using complex queries for huge amounts of data.

4.3.5. Decision

Req-1.1: Customizable View requires the creation of expensive aggregations at runtime. Although we do not implement this requirement as part of the thesis, we want to to build the application in a way that Sonova can add the feature in the future, therefore we neglected RavenDB as our data store.

We neglected CouchDB and MongoDB as an option because of their lower reading speeds and search performance, which are the most important criteria.

We chose to build the application using Elasticsearch, satisfying the requirement for fast querying times. Elasticsearch will be used for both searching as well as storing data to keep the complexity of the setup low.

When the application evolves and requirements for tasks like inserting and updating are more important, another technology like MongoDB can be added. Because Elasticsearch can also be used to search data stored using another database technolgy, very few changes to the application logic would be necessary.

4.4. Frontend Approach

To analyze and display our data a flexible frontend is necessary. There are several options for such an implementation. We have selected the following three options for further analysis:

- Kibana (see 4.4.1)
- Grafana (see 4.4.2)
- Creating a Custom Frontend (see 4.4.3)

4.4.1. Kibana

Kibana is a ready to use frontend for Elasticsearch and is part of the Elasticstack, as shown in Figure 4.3 [17]. This is a big advantage of Kibana over other query frontends, as it is optimized to work with an Elasticsearch cluster and offers some convenient functionality for cluster management (see 5.4.1 for details). Kibana is designed to analyze and search for data stored in an Elasticsearch instance. Its focus lies on easy creation and editing of visual dashboards with different kinds of graphs and visualizations pre-built.

For a prototype with some basic visualizations these solutions would be adequate. However, inserting additional data into the data store is not a common use case of Kibana. To do so would require calls to the Elasticsearch API, e.g. using the Kibana Dev Tools [38]. This is cumbersome and not feasible for everyday use. Since requirements like subsection 3.2.8 and subsection 3.2.15

rely on this functionality, Kibana is not suitable as our only frontend. However it may be deployed in addition to another frontend for the purpose of managing the cluster.

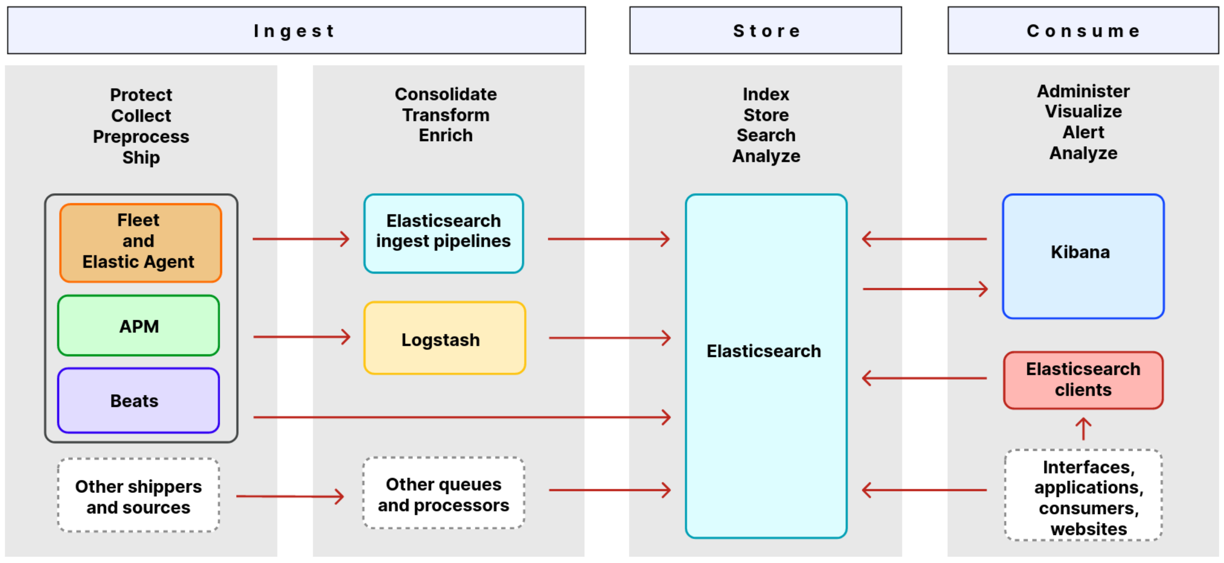


Figure 4.3.: Elasticstack Components [17]

4.4.2. Grafana

Grafana is an open source visualization tool that can be used with a number of data sources, including but not limited to Elasticsearch. It was started originally as a fork of Kibana [22]. Grafana is primarily used to create dashboards for data from various sources. Similar to Kibana, Grafana is not suitable to manipulate an existing data source [26, 27]. Therefore we decided against Grafana as our primary frontend.

4.4.3. Custom Frontend

A custom frontend using a well known web framework like React, Vue.js or Angular can be used to directly access Elasticsearch data over its API. This comes with the advantage of maximal control over the UI and all possible features, but it requires the development and maintenance of a whole new code base. In contrast to Kibana and Grafana, this solution allows us to easily insert and update data from the frontend.

4.4.4. Decision

Kibana and Grafana are great tools to configure dashboards and visualize data from Elasticsearch. However, both are lacking an easy way to insert and update data, which is an important requirement. In addition, a custom frontend can be extended and adapted to Sonova's needs. Because of these reasons, we decided to implement a custom frontend.

4.5. Custom Frontend Frameworks

The most used web frameworks and libraries in 2022 were React.js, Angular and Vue.js according to the "State of JS" survey [43], which is conducted yearly by a team of open source developers, contributors and consultants.

Table 4.1 compares the three technologies in type and the language that is generally used with it.

Name	Type	Language
Angular	Framework	TypeScript
React.js	Library	JavaScript
Vue.js	Library	JavaScript

Table 4.1.: Technical comparison Web Frameworks

Our comparison in Table 4.2 is limited to these three frameworks and represents the information taken from the survey. The values were calculated as follows:

Retention: would use again / (would use again + would not use again)

Interest: want to learn / (want to learn + not interested)

Usage: (would use again + would not use again) / total

Awareness: (total - never heard) / total

Framework / Library	Usage	Retention	Interest	Awareness
Angular	49%	43%	20%	100%
React.js	82%	77%	47%	100%
Vue.js	46%	77%	51%	100%

Table 4.2.: Survey Results Web Frameworks

According to the results, more than half of the participating developers do not plan to work with Angular again. On the other hand, 77% of the participants would choose React.js and Vue.js for another project.

4.5.1. Decision

To be able to update as well as retrieve data from Elasticsearch we chose to build a custom frontend (4.4.4). According to the list of use cases it must support

- various visualizations of data (see 3.2 Functional Requirements: Use Cases)
- attaching and editing additional data (see 3.2.8 Assigning a label to a test result).

The only web frontend technology previously used at Sonova is Angular. The maintenance of internal applications like this one is usually not the main focus of their developers and therefore they do not have time to learn new technologies for this specifically. If the Test Result Viewer were to be implemented using any other technology like React.js or Vue.js, Sonova would eventually rewrite the whole application in Angular. For this reason and since none of us have worked with any of the mentioned web frameworks before, we decided to use Angular in agreement with Sonova (see meeting notes from March 13 in section E).

In addition, an out-of-the-box Kibana instance will be part of the application as well to provide some powerful features for data management as well as access to the elastic API via the Kibana Dev Tools [29].

Following this, the analysis in subsection 4.4.3 was not continued because it would not change the design decision. Had this restriction of Sonova not existed, we would have made further comparisons to choose between React.js and Vue.js.

4.6. Backend

Implementing a backend service for the Test Result Viewer provides possibilities for some additional features:

- Forwarding frontend requests to elasticserach (see 4.6.1)
- Storing additional configuration (see 4.6.2)
- Authentication and Authorization (see 4.6.3)

4.6.1. Forwarding Frontend Requests to Elasticsearch

Without a backend it is required to deliver the database (API) credentials to the browser on the client. This could lead to critical security issues.

Another problem is that a potential attacker could send arbitrary queries to the database which could lead to a Denial-of-Service scenario. With the additional logic of a backend, it is possible to control what queries are sent to the database.

Because of the reasons mentioned Elasticsearch does not officially support the JavaScript client library `elasticsearch.js` to be used in the browser [18]. Instead it is recommended to write a lightweight backend to forward requests to the elastic API.

4.6.2. Storing Additional Configuration

As the application evolves, at some point it may be possible to create custom dashboards in the frontend. This dashboard configuration will have to be stored somewhere. Although it is possible to achieve this feature without a backend, we strongly recommend not to implement this logic in the browser.

4.6.3. Authentication and Authorization

Depending on the authentication and authorization mechanism used, this could be done without a backend. However, as soon as the chosen mechanism is not usable over HTTP a backend is required to communicate with the authentication provider.

4.6.4. Possible Backend Technologies

Limiting the number of programming languages used in this project reduces the overall complexity. Since the data consumer will be written in Python and the frontend in TypeScript, we will limit the possible technologies to frameworks using Python, JavaScript or Typescript. Some examples are listed in Table 4.3.

Name	**Language**
Django	Python
Flask	Python
Hug	Python
express.js	JS / TS
nest.js	JS / TS

Table 4.3.: Possible backend frameworks

4.6.5. Decision

In agreement with Sonova we decided to not implement a backend. They have acknowledged the possible security risk because the application will only be accessible from the internal network. Additional features may be implemented later by Sonova. See Meeting Notes (March 20) in section E.

4.7. Reverse Proxy

A reverse proxy can be used to prevent Cross Origin Resource Sharing (CORS) issues and serve different components on the same domain. This ensures that all web requests that are made from the frontend will have the same destination domain and thus be allowed without the need of adding any additional CORS configuration [12].

4.7.1. Possible Products

For this purpose various proxy tools exist. Some of the most well known solutions are:

- Traefik
- Caddy
- Nginx

Our choice should be fast and simple to configure. These tools are well established and should all support our use case without any problems.

4.7.2. Decision

Sonova currently uses Nginx as the primary reverse proxy product. In order to reduce the integration risks we decided to use Nginx for this project as well. However, we have tested Caddy for our local development and have found it provides a simpler configuration interface.

5. Architecture

This chapter contains information about the decisions made and technologies used in the final product.

Section 5.1 gives an overview about the architecture to be implemented. This is followed by more information on every component in a dedicated chapter, then section 5.6 lists all relevant design decisions using Y-Statements.

The grey boxes in figures 5.2 - 5.4 are systems run by Sonova, thus they are external to the Test Result Viewer. Implementing these integrations is not in the scope of this thesis. In the current scenario a developer manually runs the data consumer in absence of Kafka to trigger an event for new data. However, all following diagrams except Figure 5.4 show only the final state with all integrations present.

5.1. General architecture overview

Figure 5.1 describes the flow of data from creation by the CI Machine to presentation in the frontend.

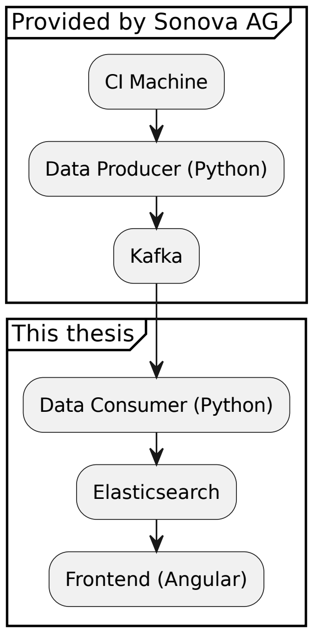


Figure 5.1.: General Data Flow

Figure 5.2 describes the system context which our product will integrate with. The main input is provided by a Kafka Topic which provides test results as raw data. Different links will be made from the Test Result Viewer to other systems in order to provide additional relevant information.

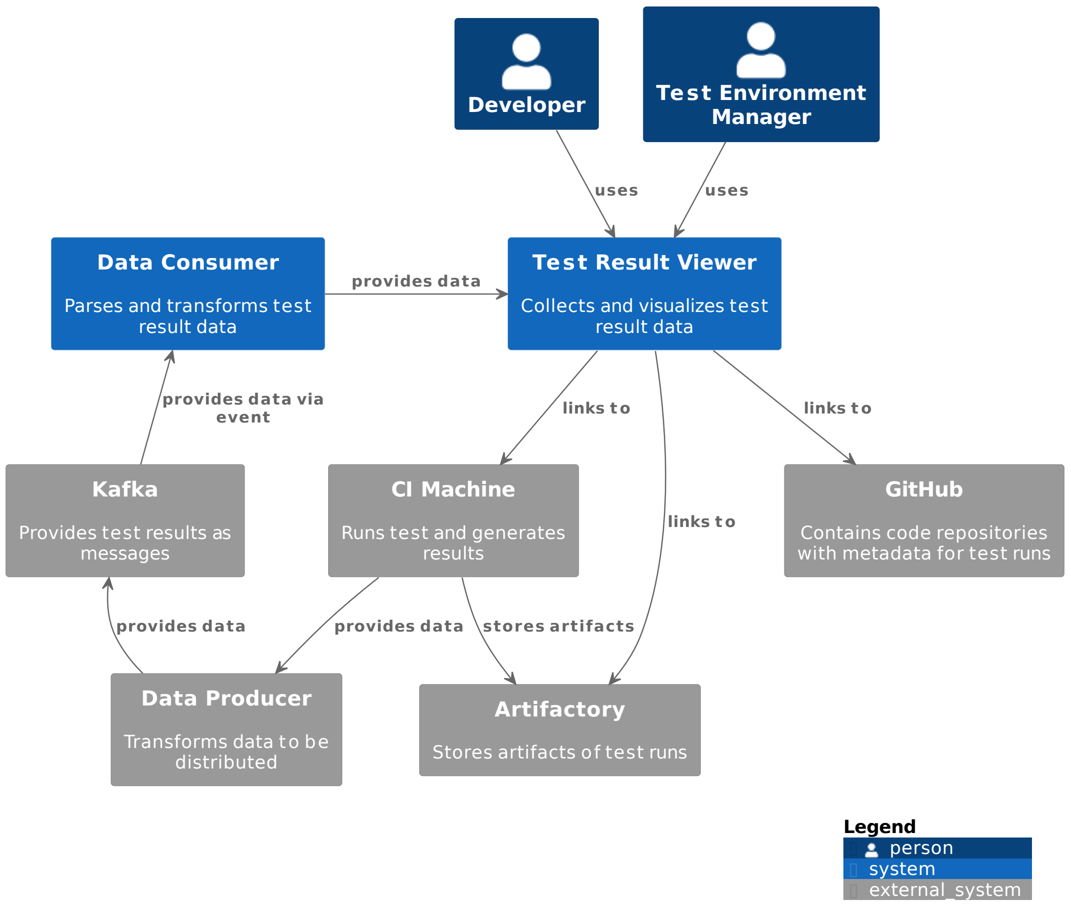

Developer
Test Environment Manager
uses
uses
Data Consumer
Parses and transforms test result data
provides data
Test Result Viewer
Collects and visualizes test result data
provides data via event
links to
links to
links to
Kafka
Provides test results as messages
CI Machine
Runs test and generates results
GitHub
Contains code repositories with metadata for test runs
provides data
provides data
stores artifacts
Data Producer
Transforms data to be distributed
Artifactory
Stores artifacts of test runs
Legend
person
system
external_system

Figure 5.2.: C4 Context Diagram

Figure 5.3 provides an overview of the internal structure for the Test Result Viewer. Two different Docker Compose projects have been set up to provide the containers needed.

The **Elasticsearch** project provides 3 containers for Elasticsearch and one for Kibana. The sizing of this has been taken from a sample implementation by Elastic and is a good starting point to provide some redundancy.

The **Viewer** project contains a docker compose setup for a single instance of nginx as well as the frontend application built using Angular.

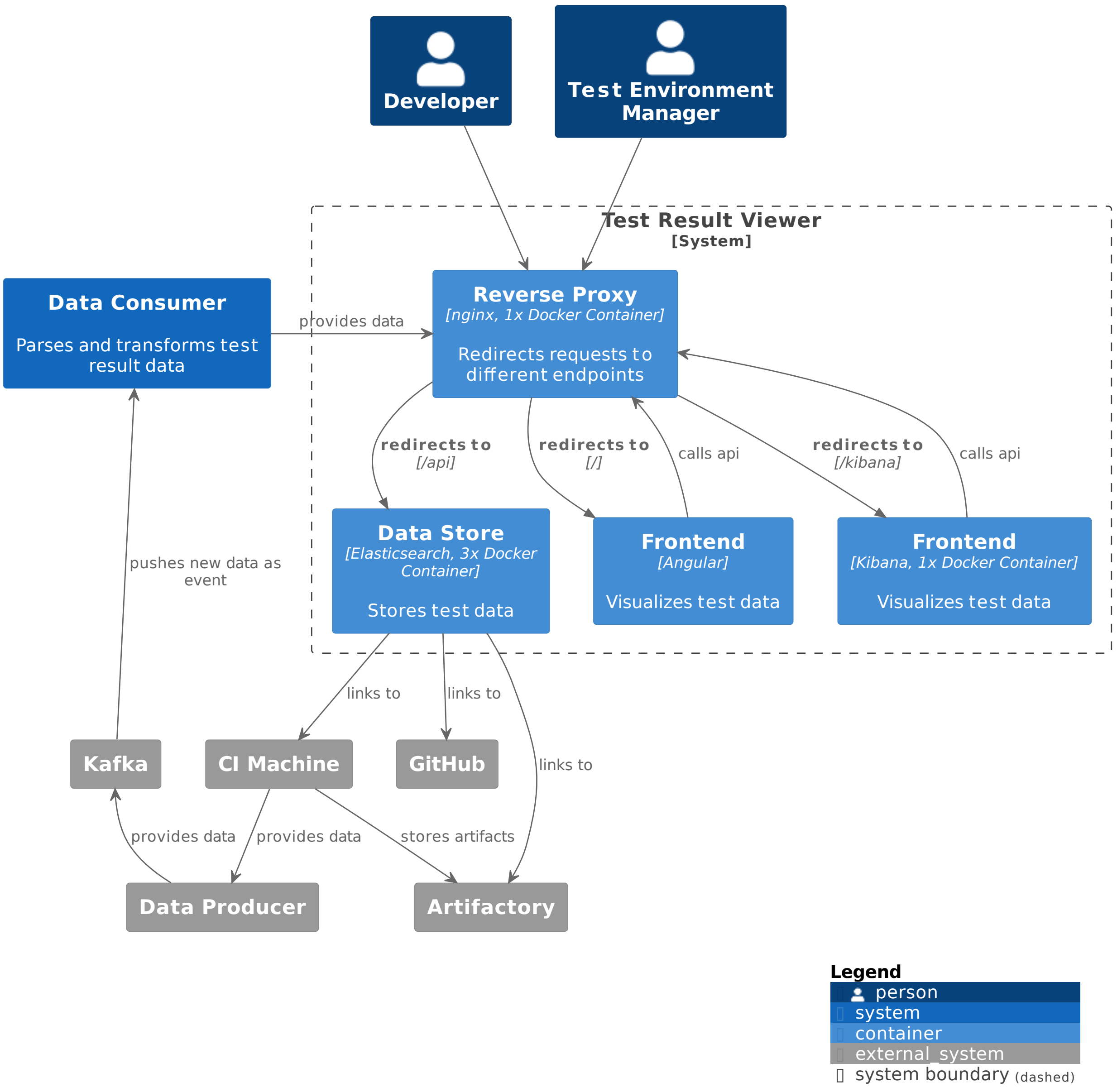


Figure 5.3.: C4 Container Diagram

5.2. Data Consumer

The data consumer is responsible for fetching test results from a source, transforming them into the format needed and then inserting them into Elasticsearch. The component is written in Python.

Figure 5.4 displays the interaction with a developer triggering the generator. In the current version, both parsers are also triggered manually from command line, supplying the relevant files as arguments. Ultimately, this will all be done automatically by Kafka.

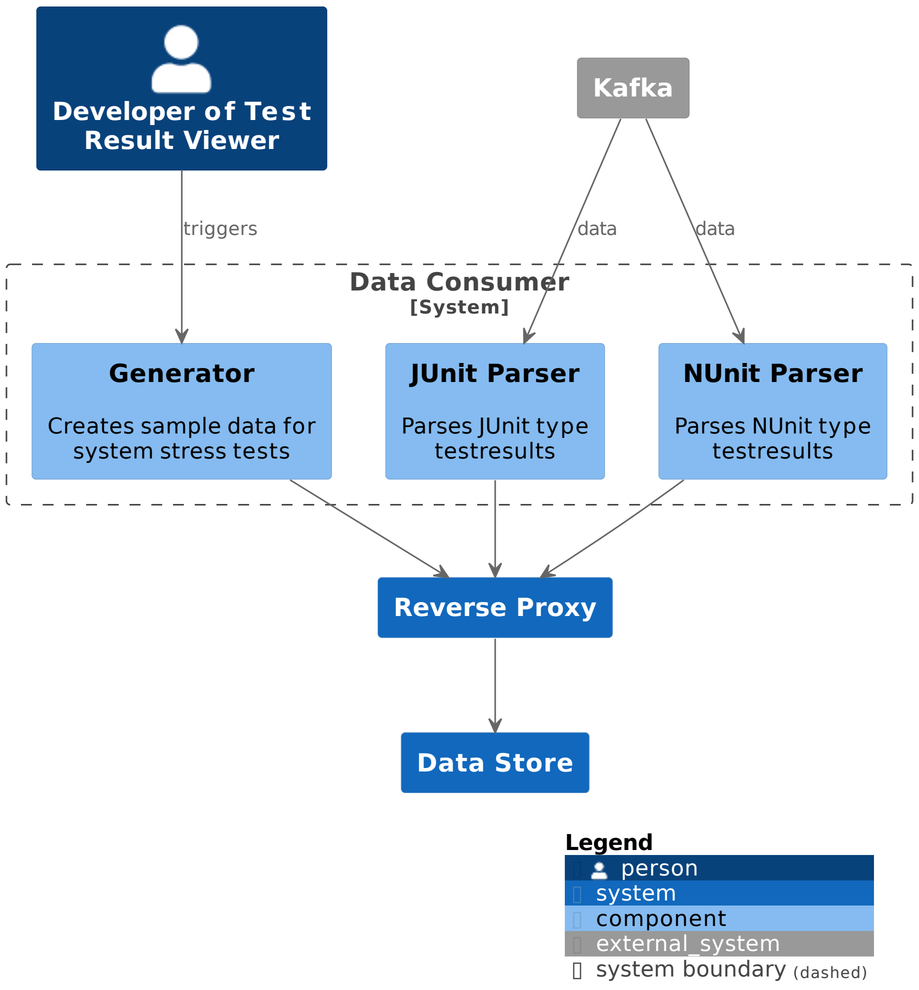


Figure 5.4.: C4 Component: Data Consumer

5.2.1. NUnit and JUnit Parser

To support different types of test result data, we need a parser that transforms the given structure to the format that is stored in Elasticsearch. Additional result types can easily be supported by supplying a new parser implementation file.

5.2.2. Generator

We have received two anonymized test sets from Sonova that reflect the structure of the data the tool will be processing. However, the amount of results included is not realistic. To create more data for testing visualizations and performance we have implemented a generator. It creates semi-randomized test data sets based on the examples provided.

The generator is only required during this thesis and can be abandoned afterwards since the developers at Sonova can use sets of real data for development and testing.

5.3. Data Store

The data store represents the central piece of our application where all test results are stored in a normalized way. According to the requirements, the main focus is on fast data querying.

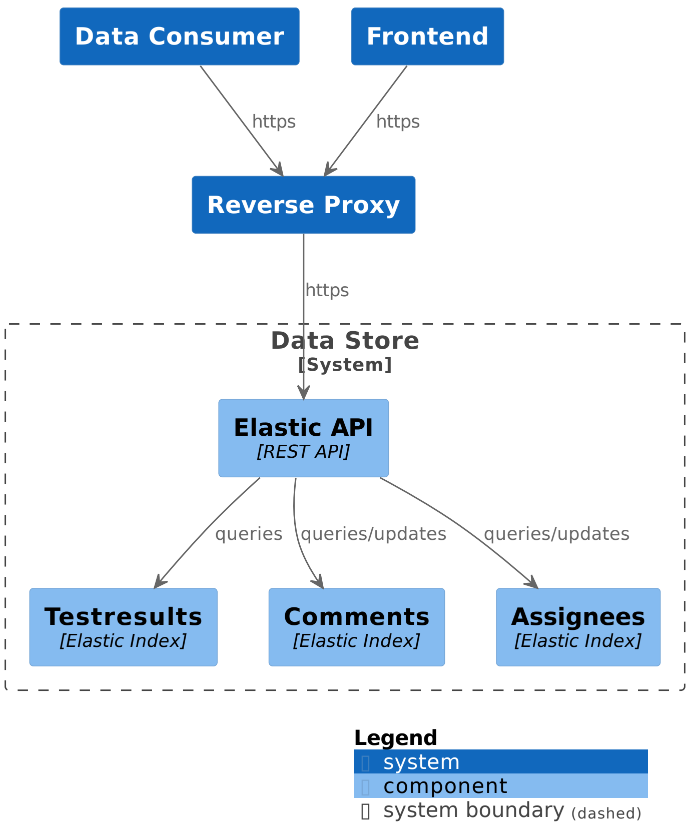


Figure 5.5.: C4 Component Diagram: Data Store

5.3.1. JSON metadata

The JSON metadata file contains information that should be stored along with the test resulst provided by the test framework. The file contains information that does not directly affect the test framework but might be important for our use cases, like details about repositories and commits, CI/CD jobs and such.

The exact information provided is decided in discussion with Sonova.

5.3.2. Test result minimal subset

The currently used version of NUnit at Sonova is 2.0. This versions XML schema is available on the website of The NUnit Project [42]. Another test framework that Sonova uses is JUnit, however the JUnit result file does not contain as much information as the NUnit file does [28]. It acts as a subset of the former. Some examples of missing information in JUnit is details about the run environment or culture information.

To account for these differences we defined a minimal subset of fields that every framework must provide to be supported by the application:

- framework name
- framework version
- test case name
- test suite name
- name / path of assembly under test
- start time of the test run

- if the test case was executed
- the result for the test case
- if result was an error: message and stack trace

If a test framework provides more information (e.g. a category, the start time of each test case), additional fields can be added as optional values. The final database schema can be found in Listing 6.4, which also integrates the JSON metadata.

5.4. Frontend

After creating a database schema and inserting the sample data into Elasticsearch, the second part of this project is creating visualizations. We used Kibana to create some basic visualizations as a prototype, but then abandoned these to focus on a custom frontend created with Angular.

5.4.1. Kibana

The decision to include Kibana in our Elasticsearch setup was based on the functionality it offers without any additional effort. It is included in the sample dockerized setup Elastic offers and requires minimal configuration.

During development of the frontend we have relied on the Kibana Dev Tools, an interface to send requests to the Elasticsearch API. It offers code completion features specifically for Elasticsearch which distinguishes it from other tools.

Kibana also offers ad-hoc data exploration and advanced cluster monitoring features with some additional setup [29].

5.4.2. Angular Web Application

The custom Angular web application contains all the logic needed to aggregate and display test results in a way that is easy for Sonova developers to work with. Here we implement all the planned use cases.

The frontend is strucutred using the SCAM pattern to ensure maintainability [33]. In the SCAM pattern each component is placed in its own module. Each page has the following structure:

```
page-name
├── data-access
│   └── page-name.service.ts
├── feature
│   ├── page-name.component.scss|html|ts
│   ├── page-name.routing.ts
│   └── page-name.module.ts
└── ui
    └── ui-component
        ├── ui-component.component.scss|html|ts
        ├── ui-component.routing.ts
        └── ui-component.module.ts
```

Figure 5.6.: Structure of a single Angular page

Figure 5.7 describes the application in more detail. It focuses on two different pages of the application with three parts each: Feature Component, UI Component and Service. This pattern applies ti any other page of the application that is accessible via the router.

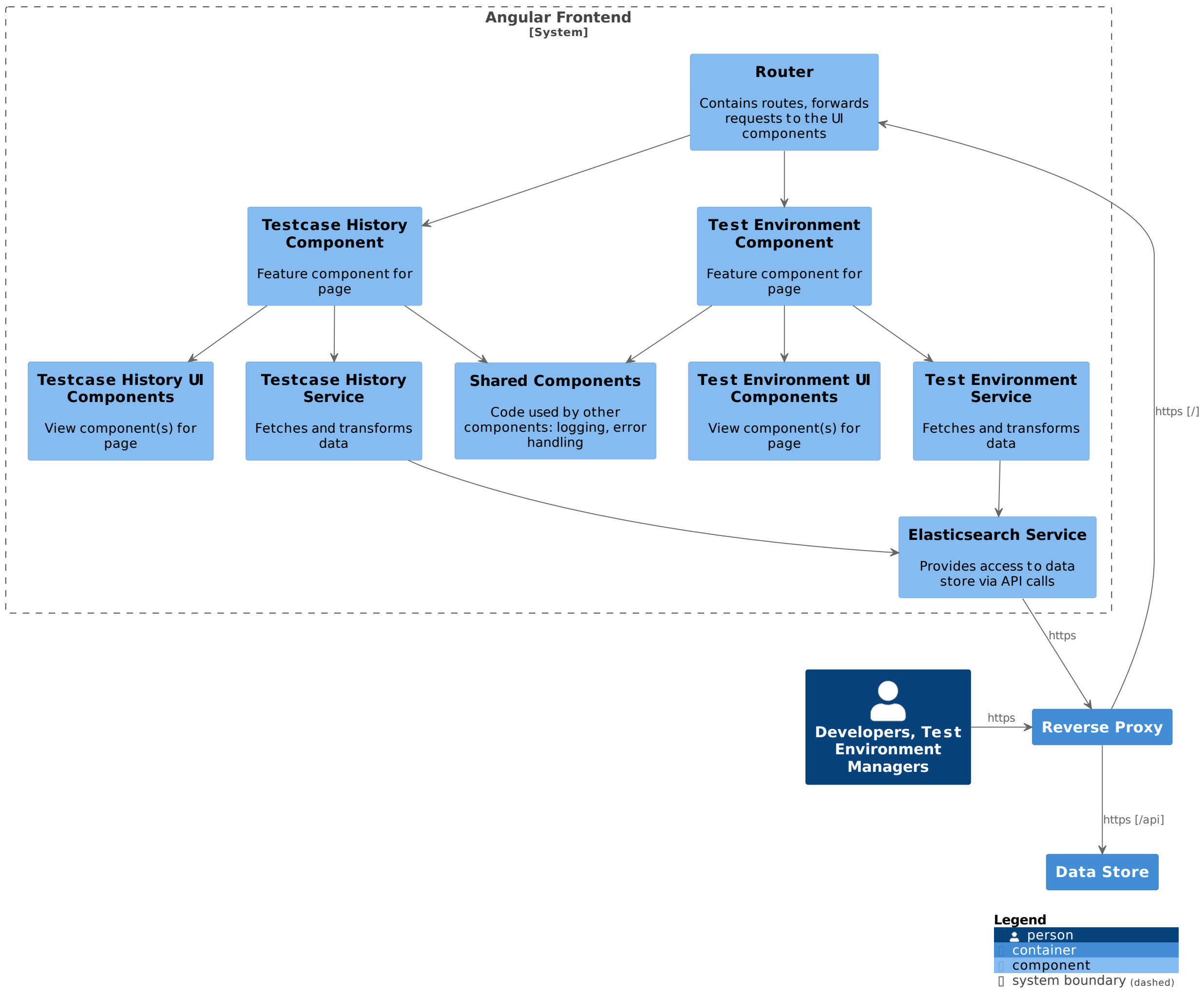

Angular Frontend
[System]
Router
Contains routes, forwards requests to the UI components
Testcase History Component
Feature component for page
Test Environment Component
Feature component for page
Testcase History UI Components
View component(s) for page
Testcase History Service
Fetches and transforms data
Shared Components
Code used by other components: logging, error handling
Test Environment UI Components
View component(s) for page
Test Environment Service
Fetches and transforms data
Elasticsearch Service
Provides access to data store via API calls
https [/]
https
Developers, Test Environment Managers
https
Reverse Proxy
https [/api]
Data Store
Legend
person
container
component
system boundary (dashed)

Figure 5.7.: C4 Component Diagram: Frontend

5.5. Reverse Proxy

To forward requests to the different parts of the frontend and the data store we use nginx as a reverse proxy. This allows addressing different destinations behind the same URL and thus eliminates the need for any CORS rules. The destinatins have already been described in Figure 5.3.

5.6. Design Decisions

The following design decisions have been documented in the form of Y-Statments [53]. Additional Information is supplied if necessary.

D-1.1: Scalability

In the context of scaling the database

facing the need to respond fast to incoming request

we decided to use a NoSQL data store

and neglected a relational database

to achieve an easy solution to scale horizontally

accepting the downside of not being ACID

For more detailed information, see section 4.3.

D-1.2: Schema

In the context of having to adapt to new requirements
facing the need to quickly adjust the database schema
we decided to use a NoSQL data store
and neglected a relational database
to achieve the possibility to change the database schema
accepting the downside of a not having a strict database schema

For more detailed information, see section 4.3.

D-1.3: Data Store

In the context of storing test data including metadata
facing the need to provide extensive and flexible querying options
we decided to use Elasticsearch
and neglected RavenDB, CouchDB and MongoDB
to achieve fast and powerful search capabilities
accepting the downside of slow insert operations.

For more detailed information, see section 4.3.

D-1.4: Search Engine

In the context of querying and filtering test data including metadata
facing the need to query thousands of entries
we decided to use Elasticsearch
and neglected MongoDB
to achieve fast querying
accepting the downside of learning a new query language.

For more detailed information, see section 4.3.

D-2.1: Data Consumer Programming Language

In the context of implementing the data consumer
facing the need to seamlessly integrate into the existing CI/CD pipeline
we decided to use Python
and neglected other languages like C#, Java, ...
to achieve minimal maintenance cost
accepting the downside of learning a new programming language **and** the lower performance of an interpreted language.

For more detailed information, see section 4.2.

D-3.1: Frontend Technology

In the context of implementing a Frontend

facing the need to visualize and manipulate test data

we decided to create a custom frontend

and neglected a Kibana or Graphana only approach

to achieve the possibility to manipulate test data

accepting the downside of an increased development effort.

For more detailed information, see section 4.4.

D-3.2: Web Framework

In the context of implementing a Frontend

facing the need to visualize and manipulate test results

we decided to use Angular

and neglected React.js, Vue.js, ...

to achieve implementing a technology already known and used at Sonova

accepting the downside of learning a more complex technology.

For more detailed information, see section 4.5.

D-4.1: Backend

In the context of implementing the Test Result Viewer

facing the need to provide authentication and prevent security risk

we decided to not implement the backend

and neglected the possibility for features like authentication and data caching

to achieve simplicity in the project setup and focus on implementing visualizations

accepting the downside of of some features not being available and increased security risks.

This decision was made together with Sonova on our weekly meeting (see section E). For more detailed information, see subsection 4.6.4.

D-5.1: Reverse Proxy Technology

In the context of implementing a Reverse Proxy

facing the need to secure access to the frontend, Kibana and Elasticsearch

we decided to implement Nginx

and neglected other proxy solutions like Caddy and Traefik

to achieve consistency with the existing infrastructure at Sonova

accepting the downside of a more complex configuration compared to other solutions.

For more detailed information, see subsection 4.7.1.

6. Implementation

In this chapter we describe how we have built the application. We start with the development server setup in section 6.1.

Section 6.2 describes how we built the data consumer, section 6.3 explains the database schema and how Elasticsearch is set up. The used technologies and libraries for the frontend are documented in section 6.4, and finally section 6.5 describes the setup for the reverse proxy and the different routes.

All code for this project is centrally stored on the OST GitLab server inside a Test Result Viewer group, available at https://gitlab.ost.ch/test-result-viewer.

6.1. Development Server

To set up different prototypes and our final development server, we requested a virtual machine running on OST infrastructure. It is running the Ubuntu 22.04.2 LTS release. Using a local Ansible setup on a developer's machine, setup of docker can be automated easily using instructions available online [5]. The playbook we used can be found inside the documentation repository[1] as `01-pb-docker.yml`

The commands used for a simple docker deployment of Elasticsearch and Kibana as single containers are documented in the following shell script in the same repository folder:
`elastic-prototype/02-elastic-kibana-single-container.sh`

This should be enough for quick tests using a fresh deployment. Elasticsearch user credentials as well as the enrollment token to configure Kibana must be read from the container output.

To reset the elastic user password, the following command can be used to start the `elasticsearch-reset-password` utility inside a running container:

```
docker exec -it es01 /usr/share/elasticsearch/bin/elasticsearch-reset-password
```

6.2. Data Consumer

For the implementation of the Data Consumer[2] using Python we were able to use a package template provided by Sonova. It includes scripts to build and test the code.

Instructions to use the package can be found in the `README.md` file inside the repository. Command-line flags differentiate between the types of input files. Listing 6.1 shows their usage.

As Figure 6.1 shows, the insertion part of the Consumer relies on the defined entities on one hand, and on the relevant code needed for the case it is executed for: either generating new data or parsing one of the supported formats.

[1] https://gitlab.ost.ch/test-result-viewer/documentation/-/tree/main/_additional-documents/server-setup
[2] https://gitlab.ost.ch/test-result-viewer/consumer

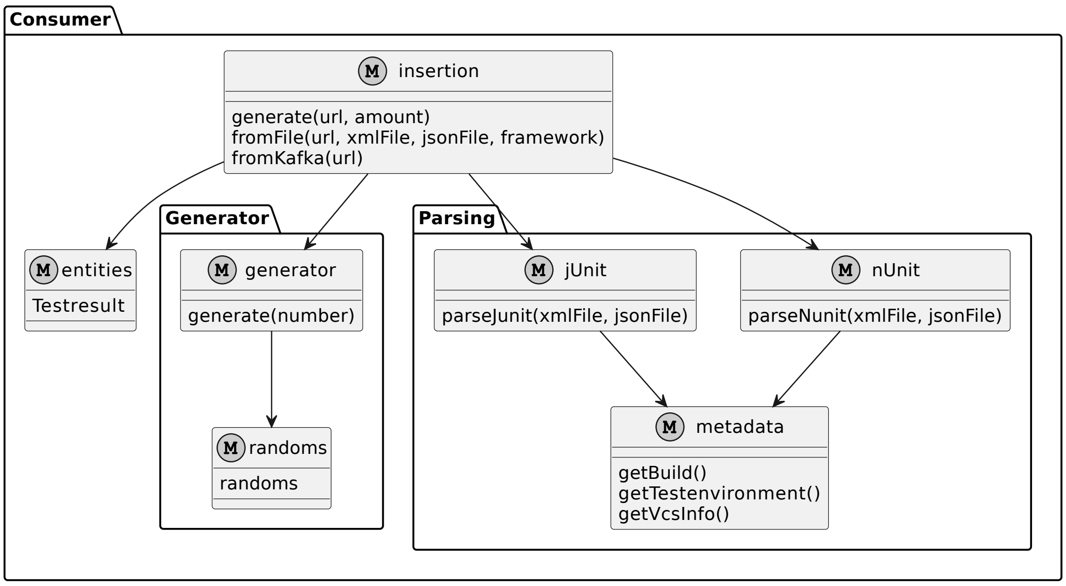


Figure 6.1.: Consumer Structure

A usage statement for the python package is displayed when the package is called without any arguments.

Listing 6.1: Python Package Usage

```
Usage: ./main.py https://user:password@elastic-url:port
   --generate <number>                 amount of results to generate
   --junit <resultfile> <metadata>     parse resultfile and medatadata as jUnit
   --nunit <resultfile> <metadata>     parse resultfile and metadata as nUnit
```

6.2.1. JSON Metadata Schema

The metadata file provided by Sonova has a schema as described in Listing 6.2.

Listing 6.2: Meta data schema

```
{
  "build": {
    "brand": string,
    "identifier": string,
    "number": number,
    "project": string,
    "trigger": string,
    "url": string
  },
  "version-control-system-info": {
    "branch": string,
    "commit-hash": string,
    "commit-time": string, # ISO format e.g 2023-04-12T23:59:59.000000Z
    "commit-message": string,
    "commit-author": string,
    "repository": string,
    "type": string,
  }[],
  "test-environment": {
    "name": string
```

```
      "capabilities": string[]
    }
  }
```

6.3. Data Store and Kibana Frontend

Our development setup of Elasticsearch and Kibana runs in multiple containers, using docker compose to orchestrate. The next chapters describe the configuration required to run Elasticsearch as well as the detailed docker compose setup. The configuration can be found in the elasticsearch git repository[3].

6.3.1. Memory settings for Elastic in Docker

Running Elasticsearch has some memory requirements. The official documentation states:

> *Elasticsearch uses a mmapfs directory by default to store its indices. The default operating system limits on mmap counts is likely to be too low, which may result in out of memory exceptions [45].*

These settings have been adjusted as part of the Ansible playbook mentioned in section 6.1:

Listing 6.3: Ansible task to update vm.max_map_count

```
- name: update vm.max_map_count setting
  ansible.posix.sysctl:
    name: vm.max_map_count
    value: '262144'
    state: present
```

They can be modified from the command line using `sysctl -w vm.max_map_count=262144`. To make the change permanent, update the `max_map_count` value in `/etc/sysctl.conf`.

6.3.2. Running the cluster with Docker Compose

Our final development deployment uses three containers for Elasticsearch and one for Kibana. It is available at https://testresults.kuendig.dev [40].

Additionally, when setting up the cluster for the first time, two containers are created for different initialization tasks. The container `elastic-setup` is used to create the certificates needed for intra-cluster communication. This configuration was provided with the Elasticsearch example implementation [40] and is visible inside the docker compose file for the cluster (D.5).

`elastic-init` runs a bash script (D.6) once the Elasticsearch containers are ready. We created this script to generate search templates (see 6.3.6), index mappings (6.3.5) and ingest pipelines (6.3.7). The container is built directly from the file system using a custom dockerfile to include the tools needed (D.7).

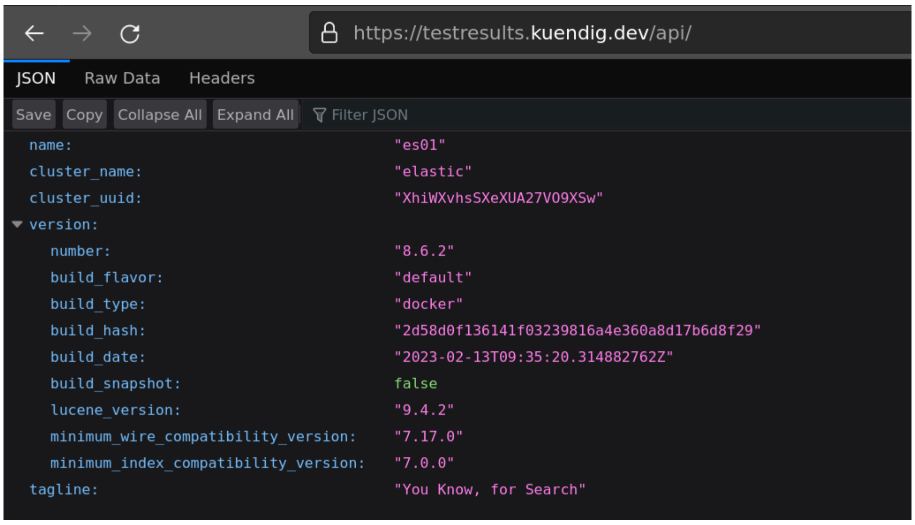
To encrypt client connections to our server instance, we have generated a free ZeroSSL certificate and made the configuration in Kibana for the cluster to use it. Certificate files are mapped into the docker container using a docker bind mount [19].

To access a shell inside a container, we can use the following command:

```
docker exec -it <container-name> /bin/bash
```

[3] https://gitlab.ost.ch/test-result-viewer/elasticsearch

Figure 6.2 shows the default response the Elasticsearch API returns when accessed via URL.



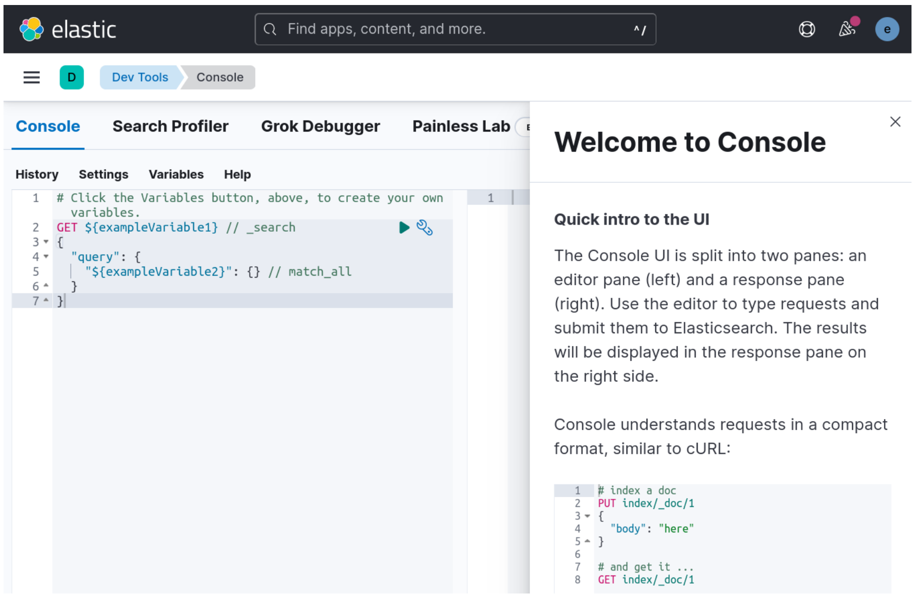
```
name:                                  "es01"
cluster_name:                          "elastic"
cluster_uuid:                          "XhiWXvhsSXeXUA27VO9XSw"
version:
  number:                              "8.6.2"
  build_flavor:                        "default"
  build_type:                          "docker"
  build_hash:                          "2d58d0f136141f03239816a4e360a8d17b6d8f29"
  build_date:                          "2023-02-13T09:35:20.314882762Z"
  build_snapshot:                      false
  lucene_version:                      "9.4.2"
  minimum_wire_compatibility_version:  "7.17.0"
  minimum_index_compatibility_version: "7.0.0"
tagline:                               "You Know, for Search"
```

Figure 6.2.: Elasticsearch API Response

6.3.3. Kibana Dev Tools

After the Elasticsearch Cluster has been successfully set up with Kibana as a frontend, there is an easy way to send requests to the Elasticsearch API.

In the Kibana Menu find the `Management / Dev Tools` section for an interactive page to run any HTTP commands against the connected Elasticsearch instance.



```
# Click the Variables button, above, to create your own
  variables.
GET ${exampleVariable1} // _search
{
  "query": {
    "${exampleVariable2}": {} // match_all
  }
}
```

Welcome to Console

Quick intro to the UI

The Console UI is split into two panes: an editor pane (left) and a response pane (right). Use the editor to type requests and submit them to Elasticsearch. The results will be displayed in the response pane on the right side.

Console understands requests in a compact format, similar to cURL:

```
# index a doc
PUT index/_doc/1
{
  "body": "here"
}

# and get it ...
GET index/_doc/1
```

Figure 6.3.: Kibana Dev Tools

6.3.4. Elasticsearch Indexes

An index in Elasticsearch stores one type of documents. The Test Result Viewer currently uses three different indexes:

The `testresults` index presented in Listing 6.4 stores the main data created when importing result files. It should be agnostic of the framework used and present all data in a unified way.

Taking the limitations from subsection 4.1.1 into account we created the current database schema.
The other indexes `assignment` and `comment` (Listings 6.5 and 6.6) serve to assign more information at the test case level.

Listing 6.4: Testresults Index

```
{
  "testcase": string,
  "testsuite": string,
  "testassembly": string,
  "description": string,
  "start-time": string # ISO format e.g 2023-04-12T23:59:59.000000Z,
  "build": {
    "brand": string,
    "identifier": string,
    "number": number,
    "project": string,
    "trigger": string,
    "url": string
  },
  "vcs-info": [
    {
      "branch": string,
      "commit-hash": string,
      "commit-time": string, # ISO format e.g 2023-04-12T23:59:59.000000Z
      "repo": string,
      "type": string
    }
  ],
  "test-environment": {
    "name": string,
    "capabilities": string[]
  },
  "environment": {
    "framework": string,
    "framework-version": string,
    "clr-version": string,
    "os-version": string,
    "platform": string,
    "cwd": string,
    "machine-name": string,
    "user": string,
    "user-domain": string
  },
  "culture-info": {
    "current-culture": string,
    "current-uiculture": string
  },
  "executed": boolean,
  "result": string,
  "time": decimal,
  "asserts": number,
  "categories": string[],
  "properties": [
    {
      "name": string,
      "value": string
    }
  ],
  "reason": string,
  "failure": {
    "message": string,
    "stacktrace": string
  }
}
```

Listing 6.5: Assignment Index

```
{
  "assignment": string,
  "testsuite": string,
  "testcase": string
}
```

Listing 6.6: Comment Index

```
{
  "comment": string,
  "testsuite": string,
  "testcase": string
}
```

6.3.5. Index Mappings

During development, we used dynamic mapping for the testresults index. This means that Elasticsearch dynamically determines the type of fields based on the contents and optimizes accordingly [31]. This is suitable for creating a prototype. However, as a developer you know more about your data than Elasticsearch. Using explicit mapping, you can tell Elasticsearch what kind of data each field contains and how it should process it [20]. This gives you the benefits of optimized queries and search results.

The mapping is created during execution of the Elasticsearch setup using the `init.sh` script. The exact mapping for each index is available in the Elasticsearch repository[4]. For more information regarding Elasticsearch initialization, see subsection 6.3.2.

6.3.6. Search Templates

All Elasticsearch queries in the application are performed using so-called Search Templates. They are comparable to Prepared Statements in a RDBMS, in the sense that they can be used to separate queries from the application logic and keep them inside the data store component [39]. This solution has the benefit that you can change a query without changing the frontend code.

6.3.7. Ingest Pipelines

The functionality of Elasticsearch Ingest Pipelines (see Figure 4.3) allows to operate on data whenever it enters the cluster. In our case it was easy to define some rules to classify and if necessary convert incoming data wherever it was not correctly recognized by the index mapping. For this we mainly used the convert processor [10].

[4]https://gitlab.ost.ch/test-result-viewer/elasticsearch/-/tree/main/init/definitions/index/mapping

6.3.8. Elastic Query Types

The following listings serve as examples for different types of requests in Elasticsearch [37]. The difference between filters and queries is the logic of returning results. While a filter effectively removes all data that does not match the criteria exactly, a query returns data with a score on how well it matches the given criteria.

In Listing 6.7 a search query is performed using a single criterion.

Listing 6.7: Match: single criterion filtering

```
GET /testresult/_search
{
  "query": {
    "match": { "testcase": "test case 18" }
  }
}
```

Instead of a single hard criterion, you can also submit a query with multiple criterions and different match semantics. An example for such a query is Listing 6.8.

Listing 6.8: Boolean: combine multiple filtering criteria

```
GET /testresult/_search
{
  "query": {
    "bool": {
      "must": [
        { "match": { "testcase": "test case 18" } }
      ],
      "should": [
        { "match": { "testsuite": "suite2"} }
      ]
    }
  }
}
```

Sometimes it is required to get all distinct values for a specific field while the other fields are not important. This problem can be solved using a collapse query. An example for this type of query can be seen in Listing 6.9.

Listing 6.9: Collapse: Get all distinct values for different fields

```
GET /testresult/_search
{
 "size": 10,
 "collapse": {
   "field": "testcase.keyword"
 },
 "sort": {
   "testcase.keyword": {
     "order": "desc"
   }
 },
 "_source": []
}
```

Often you are not interested in a simple collection of documents, but their aggregation over a specific field. The example in Listing 6.10 shows a query, which groups all documents by their start time.

Listing 6.10: Aggregation: Get documents grouped by start time

```
GET /testresult/_search
{
  "size": 0,
  "aggs": {
    "byStartTime": {
      "terms": {
        "field": "start-time",
        "size": "100",
        "order": {
          "_key": "desc"
        }
      }
    }
  }
}
```

6.4. Angular Frontend

The custom frontend for this application is written using Angular and stored in a separate git repository[5]. The following sections describe a few key aspects of the implementation.

6.4.1. Code Documentation

The documentation for the Angular frontend is generated and deployed during the CI/CD pipeline using compodoc [9]. The generated documentation can be accessed at http://test-result-viewer.pages.gitlab.ost.ch/ba2023/doc/.

6.4.2. Code Coverage Report

The frontend is automatically tested for every commit that is pushed to the repository. After the test were run, the coverage report is published at http://test-result-viewer.pages.gitlab.ost.ch/ba2023/coverage/angular-frontend/

6.4.3. Reactive Programming using RxJS

In our frontend we need a way to deal with data being asynchronously fetched from Elasticsearch and other sources. Angular handles this problem using RxJS, a reactive programming library. With the usage of such a library it is important to follow some best practices in order to avoid performance problems and memory leaks [21]. Detailed descriptions of the RxJS components can be found in their official documentation[6].

6.5. Nginx Reverse Proxy

The Nginx reverse proxy setup contains three different paths to forward requests, as described in section 5.5

The `compose.yaml` (D.1) and `nginx/nginx.conf` files in the frontend repository contain all information relevant to the setup. Table 6.1 lists the three destinations.

[5]https://gitlab.ost.ch/test-result-viewer/ba2023/-/tree/main/frontend
[6]https://rxjs.dev/api

Route	Component	Destination
`/`	Angular Frontend	`host.docker.internal:4200`
`/api`	Elasticsearch	`host.docker.internal:9200`
`/kibana`	Kibana	`host.docker.internal:5601`

Table 6.1.: Nginx proxy destinations

7. Quality Measures

This chapter focuses on the project as a whole and the measures we have applied to ensure a positive outcome in any aspect. In section 7.1 we describe the conventions we used to keep the code consistent. To detect any problems as soon as possible, we used various testing and linting tools described in section 7.2. To ensure the tests are actually executed on a clean environment, we had setup an CI/CD piplie which is describe in section 7.3. In the section 7.4 we track, the progress of the NFRs.

7.1. Conventions

For the frontend we follow the Angular Coding Style Guide [4]. The application is structured according to the SCAM approach [33].

7.2. Testing

To ensure that we do not introduce bugs unintentionally we wrote tests for the consumer as well as for the frontend. The used tools are documented in Table 7.1.

Component	**Purpose**	**Framework**	**Justification**
Consumer	Testing	pytest	Framework used by Sonova and already configured in the template provided by Sonova
Consumer	Linting	isort, black, flake8, mypy, add-trailing-comma, autoflake	Tools used by Sonova and already configured in the template provided by Sonova
Frontend	Testing	Jasmine	Default testing framework shipped with Angular 15
Frontend	Linting	eslint with Angular configuration	Default linting tool and configuration shipped with Angular 15

Table 7.1.: Used testing frameworks

7.3. CI/CD Pipeline

In our repositories we use an automatic CI/CD Pipeline to ensure the quality of all changes that are pushed to the server.

A visualization of the pipeline configuration for each repository can be seen in the figures 7.1 to 7.3.

The pipeline for the frontend

- lints the frontend
- tests the frontend

- publishes a code coverage report
- publishes the Angular documentation.

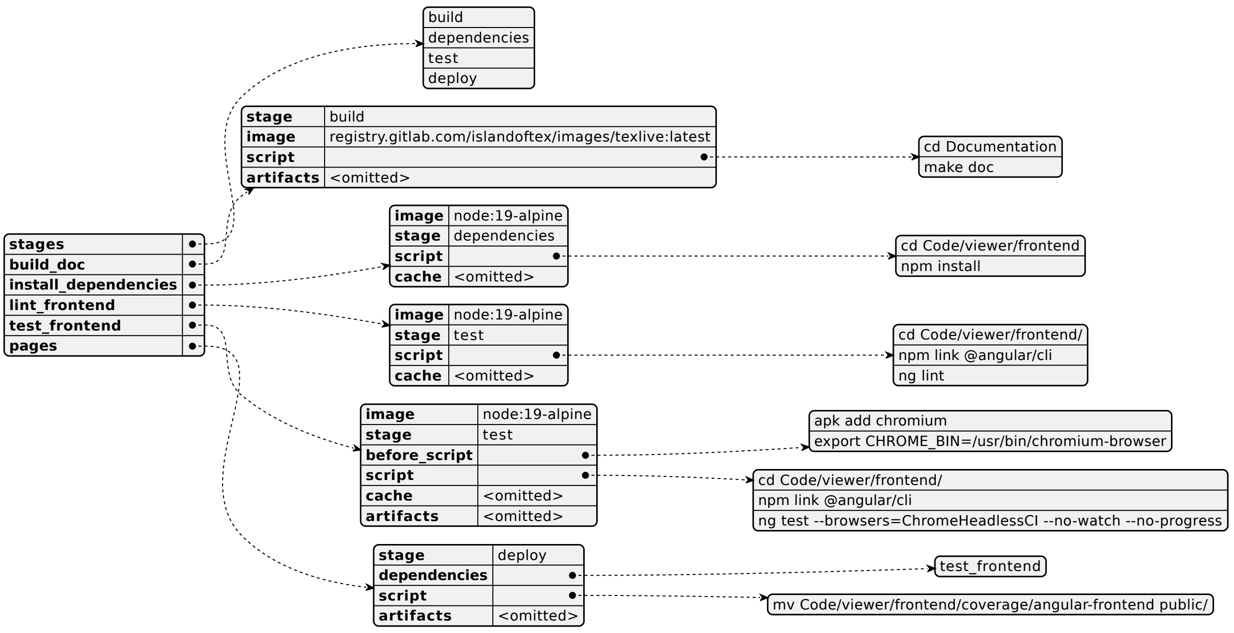


Figure 7.1.: Frontend pipeline YAML visualized

The pipeline for the consumer

- builds the package
- tests the package.

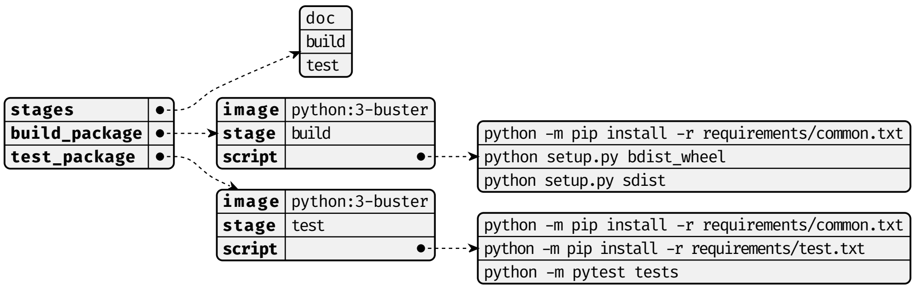

doc
build
test
stages
build_package
test_package
image | python:3-buster
stage | build
script
python -m pip install -r requirements/common.txt
python setup.py bdist_wheel
python setup.py sdist
image | python:3-buster
stage | test
script
python -m pip install -r requirements/common.txt
python -m pip install -r requirements/test.txt
python -m pytest tests

Figure 7.2.: Consumer pipeline YAML visualized

The pipeline for the documentation

- builds the documentation.

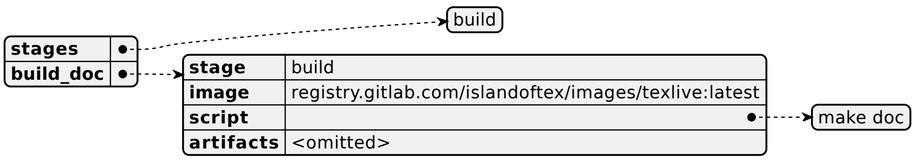


Figure 7.3.: Documentation pipeline YAML visualized

7.4. Quality Tracking

7.4.1. Nonfunctional Requirements

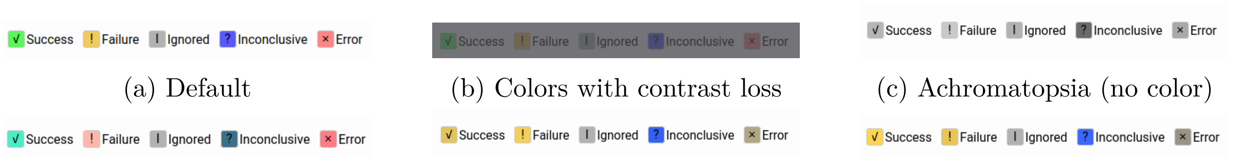
Nonfunctional requirements of this project as defined in section 3.3 are tracked as described in Table 7.2.

NFR-1 Performance	Query time can be observed while developing. Default query filter for views should limit the time range to at most 7 days.
NFR-2 Confidentiality	TLS Setup of the elastic cluster as well as the nginx proxy has been completed with the initial setup. When the project is finished, directions on how to set up new certificates at Sonova must be provided.
NFR-3 Browser Support	Developers must use the browser with the highest priority while developing the frontend. Other browsers should be tested when time and scope allows.

Table 7.2.: Tracking of NFRs

7.4.2. Considerations for color vision deficiencies

As some employees at Sonova may suffer from different color vision deficiencies, it is advisable that the application is usable even with such a restriction. To ensure that the symbols are distinguishable from the background color, we used Firefox' functionality of *Color vision simulation* [8]. The results can be seen in Figure 7.4.



(a) Default (b) Colors with contrast loss (c) Achromatopsia (no color)
(d) Tritanopie (no blue) (e) Deuteranopia (no green) (f) Protanopia (no red)

Figure 7.4.: Results of the color vision deficiencies simulation

The results show that in case of problems regarding the distinction of colors, the icons we have included in the boxes make the distinction clear. In the case of contrast loss even the icons are hard to distinguish. However we recommend this should only be addressed in case someone really suffers from this condition, otherwise it would add unnecessary effort.

8. Results

The main goal of the project has been the implementation of a central data store that unifies different formats of test results by using a Data Consumer package for transformation. Several visualizations should be realized in a protoype manner.

All requirements listed as Minimum Viable Product are fulfilled by the final product, as well as some extended requirements which partly require the modification of data from the frontend. Specifically these are Hierarchical test order, Assigning a label to a test result and Attach a comment to a test case.

9. Conclusion

The scope of our thesis has not by far been enough to cover all requirements that Sonova has envisioned for this platform, so there are many things that can be improved and extended in the future. We will not specify any of these unimplemented requirements here, as they are listed in detail in the appendix.

However the one component we do think should be implemented in order to make the product more adaptable and also more stable is a dedicated backend. It would enable many improvements and extensions in terms of security and reliability, while further decoupling the Angular frontend from the Elasticsearch data store.

Focusing on the current architecture and the technologies used, they have proven themselves to be a very powerful and flexible combination for the overall implementation of the Test Result Viewer. There usually were several possibilities in how to achieve some given task and it was our challenge to find the most suitable one. We hope that our choices will prove themselves useful and that Sonova will develop our product into a powerful and flexible frontend that enables their developers to spend more time writing code.

Part II.

Project Documentation

1. Project Proposal

This chapter contains the assignment as we received it at the beginning of this project.

1.1. Introduction

This bachelor thesis is conducted for the external partner Sonova AG.

> *Sonova is a global leader in innovative hearing care solutions: from personal audio devices and wireless communication systems to audiological care services, hearing aids and cochlear implants. The Group was founded in 1947 and is headquartered in Stäfa, Switzerland [44].*

The topic of test result visualization seems to have been oversimplified by most, if not all, Continuous Integration (CI) systems. If all tests can be run before introducing a change to a source code repository, it can be guaranteed that all tests remain green on the main branch of the repository, which yields that no advanced test result visualization is required.

Unfortunately, there are many real-life examples where this approach fails to deliver a convenient working environment to software developers. There are many aspects which might make this simple case difficult to manage eventually.

Factors that influence the fast, reliable test execution to guarantee retaining the pristine state of the entire system:

- Multiple projects per repository in various configurations and combinations
- Very long-running tests that prohibit enforcing pre-merge tests
- Non-deterministic tests due to variations of execution timing on the hardware
- Spurious failures of the infrastructure for running tests
- Flaky hardware for hardware-dependent tests

All these factors might lead to a system that would be extremely difficult to manage without proper tooling. In the absence of a suitable off-the shelf solution for advanced test result visualization, Sonova plans to build its own test management platform to properly aggregate and present the results of the automated test runs.

1.2. Goals of the project

In this bachelor thesis the students shall develop an MVP for aggregating and visualizing test results. A set of requirements for this system has been collected beforehand. It is infeasible to develop a complete platform that could satisfy all of them in the limited time of this project. Therefore, the focus lies on implementing a solid foundation with a subset of the required features that can be extended later.

Defining the feature set to be implemented is one of the first tasks and must happen in collaboration with Sonovas Development Environment team and the supervisor. Refinement of the delivered features is possible, depending on the teams progress. All architectural decisions and selection of technologies needs to happen in agreement with Sonova, as the system context is already well defined, consisting of multiple external systems to interact with, for example:

- Jenkins, running the build jobs.
- Artifactory, storing the build artifacts.
- Github, hosting the code repositories.

The progress of the team will be discussed in weekly meetings with the supervisor. The students are responsible for preparing these meetings and protocolling decisions. Access to confidential or sensitive internal data of Sonova is not necessary to conduct the project.

2. Project Planning

This project follows the well known scrum methodology for software engineering wherever possible, but tries to reduce it to a level of simplicity fitting for a team size of two developers [49].

2.1. Team

The roles assigned in this project:

Developers:	Olivier Lischer, Luzia Kündig
Product Owner:	Dariusz Danilko
Scrum Master:	none

The main responsibilities for the developers are divided into

Code & Functionality:	Olivier Lischer
Documentation & Architecture:	Luzia Kündig

This does not mean a strict division of tasks, but a more general focus to keep everything up to date.

2.2. Sprints and Meetings

The definitions of our sprint meetings and schedules:

Sprint Length:	1 Week
Sprint Planning:	Together with our advisor and as needed the external partners at Sonova. Every week on Monday, 1pm
Daily Scrum:	Not planned
Sprint Review:	Every week on Monday, after Sprint Planning
Sprint Retrospective:	Not planned

Because of the small development team size and our different schedules, we decided against planning a Daily Scrum, Sprint Reviews and Sprint Retrospective meetings. For the same reasons, the position of Scrum Master is not defined.

However, we are in regular exchange over Microsoft Teams or in person. If there is anything to discuss besides the weekly spring planning, a meeting can be scheduled.

2.3. Git Workflow

For the scope of our project, we separate our work into 2 different branches on git.

The ***main*** branch only contains finished items of work.
The ***development*** branch is used for active code development.
Every Sunday before the weekly meeting, all branches should be merged into main to represent the current finished state.

2.4. Tools

The following tools enable us to collaborate on this project.

Issue tracking:	GitLab [1]
Time tracking:	Toggl Track [2]
Communication:	Microsoft Teams

2.5. Project Plan

The project was planned using the RUP method. RUP consinst of four phases:

Inception: setup repositories, define responsibilities

Elaboration: build prototypes, perform risk analysis

Construction: building the actual product, biggest risk should be already eliminiated

Transition: finish product and performing transfer to customer

The project plan for this project can be seen in Figure 2.1.

[1] https://gitlab.ost.ch/test-result-viewer/
[2] https://track.toggl.com/

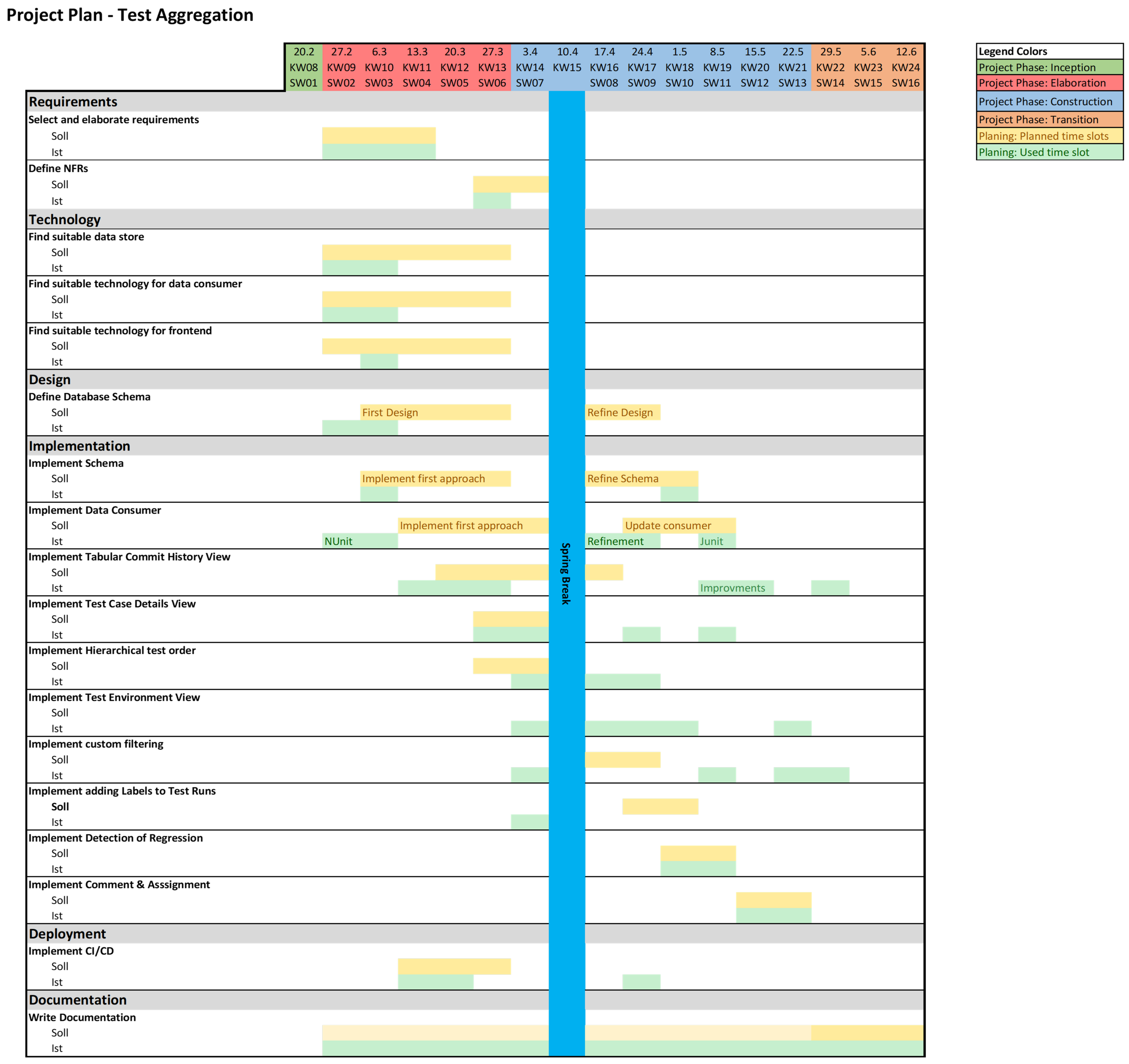


Figure 2.1.: Project Plan

2.6. Milestones

The following milestones have been planned without an exact date. This would have been very difficult since we did not really know any of the tools beforehand. However, in our weekly meetings with Thomas Corbat and Dariusz Danilko we are able to keep a good track on our progress and adjust plans immediately if necessary. Sonova is always up to date with our progress and can give inputs.

Milestone	Description	Week achieved
M1	Initial database schema	Week 01
M2	Prototype using Kibana	Week 03
M3	First visualisation in Angular frontend	Week 05
M4	Minimum Viable Product	Week 07
M5	Provide more functionality than the existing solution	Week 12

2.7. Risk Assessment

In this section we describe the risk we encounter and how we mitigated them.

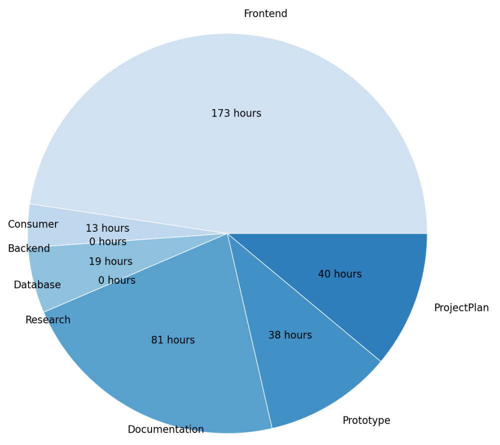
1. New Technologies: Both students have never worked with the proposed tools Elasticsearch, Kibana and Angular.
   Category: Low
   - March 01: Plan enough time to build prototypes and get familiar with the tools.
2. Extensive list of requirements: Many use cases have been specified by the Sonova team. It could prove a challenge to identify and focus on the most important ones.
   Category: Low
   - March 01: Frequent exchange with the stakeholders to verify the current focus is appropriate.

3. Time Tracking Report

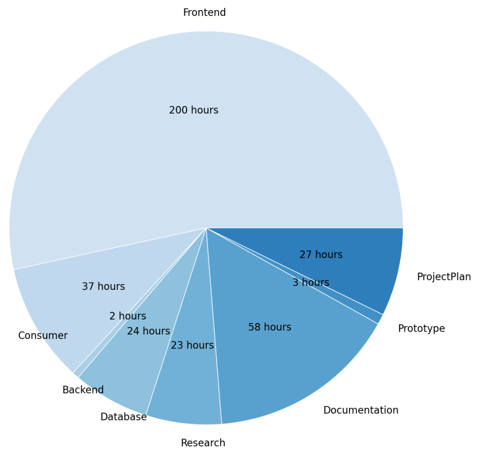
In this chapter we provide an overview of where and how much time we spent for this project.

3.1. Time per Category

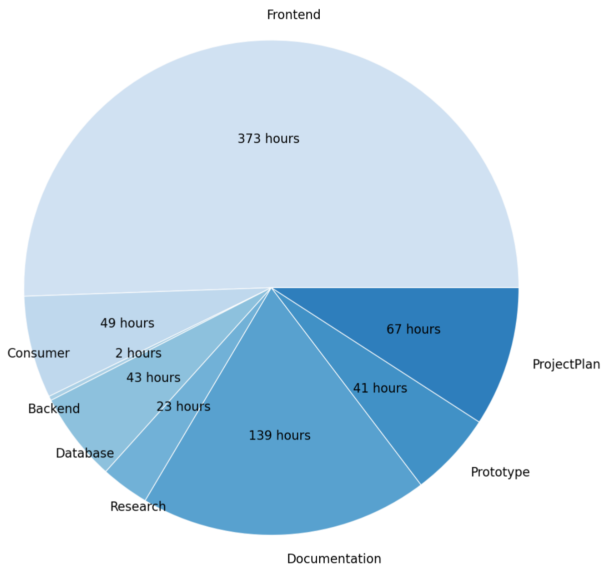
In Figure 3.1 you see how we spent our time grouped by categories.



(a) Time per category by Luzia Kündig



(b) Time per category by Olivier Lischer

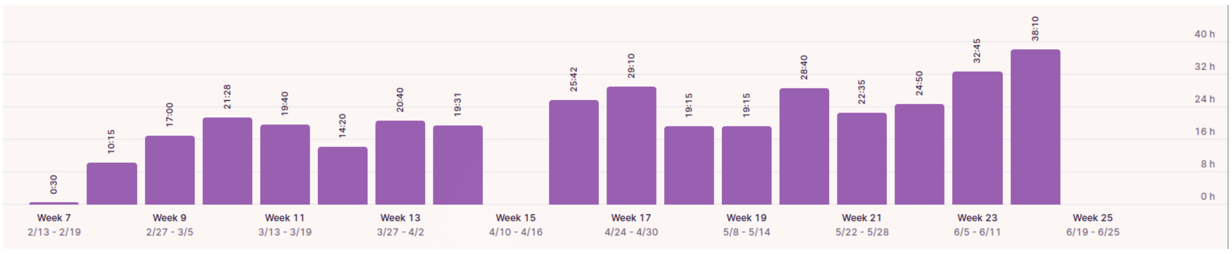


(c) Time per category by Team

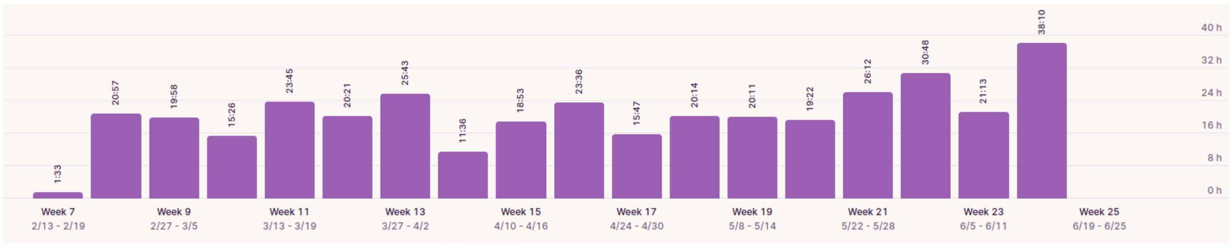
Figure 3.1.: Time per category

3.2. Time per Week

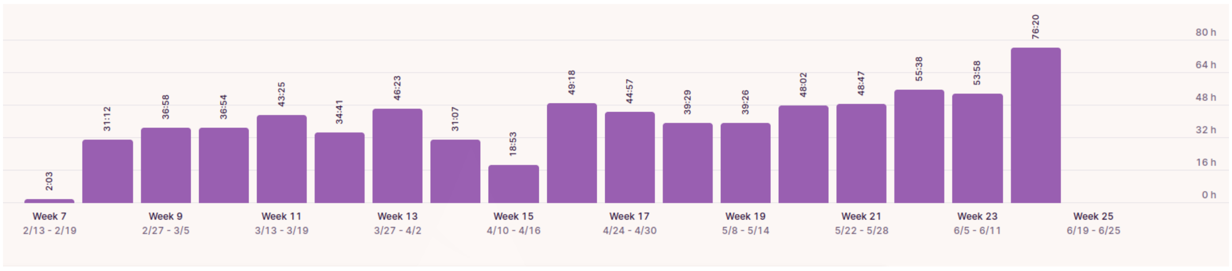
In Figure 3.2 you can see how much time we spent per week.



(a) Time per week by Luzia Kündig



(b) Time per week by Olivier Lischer



(c) Time per week by Team

Figure 3.2.: Time per week

Part III.

Appendix

A. Requirements as provided by Sonova

Views

Req-1.1: Customizable View

The system shall provide the tester the ability to create a customized views.

Req-1.2.1: History of a test case

The system shall provide the tester the ability to create a views that show the history of a test case result over time.

Req-1.2.2: History of a test environment

The system shall provide the tester the ability to create a views that show the history of a the count of all, failed, errored and skipped tests for a given test environment.

Req-1.3: Customizable view elements

The system shall provide the tester the ability to customize the view elements. For example the columns in a table.

Req-1.4: Filter are applied to all views

The system shall apply the selected filters on all views.

Req-1.5: Switch views

The system shall allow the tester to change between different views without losing applied filters.

Req-1.6: Hierarchical test order

The system shall provide the tester the ability to show the tests in a hierarchical order and on each hierarchy level the count of all, failed, errored and skipped tests.

Req-1.7: Show running tests

The system shall provide the tester a view that shows the currently running jobs which will in the future provide test results that matches the filter criteria.

Req-1.8: Show queued tests

The system should provide the tester a view that shows the queued jobs which will in the future provide test results that matches the filter criteria.

Req-1.9: Show test case details

The system shall be able to show for an executed test the following details:

- if the test is queued
  - version control system details
  - the URL to the test job
  - the job trigger event (time-based, another service or manually). If the job was triggered manually, then the name of the user should be shown.
- and if the test has started
  - test execution start datetime
  - estimated remaining time
  - the relation to the used test environment
  - additional metadata, for example, build and test configuration
- and if the test has finished
  - test execution duration
  - test execution end datetime
  - test failures and errors if available

Req-1.10.1: Show GitHub details

In addition to Req-1.9: Show test case details, the system should be able to show

- the GitHub pull request URL if available
- the GitHub repository URL

Req-1.10.2: Show the logs of the test execution

In addition to Req-1.9: Show test case details, If a test was executed, the system should be able to show the logs, or a URL to the relevant line in a log file, of the test execution.

Req-1.10.3: Show additional output of the test execution

In addition to Req-1.10.2: Show the logs of the test execution, If a test was executed, the system should be able to show the content of additional files, or a URL to the relevant line in a file, of the test execution.

Req-1.10.4: Show the test owner

In addition to Req-1.9: Show test case details, the system should be able to show the test owner, see Req-4.1: Test owner.

Req-1.10.5: Provide the URL to the test statistics

In addition to Req-1.9: Show test case details, the system should be able to provide the URL to the test statistics. see Req-5.5: Link test case executions with statistics

Req-1.11: Show build details

The system shall show the build details if the unit under test was a build.

Data structure and filtering

Req-2.1: Labels

The system shall give the tester the ability to create, assign and un-assign labels to/from test runs.

Req-2.2: Add labels to multiple tests at once

The system shall give the tester the ability to assign labels to all test cases matching the filter query or search result.

Req-2.3: Filters

The system shall give the tester the ability to filter tests by

- by test case
- by time range
  - test execution start time
  - test execution duration
- by owner
- by labels
- by VCS branch
- by job id

Req-2.4: Combination of filters

The system shall give the tester to ability to combine any of the filters with

- logical operations AND, OR, NOT
- subset inclusion (contains element x in sequence y)
- subset exclusion (contains not element x in sequence y)
- substring condition (string x is substring of string y)

to narrow-down the search.

Req-2.5: Autocompletion support for filters

The system shall support the tester with autocompletion suggestions for filters while typing.

Req-2.6: Search

The system shall give the tester the ability to search with regex pattern

- for test case by name
- VCS commit message
- in test run log files
- in error and failure messages
- in test case meta data

Req-2.7: Filter and search persistence

The system shall give tester the ability to save an applied query or search in the system and the ability to reapplied saved queries and searches. Further, the system shall allow the tester to remove saved queries and searches.

Req-2.8: Search history

The system shall save the last 20 searches and present them when the tester opens the search context.

Req-2.9: Reset all applied filters and searches

The system shall give tester the ability to reset all applied searches and filters.

Req-2.10: Marking some test results as irrelevant / deleting them from the database

The system shall give tester the ability to mark specific test runs as irrelevant or even allow deleting such test runs from the database in case it was discovered that the results of specific test runs are not reliable. This requirement can be covered by Req-2.1: Labels

Data analysis

Req-3.1: Detection of regression

The system shall detect new regression automatically and provide this information to the tester.

Req-3.2: Detection of flickering tests

The system shall detect flickering tests automatically and provide this information to the tester.

Req-3.3: Detection of test environment related failures

The system shall detect when tests fail because of a broken test environment and provide this information to the tester.

Req-3.4: Advanced detection of test failures

The system should detect when many test fail with a similar reason.

Test case management

Req-4.1: Test owner

The system shall allow the user to create, change, remove and assign test owners to test cases.

Req-4.2: Comments on test case execution

The system shall allow the user to create, change and remove comments on test case executions.

Interaction with other systems

Req-5.1: Link test cases to Jenkins jobs

The system shall give the tester the ability to link test case executions with a uniquely identifiable Jenkins job ID.

Req-5.2: Link test cases to TeamCity

The system shall give the tester the ability to link test case executions with a uniquely identifiable TeamCity job ID.

Req-5.3: Link test cases to Jira issues

The system shall give the tester the ability to link test cases with Jira issues.

Req-5.4: Link test cases to Polarion items

The system shall give the tester the ability to link test cases with Polarion items.

Req-5.5: Link test case executions with statistics

The system shall give the tester the ability to link test case executions with generated statistics.

Req-5.6: Link test case executions with artifacts on Artifactory

The system shall give the tester the ability to link test case executions with relevant artifacts on Artifactory.

Data Export / Sharing

Req-6.1: Sharing of views and filters

The system shall allow the tester to share a view and the currently applied filters using a URL with query parameters

Req-6.2: Exporting a view to a file

The system shall support exporting test results to a file (.xlsx, CSV or similar)

Access

Req-7.1: Access from the internet

The system shall be accessible from the internet without a VPN connection.

Req-7.2: Single Sign On (SSO) support

The system shall support SSO.

Req-7.3: Provide a RESTful API

One could fill another page with just requirements for this API. To keep it short: The REST API should support the requirements defined here where it makes sense.

Non-Functional

Req-8.1: Query response time

The system shall respond to a filter query within 1s and perform a full filter query in less than 2s.

Test environment management

It is currently not clear if this is in the scope of the test viewer/manager system.

Req-9.1: Manage test environments

The system shall be able to manage a list of all available test environments with a detailed list of their capabilities and their current state. The state can be online/operational, offline or broken. The capabilities include the operation system type and version, connected hardware (name and version) and the installed software (name and version).

Req-9.2: Log for test environments

The system shall provide a log where the test environment manager can enter log messages.

Req-9.3: Take test environments offline

The system shall test environment manager to ability to take a machine offline and optionally mark it as broken.

Req-9.4: Statistics for usage

The system shall provide the test environment manager the statistics for the usage of certain test environment and the requested capabilities.

Storage

Req-10.1: Backup

The system shall provide means to backup and restore data from backups

Req-10.2: Retention policy

The system shall provide the tester the ability to define different retention policies for test case results. The retention policy shall be configurable as a duration (days, weeks, months) or as never, meaning the test result is kept forever.

Req-10.3: Archive

The system shall provide the tester the ability to archive test results as ZIP files. Further, the system shall allow to reload test results from ZIP files.

B. Main use cases to cover provided by Sonova

- Detection of regression of a particular test case (Req-3.1: Detection of regression)
- Detection of flickering tests (Req-3.2: Detection of flickering tests)
- Detection of test environment problems (Req-3.3: Detection of test environment related failures)
- Detection of similar/related test failures (Req-3.4: Advanced detection of test failures)
- History of test results a single test case (Req-1.2.1: History of a test case)
- History of test results on a specific test environment (Req-1.2.2: History of a test environment)
- Assigning a label to a test result (Req-2.1: Labels) e.g. in order to:
  - mark the test result as irrelevant (Req-2.10: Marking some test results as irrelevant / deleting them from the database)
  - mention a related task/bug
  - mention someone who should look into a specific failure

C. Data characteristics provided by Sonova

Sample test results and metadata are to be provided by Sonova, however, the target system will have to deal with a much larger quantity of data. We can assume that we should target the following load on the system:

- Each day, the system receives 1000 test result and metadata documents
- Each test result document contains 1000 test case results
- Each metadata document contains 200 key-value pairs
- Each day there would be 100 queries issued by users
- Each user query should take $<$ 2-5 seconds to be processed

D. Source Code Snippets

Viewer Repository

Listing D.1: compose.yaml

```yaml
services:
  proxy:
    image: nginx:stable-alpine
    extra_hosts:
      - "host.docker.internal:host-gateway"
    ports:
      - 9080:80
      - 443:443
    volumes:
      - ./frontend/dist/angular-frontend:/usr/share/nginx/html
      - ./nginx/:/etc/nginx/
```

Listing D.2: compose.dev.yaml, used for development only

```yaml
services:
  proxy:
    image: caddy:2-alpine
    extra_hosts:
      - "host.docker.internal:host-gateway"
    ports:
      - 9080:80
    volumes:
      - $PWD/caddy/Caddyfile:/etc/caddy/Caddyfile
```

Listing D.3: Nginx Reverse Proxy configuration

```
events {
}

http {
  upstream kibana {
    server host.docker.internal:5601;
    keepalive 15;
  }
  upstream elastic {
    server host.docker.internal:9200;
    keepalive 15;
  }
  upstream frontend-dev {
    server host.docker.internal:4200;
    keepalive 15;
  }
  default_type      application/octet-stream;
  include       mime.types;
  server {
    listen      443 ssl;
    server_name   testresults.kuendig.dev;
    ssl_certificate   testresults.kuendig.dev.crt;
    ssl_certificate_key testresults.kuendig.dev.key;
    ssl_protocols   TLSv1.2 TLSv1.3;
    ssl_ciphers   HIGH:!aNULL:!MD5;
    server_name   testresults.kuendig.dev;

    location /api/ {
      proxy_pass https://elastic;
      proxy_set_header X-Real-IP $remote_addr;
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
      proxy_set_header X-NginX-Proxy true;
      proxy_set_header Connection "Keep-Alive";
      proxy_set_header Proxy-Connection "Keep-Alive";
      proxy_redirect off;
      proxy_buffering off;
      rewrite ^/api/(.*)$ /$1 break;
    }

    location /kibana/ {
      proxy_pass https://kibana;
      proxy_set_header X-Real-IP $remote_addr;
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
      proxy_set_header X-NginX-Proxy true;
      proxy_set_header Connection "Keep-Alive";
      proxy_set_header Proxy-Connection "Keep-Alive";
      proxy_redirect off;
      proxy_buffering off;
    }

    location / {
      root /usr/share/nginx/html;
      index index.html index.htm;
      try_files $uri $uri/ /index.html; # redirect all unknown pages to index.html,
  see https://angular.io/guide/deployment#fallback-configuration-examples
    }
  }
}
```

Listing D.4: Caddy Reverse Proxy configuration

```
:80 {
  encode zstd gzip

  route /kibana* {
    reverse_proxy https://host.docker.internal:5601 {
      transport http {
        tls_insecure_skip_verify
      }
    }
  }

  route /api* {
    uri strip_prefix /api // only for local development
    reverse_proxy https://host.docker.internal:9200 {
      transport http {
        tls_insecure_skip_verify
      }
    }
  }

  route {
    reverse_proxy host.docker.internal:4200
  }
}
```

Elasticsearch Repository

Listing D.5: compose.yaml

```
services:
  setup:
    image: docker.elastic.co/elasticsearch/elasticsearch:${STACK_VERSION}
    volumes:
      - certs:/usr/share/elasticsearch/config/certs
    user: "0"
    command: >
      bash -c '
        if [ x${ELASTIC_PASSWORD} == x ]; then
          echo "Set the ELASTIC_PASSWORD environment variable in the .env file";
          exit 1;
        elif [ x${KIBANA_PASSWORD} == x ]; then
          echo "Set the KIBANA_PASSWORD environment variable in the .env file";
          exit 1;
        fi;
        if [ ! -f config/certs/ca.zip ]; then
          echo "Creating CA";
          bin/elasticsearch-certutil ca --silent --pem -out config/certs/ca.zip;
          unzip config/certs/ca.zip -d config/certs;
        fi;
        if [ ! -f config/certs/certs.zip ]; then
          echo "Creating certs";
          echo -ne \
          "instances:\n"\
          "  - name: es01\n"\
          "    dns:\n"\
          "      - es01\n"\
          "      - localhost\n"\
          "    ip:\n"\
          "      - 127.0.0.1\n"\
          "  - name: es02\n"\
          "    dns:\n"\
          "      - es02\n"\
          "      - localhost\n"\
          "    ip:\n"\
          "      - 127.0.0.1\n"\
          "  - name: es03\n"\
          "    dns:\n"\
          "      - es03\n"\
          "      - localhost\n"\
          "    ip:\n"\
          "      - 127.0.0.1\n"\
          > config/certs/instances.yml;
          bin/elasticsearch-certutil cert --silent --pem -out config/certs/certs.zip
  --in config/certs/instances.yml --ca-cert config/certs/ca/ca.crt --ca-key
  config/certs/ca/ca.key;
          unzip config/certs/certs.zip -d config/certs;
        fi;
        echo "Setting file permissions"
        chown -R root:root config/certs;
        find . -type d -exec chmod 750 \{\} \;;
        find . -type f -exec chmod 640 \{\} \;;
        echo "Waiting for Elasticsearch availability";
        until curl -s --cacert config/certs/ca/ca.crt https://es01:9200 | grep -q
  "missing authentication credentials"; do sleep 30; done;
        echo "Setting kibana_system password";
        until curl -s -X POST --cacert config/certs/ca/ca.crt -u
  "elastic:${ELASTIC_PASSWORD}" -H "Content-Type: application/json"
  https://es01:9200/_security/user/kibana_system/_password -d
  "{\"password\":\"${KIBANA_PASSWORD}\"}" | grep -q "^{}"; do sleep 10; done;
        echo "All done!";
      '
    healthcheck:
      test: ["CMD-SHELL", "[ -f config/certs/es01/es01.crt ]"]
      interval: 1s
      timeout: 5s
      retries: 120

  es01:
    depends_on:
      setup:
        condition: service_healthy
    image: docker.elastic.co/elasticsearch/elasticsearch:${STACK_VERSION}
    volumes:
      - certs:/usr/share/elasticsearch/config/certs
      - esdata01:/usr/share/elasticsearch/data
    ports:
      - ${ES_PORT}:9200
    environment:
      - node.name=es01
      - cluster.name=${CLUSTER_NAME}
      - cluster.initial_master_nodes=es01,es02,es03
      - discovery.seed_hosts=es02,es03
      - ELASTIC_PASSWORD=${ELASTIC_PASSWORD}
      - bootstrap.memory_lock=true
      - xpack.security.enabled=true
      - xpack.security.http.ssl.enabled=true
      - xpack.security.http.ssl.key=certs/es01/es01.key
      - xpack.security.http.ssl.certificate=certs/es01/es01.crt
      - xpack.security.http.ssl.certificate_authorities=certs/ca/ca.crt
      - xpack.security.transport.ssl.enabled=true
      - xpack.security.transport.ssl.key=certs/es01/es01.key
      - xpack.security.transport.ssl.certificate=certs/es01/es01.crt
      - xpack.security.transport.ssl.certificate_authorities=certs/ca/ca.crt
      - xpack.security.transport.ssl.verification_mode=certificate
      - xpack.license.self_generated.type=${LICENSE}
    mem_limit: ${MEM_LIMIT}
    ulimits:
      memlock:
        soft: -1
        hard: -1
    healthcheck:
      test:
        [
          "CMD-SHELL",
          "curl -s --cacert config/certs/ca/ca.crt https://localhost:9200 | grep -q
'missing authentication credentials'",
        ]
      interval: 10s
      timeout: 10s
      retries: 120

  es02:
    depends_on:
      - es01
    image: docker.elastic.co/elasticsearch/elasticsearch:${STACK_VERSION}
    volumes:
      - certs:/usr/share/elasticsearch/config/certs
      - esdata02:/usr/share/elasticsearch/data
    environment:
      - node.name=es02
      - cluster.name=${CLUSTER_NAME}
      - cluster.initial_master_nodes=es01,es02,es03
      - discovery.seed_hosts=es01,es03
      - bootstrap.memory_lock=true
      - xpack.security.enabled=true
      - xpack.security.http.ssl.enabled=true
      - xpack.security.http.ssl.key=certs/es02/es02.key
      - xpack.security.http.ssl.certificate=certs/es02/es02.crt
      - xpack.security.http.ssl.certificate_authorities=certs/ca/ca.crt
      - xpack.security.transport.ssl.enabled=true
      - xpack.security.transport.ssl.key=certs/es02/es02.key
      - xpack.security.transport.ssl.certificate=certs/es02/es02.crt
      - xpack.security.transport.ssl.certificate_authorities=certs/ca/ca.crt
      - xpack.security.transport.ssl.verification_mode=certificate
      - xpack.license.self_generated.type=${LICENSE}
    mem_limit: ${MEM_LIMIT}
    ulimits:
      memlock:
        soft: -1
        hard: -1
    healthcheck:
      test:
        [
          "CMD-SHELL",
          "curl -s --cacert config/certs/ca/ca.crt https://localhost:9200 | grep -q
'missing authentication credentials'",
        ]
      interval: 10s
      timeout: 10s
      retries: 120

  es03:
    depends_on:
      - es02
    image: docker.elastic.co/elasticsearch/elasticsearch:${STACK_VERSION}
    volumes:
      - certs:/usr/share/elasticsearch/config/certs
      - esdata03:/usr/share/elasticsearch/data
    environment:
      - node.name=es03
      - cluster.name=${CLUSTER_NAME}
      - cluster.initial_master_nodes=es01,es02,es03
      - discovery.seed_hosts=es01,es02
      - bootstrap.memory_lock=true
      - xpack.security.enabled=true
      - xpack.security.http.ssl.enabled=true
      - xpack.security.http.ssl.key=certs/es03/es03.key
      - xpack.security.http.ssl.certificate=certs/es03/es03.crt
      - xpack.security.http.ssl.certificate_authorities=certs/ca/ca.crt
      - xpack.security.transport.ssl.enabled=true
      - xpack.security.transport.ssl.key=certs/es03/es03.key
      - xpack.security.transport.ssl.certificate=certs/es03/es03.crt
      - xpack.security.transport.ssl.certificate_authorities=certs/ca/ca.crt
      - xpack.security.transport.ssl.verification_mode=certificate
      - xpack.license.self_generated.type=${LICENSE}
    mem_limit: ${MEM_LIMIT}
    ulimits:
      memlock:
        soft: -1
        hard: -1
    healthcheck:
      test:
        [
          "CMD-SHELL",
          "curl -s --cacert config/certs/ca/ca.crt https://localhost:9200 | grep -q
  'missing authentication credentials'",
        ]
      interval: 10s
      timeout: 10s
      retries: 120

  kibana:
    depends_on:
      es01:
        condition: service_healthy
      es02:
        condition: service_healthy
      es03:
        condition: service_healthy
    image: docker.elastic.co/kibana/kibana:${STACK_VERSION}
    volumes:
      - certs:/usr/share/kibana/config/certs
      - kibanadata:/usr/share/kibana/data
      - type: bind
        source: ./kibana/kibana.yml
        target: /usr/share/kibana/config/kibana.yml
      - type: bind
        source: ./cert/kibana.crt
        target: /usr/share/kibana/config/certs/kibana.crt
      - type: bind
        source: ./cert/kibana.key
        target: /usr/share/kibana/config/certs/kibana.key
    ports:
      - ${KIBANA_PORT}:5601
    environment:
      - SERVERNAME=testresults.kuendig.dev
      - ELASTICSEARCH_HOSTS=https://es01:9200
      - ELASTICSEARCH_USERNAME=kibana_system
      - ELASTICSEARCH_PASSWORD=${KIBANA_PASSWORD}
      - ELASTICSEARCH_SSL_CERTIFICATEAUTHORITIES=config/certs/ca/ca.crt
      - xpack.security.encryptionKey=${KIBANA_ENCRYPTION_KEY}
    mem_limit: ${MEM_LIMIT}
    healthcheck:
      test:
        [
          "CMD-SHELL",
          "curl -s -I http://localhost:5601 | grep -q 'HTTP/1.1 302 Found'",
        ]
      interval: 10s
      timeout: 10s
      retries: 120

  init:
    depends_on:
      es01:
        condition: service_healthy
    build: ./init
    volumes:
      - ./init/definitions:/definitions
      - type: bind
        source: ./init/init.sh
        target: /usr/local/bin/init.sh
    environment:
      - ES_HOST=es01:9200
      - ES_USER=elastic
      - ES_PASS=${ELASTIC_PASSWORD}
    entrypoint: ["sh", "/usr/local/bin/init.sh"]
    restart: no

volumes:
  certs:
    driver: local
  esdata01:
    driver: local
  esdata02:
    driver: local
  esdata03:
    driver: local
  kibanadata:
    driver: local
```

Listing D.6: init.sh

```
#!/bin/sh

if ! [[ $ES_USER ]]; then
    echo "ES_USER not set"
    exit 1;
```

```
fi

if ! [[ $ES_PASS ]]; then
    echo "ES_PASS not set"
    exit 1;
fi

if ! [[ $ES_HOST ]]; then
    echo "ES_HOST not set"
    exit 1;
fi

setupTemplateSearch () {
    for f in $(find /definitions/search-templates/ -type f -name '*.json'); do
  name=$(basename $f | rev | cut -d '.' -f 2 | rev)
  data="{\"script\": { \"lang\": \"mustache\", \"source\": $(cat $f | tr -d '\n' |
  jq -R)}}"
  echo -e "${name}"
  curl --insecure "https://${ES_USER}:${ES_PASS}@${ES_HOST}/_scripts/${name}" -X PUT
  -d "${data}" --header "Content-Type: application/json"
  echo -e "\n"
    done
}

setupMapping () {
    for f in $(find /definitions/index/mapping/ -type f -name '*.json'); do
  name=$(basename $f | rev | cut -d '.' -f 2 | rev)
  data="{\"mappings\": $(cat $f)}"
  echo -e "${name}"
  curl --insecure "https://${ES_USER}:${ES_PASS}@${ES_HOST}/${name}" -X PUT -d
  "${data}" --header "Content-Type: application/json"
  echo -e "\n"
    done
}

setupPipeline () {
  for f in $(find /definitions/pipelines/ -type f -name '*.json'); do
    name=$(basename $f | rev | cut -d '.' -f 2 | rev)
    data="{\"processors\": $(cat $f)}"
    echo -e "${name}"
    echo -e "${data}"
    curl --insecure
  "https://${ES_USER}:${ES_PASS}@${ES_HOST}/_ingest/pipeline/${name}" -X PUT -d
  "${data}" --header "Content-Type: application/json"
    echo -e "\n"
  done
}

indexSettings () {
  for f in $(find /definitions/index/settings -type f -name '*.json'); do
    name=$(basename $f | rev | cut -d '.' -f 2 | rev)
    data="$(cat $f)"
    echo -e "${name}"
    echo -e "${data}"
    curl --insecure "https://${ES_USER}:${ES_PASS}@${ES_HOST}/testresult/_settings"
  -X PUT -d "${data}" --header "Content-Type: application/json"
    echo -e "\n"
  done
}

echo -e "Create template search"
setupTemplateSearch

echo -e "Create indicies with mapping"
#setupMapping

echo -e "Create Index Pipelines"
#setupPipeline

echo -e "Index Settings"
#indexSettings
```

Listing D.7: Dockerfile to run the init Container

```
FROM alpine:latest

RUN apk add --no-cache curl \
&& apk add --no-cache jq
```

E. Meeting Minutes

February 17, Administrative Meeting

Attending: Olivier Lischer, Luzia Kündig, Thomas Corbat

Summary

High level description of the task to be completed. Provided some guidelines on documentation content, as well as a detailed description of the task.

Planning of regular internal meetings: Weekly on Mondays, 13 pm. Expected to provide an agenda for every upcoming meeting by monday morning, containing the following items:

- What work has been done the past week, what had been planned?
- Any questions, decisions to be made with suggestions and reasoning
- Goals for the coming week

An official kickoff meeting with Sonova is planned on February 20 in Microsoft Teams.

February 20, Kickoff Meeting

Attending: Olivier Lischer, Luzia Kündig, Thomas Corbat, Dariusz Danilko, Andreas Berthoud

Summary

Detailed introduction to the current situation at Sonova, provided by Dariusz Danilko. Sample test result files and updated task description with a mockup as an example for the expected interface.

- Use database queries without joins whenever possible
- Reduce database maintenance effort with noSQL
- There might be different retention periods for test results in the future

February 27, Weekly Meeting

Week 01. Attending: Olivier Lischer, Luzia Kündig, Thomas Corbat, Dariusz Danilko

Agenda

What have we done so far:

- Draft: Project Plan
- Draft: Documentation, Structure
- Draft: Database Schema
- Tests with Elastic Setup, inserting and querying data
- Research of different database solutions

What are our next steps:

- Build first visualizations using sample data
- Prioritize given use cases
- Analyze potential limitations of NoSQL approach
- Elaborate on Elastic Setup: Security and Backup Strategy

Questions / Remarks

- Access to our Gitlab Repository for Thomas, Darek, Andreas?
- Req-1.10.1 Show GitHub details: Details attached to Test Case or Test Run?
- Can the same test be executed inside several test suites?

Discussion

Questions were answered:

- Gitlab access for all
- Github Details attached to test run
- Same test run not in different test suites but projects

Present database schema and project plan to Thomas. Database schema might need an additional entity (Test job).

March 06, Weekly Meeting

Week 02. Attending: Olivier Lischer, Luzia Kündig, Thomas Corbat, Dariusz Danilko

Agenda

What have we done so far:

- Comparison SQL vs. NoSQL
- Comparison CouchDB vs. MongoDB vs. RavenDB vs. Elasticsearch
- Analysis of XML Structure
- Create more manual sample data and insert into Elasticsearch
- Prototype available at testresults.kuendig.dev
- Script to parse XML / JSON for easy insertion into DB (partially)
- Prioritize given use cases
- Risk Assessment

What we didn't manage last week:

- Build first visualizations using sample data

What are our next steps:

- Finish script
- Decide for a database schema
- Build first visualizations using sample data

Questions / Remarks

- Confirm our understanding regarding data structure / current indexes in Elasticsearch
- Agree on prioritization of use cases

Discussion

- Use Cases
  - Add an "Overview over last / specific commit" as Use Case
  - Multiple Views are allowed for all Use Cases (even encouraged)
  - Prioritization is fine
- Logic diagram, understanding of data structure
  - Job name + build number are not guaranteed unique. Darek will get back to us on this after consulting with Andreas.
  - Unclear: will there be a more specific identifier for the object/binary under test?
- The system only has to work using the XML and JSON input

March 13, Weekly Meeting

Week 03. Attending: Olivier Lischer, Luzia Kündig, Thomas Corbat, Dariusz Danilko

Agenda

What have we done so far:

- Phrase User Stories for Use Cases with (very) high prioritization
- Decide for a database schema
- Build first visualizations using sample data (Kibana)
- Model data flow from XML/JSON to DB and view
- Update project plan
- Add bulk insertion to script
- Create random test data automatically
- Start to implement unit tests
- Research frontend framework choices

What we didn't manage last week:

- All planned tasks completed

What are our next steps:

- Continue working on frontend use cases based on decisions today
- Start to Integrate more Test Result Formats?

Questions / Remarks

- Angular vs. React.js vs. Vue.js - Why was Angular proposed? *Answer:* Because some developers at Sonova already have some experience with Angular. The choice of framework is still open, but chances are high it will be re-written in Angular after this thesis if another framework is used.
- Start with custom Frontend, or still work with Kibana? *Answer:* Kibana was the suggested frontend to start with easy database access to verify functionality and chosen schema. If the database part is stable and focus is more on the use cases, then the switch to a custom frontend is a good idea.

Discussion

- Build configuration is stored in the build identifier. There may be some changes/updates to the examples provided.
- We will use Angular as Frontend Framework. But keep analysis in documentation and justify decision.
- We will switch now from Kibana to Angular.

- Keep in mind different visualizations will depend on different timestamps. Test job start time will be relevant for focusing on test environment performance, commit time will be relevant for visualizations based on code changes.
- Keep in mind, that for some use cases there will be new or deleted tests in a given time frame.
- It may be a good idea to add support for regression or environment failure to the generator script.
- A test job always builds the project. This can cause various build errors. Does not affect our visualizations.
- Terminology clarification:
  Failure: Assertion inside a test case returned false
  Error: Unexpected exception was thrown when executing the test

March 20, Weekly Meeting

Week 04. Attending: Olivier Lischer, Luzia Kündig, Thomas Corbat, Dariusz Danilko, Andreas Berthoud

Agenda

What have we done so far:

- Angular is ready
- Connect Frontend with Elasticsearch using simple backend
- Implement Reverse proxy to prevent CORS issues
- Create basic site structure
- Create UI sketches
- Create a basic visualization (table)
- Create a more complex visualization (chart)
- Update CI/CD to test Frontend

What we didn't manage last week:

- All planned tasks completed

What are our next steps:

- Implementing Use Cases

Questions / Remarks

- Backend:
  - Query Elasticsearch directly from frontend or over a backend?
  - Backend sooner or later required, for storing application data
  - Backend Technology: Are there preferences (Python, JS/TS, ...) in case we implement something?
- Reverse Proxy:
  - We would like to implement a reverse proxy to prevent CORS issues.
  - Are there preferences (Traefik, Caddy, NGINX, ...)?

Discussion

- A backend should only be implemented if it is really necessary to implement certain use cases. Otherwise, to keep the project as simple as possible, some data parsing could be moved to the python data consumer. This should keep the frontend free of such logic.
- To implement a proxy, nginx is the tool that is known and used at Sonova.
- Input about identifying the build: the URL encodes different relevant information, which makes it difficult to use for our project. It is planned to parse the URL beforehand and add more info to the metadata JSON file.

- Feedback UI Sketches / Use Cases:
  - Test results per run: lists are usually collapsible for test suite hierarchy. Checking out some existing testing tools on how they visualize this might be a good idea.
  - Useful feature in history: only show test cases with failures at some point
- Next use cases to work on / goal for next week: Table overview of commits and test cases.

March 27, Weekly Meeting

Week 05. Attending: Olivier Lischer, Luzia Kündig, Thomas Corbat, Dariusz Danilko

Agenda

What have we done so far:

- Implement Nginx as Reverse Proxy
- Improve CI/CD pipeline
- Implement Elasticsearch Service in Angular
- Implement first version of tabular commit history

What we didn't manage last week:

- All planned tasks completed

What are our next steps:

- Add filtering to existing tabular view
- Implement Req-1.9: Show test case details
- If enough time: Implement Req-1.6: Hierarchical test order (Test Suite)

Questions / Remarks

- No questions

Discussion

- Feedback Test Case History: could be multiple products per branch and commit. Some way of displaying this, no details if all green, expandable if not all is green, also how many times was it executed?
- Maybe include some option for color blindness, not red/green
- Only display data if it is interesting - not if all was good. Add an option to display all successes as well
- Generate test data for up to 1 year to test performance
- There will be a retention policy for test results at some point in the future (based on tags)

April 03, Weekly Meeting

Week 06. Attending: Olivier Lischer, Luzia Kündig, Thomas Corbat, Dariusz Danilko

Agenda

What have we done so far:

- Add icons for Success, Error, . . .
- Add date based filtering
- Add test suite to history view
- Rewrite part of view because of faulty query
- Partially implementing Req-1.9 (extended result view)
- Research for implementing Req-1.6 (hierarchical view of test cases)
- Update documentation with all requirements, specify the ones selected for implementation

What we didn't manage last week:

- Req-1.6: Hierarchical test order

What are our next steps:

- Improve view for Req-1.9: Show test case details
- Req-1.6: Hierarchical test order

Questions / Remarks

- For Darek: Do you have an extended JSON file with the missing metadata?

Discussion

- There will be some new items in the Metadata File provided (project, build and URL)
- There is always one "main" repository that a given view should focus on. This should be implemented for the users to select for themselves as a filter.
- Projects group repositories, repositories can be used in different projects (therefore the same commit concerns different projects).
- One Test run is usually relevant for multiple commits/repositories (`VCSInfo`) for different reasons. Main module / sub module, code repo / job definition repo etc.
- Next Weeks Presentation: Focus on the problem description and our current approach on solving it. This will also be a possibility to ask Philipp and Guido what they value in the final submission of our thesis.

April 17, Weekly Meeting

Week 07. Attending: Olivier Lischer, Luzia Kündig, Thomas Corbat, Dariusz Danilko

Agenda

What have we done so far:

- “Zwischenpräsentation”
- Fix memory leaks caused by Observables
- Possibility to write custom filter queries
- Add / Update / Delete labels for a single test run

What we didn’t manage last week:

- Req-1.6: Hierarchical test order

What are our next steps:

- Implementing “Hierarchical test order”
- Implementing “Detection of regression of a particular test case”

Questions / Remarks

- What views are required / needed by the infrastructure team?
- Should we prepare the data consumer to consume data from Kafka?

April 24, Weekly Meeting

Week 08. Attending: Olivier Lischer, Luzia Kündig, Thomas Corbat, Dariusz Danilko

Agenda

What have we done so far:

- Update test environment view
- Refactor code for testcase-history view to be more reusable
- Option to select one filter criteria from a dropdown (current: branch name)
- Update consumer package to use Sonova template, refactor
- Update documentation: requirements and analysis

What we didn't manage last week:

- Req-1.6: Hierarchical test order
- Req-3.1: Detection of regression

What are our next steps:

- Continue working on Req-1.6, Req-3.1
- Test proper filtering for "two level JSON properties"

Questions / Remarks

- JUnit sample file?

Discussion

In NUnit version 3, start and end time of a job will be incorporated. Update will be made at some point, but not planned yet. Inclusion of v3 schema in our solution is not part of the scope, but could be handled in different ways.

- keep data in separate indexes in elastic
- migrate old data
- add logic to data consumer to parse old and new versions

A sanitized JUnit sample should be provided by next week.

Is there a chance to use some relevant data - either run current product at Sonova or get some more sanitized examples. We provide instructions on how to run the different components in our repository README files. Darek will try to set up and run it in the next week.

Views would mostly be structured by project or by build job.

It might be helpful to retain the order of test cases inside a suite. As of now, NUnit v2 does not provide enough metadata to do this out of the box.

May 02, Weekly Meeting

Week 09. Attending: Olivier Lischer, Luzia Kündig, Thomas Corbat, Dariusz Danilko

Agenda

What have we done so far:

- Setup Code Coverage and Report (http://test-result-viewer.pages.gitlab.ost.ch/ba2023/)
- Update many tests
- Investigate performance issue
- Solve performance issue in test environment view
- Solve performance issue in test case history
- Add loading indicator to tables
- Req-1.6: Test cases collapsible by test suite

What we didn't manage last week:

- Req-3.1: Detection of regression

What are our next steps:

- Improve Test case History view for tests running multiple times
- Generalize Test Environment view to make it easily adaptable for other data
- Req-3.1: Detection of regression
- Add old test environment overview graph to dashboard page

Questions / Remarks

- Thesis Title for BA exhibition (Test Aggregator, Test Result Viewer, ...)
- What amount of data should be displayed by default? Last 7 days? Last x commits? Should we provide a selection (buttons) per view, like "last 24h, last 7 days, all"?

Discussion

- Data to be displayed for regression: last 24h, environment info: last 7d
- Environment View: does not need the actual test names, better to display hosts as rows
- Simple green/red box for general "mixed" results should be enough for this project
- Thesis name for exhibition: Test Result Viewer

May 08, Weekly Meeting

Week 10. Attending: Olivier Lischer, Luzia Kündig, Thomas Corbat, Dariusz Danilko

Agenda

What have we done so far:

- Fix performance problem in test environment view
- Update documentation according to feedback
- Start to implement regression view
- Add buttons to display data for last 24h / 7d
- If test has multiple different results it is displayed differently

What we didn't manage last week:

- Add old test environment overview graph to dashboard page

What are our next steps:

- Clean up test result service and types
- Continue regression view
- Add old test environment overview graph to dashboard page

Questions / Remarks

- Is this definition of a flickering test correct: Same test case and commits combination, different result
- Proposing changes to the JSON file
  - `Build` →`build`
  - `VersionControlSystemInfo` →`version-control-system-info`
  - `TestEnvironment` →`test-environment`

Discussion

- Regression: there might well be commits with none or several results for certain test cases. Relevant for looking for regression should be the branch / product name.
- Display of color gradient for differing results is fine. Check regarding color blindness should be done, either with automated filter tools or by asking some people we know to be affected.
- Definition of flickering tests: same commit, same test case, different results.
- Current color choice for results is fine, should be easy to change.
- Fields will be added to VCSInfos: `commit-message`, `commit-author`
- **Test ownership** is fixed in code for every commit, way of retrieving this information is not yet clear.
- **Test assignment** (on failure) is not fixed, should be possible to add and update within our tool. Way of achieving this is unclear, it might be as a comment. It is relevant for a combination of test case / branch. Different ideas of implementation should be researched.
- Filter Query should be persistent in URL to create bookmarks for specific views.

May 15, Weekly Meeting

Week 11. Attending: Olivier Lischer, Luzia Kündig, Thomas Corbat, Dariusz Danilko

Agenda

What have we done so far:

- Add chart in “last run view” (dashboard)
- Update regression view
- Filter query stored in URL
- Write JUnit parser in consumer
- Update metadata JSON schema

What we didn’t manage last week:

- Clean up test result service (didn’t work on this because of backend may be required)

What are our next steps:

- Elasticsearch Mapping for index
- Analyze comment feature (Backend required?)

Questions / Remarks

- No questions

Discussion

- Regression view should show more information about an error cause, as expandable tiles
- Most views would benefit from showing the first parts of a stack trace in a failure or error case
- Density of the data for the test-environment view is positive
- If possible, additional aggregation for a specific property would be a nice to have feature
- Data consumer: JUnit sample file is available and will be tested
- Tests are usually run in alphabetical order, can be randomized by the developer
- Comment feature:
  - Prioritize above all other feature improvements (especially regression)
  - Updating existing values will only be done a few times per day at most
  - **Possibility using native elastic functionality should be prioritized**
  - Dedicated backend as the fallback solution if necessary (more effort needed)
  - Possibility: Adding a "manual" lookup for developers that displays assignment of issues to test runs, these can then be compared to the results in another window.

May 22, Weekly Meeting

Week 12. Attending: Olivier Lischer, Luzia Kündig, Thomas Corbat, Dariusz Danilko

Agenda

What have we done so far:

- Comment feature is implemented
- Assignment feature is implemented
- More work on documentation, analysis chapter

What we didn't manage last week:

- Filtering for comment & assignment not yet working

What are our next steps:

- Architecture improvements are needed for filtering comment & assignment
- Architecture improvements are needed overall :-)
  - Angular structure for more complex application
  - Optimize data store efficiency with mappings in elastic
- With / after that: Architecture Documentation

Questions / Remarks

- Could you provide some screenshots from your systems (e.g. Jenkins) that highlight the point we make in the documentation: "most tools only provide simple visualizations, not enough for a detailed overview"?
- We list the main problems of the current solution - are there more points we missed?
  - a long loading time
  - hard to change / extend
  - badly maintained

Discussion

- Question 1, Thomas: Will deliver some screenshots with sensitive data removed
- Question 2, Darek: I think that you covered all the pain points in those three bullet points. However, I would just like to add some more background for why these three problems occur.

  I would say that the biggest problem is the architecture of the existing system, which makes it difficult to adapt it to fulfill all the requirements while keeping the system performant at the same time. The main problem stems from the fact that the TestResult.xml files are currently stored as one document, which makes it difficult/inefficient to run arbitrary queries on the data (== long loading time + hard to change / extend)

  Another problem is that both the database and the viewer are currently developed in-house. Using third-party, state-of-the-art solutions, especially for the database, seems like a better use of developers' time (== do not reinvent the wheel). Given that such systems tend to grow in size over time. And at the same time the technology used to develop them ages, it means that there will be less and less people able to / willing to maintain and extend the functionality of such complex in-house solutions (== inefficient maintenance + hard to change / extend).

- Assignment field: Three ways of storing the possible values for filtering.
  - Use distinct selection from existing
  - Store all possible assignees in a separate index as well
  - Query from LDAP

  Sonova prefers the second option, since LDAP would possibly return a too large amount of users, distinct selection would carry the risk of duplicated values because of spelling errors.

- Comment length: This field would probably contain some reference to Jira items or any other place to provide more information. This possibility is not available in the current solution, so it could be used for many more things.

- Hand-in of the final report: Thomas requires 1 paper copy each in color and in black and white print.

June 05, Weekly Meeting

Week 15. Attending: Olivier Lischer, Luzia Kündig, Thomas Corbat, Dariusz Danilko

Agenda

What have we done so far:

- Filtering by comment and assignment
- Working on documentation
- Updated Frontend Architecture

What we didn't manage last week:

- All planned tasks completed

What are our next steps:

- Code clean up
- Documentation

Questions / Remarks

- Sonova logo on title page and A0 poster?

June 12, Weekly Meeting

Week 14. Attending: Olivier Lischer, Luzia Kündig, Thomas Corbat, Dariusz Danilko

Agenda

What have we done so far:

- Code Clean up
- Documentation

What we didn't manage last week:

- -

What are our next steps:

- Submit documentation for printing by Tuesday evening
- Submit A0 Poster for printing by Wednesday evening
- Finish with more code and frontend design cleanup

Questions / Remarks

- How long should the presentation on June 26 be?

Discussion

- Final presentation: 30 minutes, then 20 minutes for questions
- Language for the presentation: German is fine
- Feedback for most artefacts

F. Personal Reports

We provide personal reflections on topics that were important to us during the project.

Luzia Kündig

Apart from the software engineering project two semesters prior, this was my first big software development project and I have learned a lot in many different fields. We have both invested a substantial amount of time in researching the technologies we were planning to use. Still there were iterations in terms of how much of their functionaliy we could apply. Our first implementation of Angular or Elasticsearch might have been a very basic one, but after some time spent with them we soon came to learn and use the more advanced tools and concepts. This realization during the project has helped me a lot in understanding, for I could explore new concepts in terms of a problem I have encountered myself.

Working together with Olivier has been a very pleasant and positively challenging experience. We both have different levels of experience, but where he has more of it in terms of software development, I could do my part in setting up the infrastructure, extending and refining the prototypes he built, proposing some architectural decisions and keeping track of our documentation. In this way we could keep separate responsibilities and work very productively, even when spending a good amount of time away from school and working separately on the project. However, in retrospective I would try to spend more time discussing the implementation together. It might slow down the progress for the moment, but result in more reflected decisions and better understanding of the work of the other.

Our advisor Thomas and our partner Darek at Sonova have always been happy to answer any questions we might have had, even when asked for the third time about the how and why of their current situation. :-) I am grateful for their support and also their great enthusiasm about the product we have developed.

Olivier Lischer

During this project I touched many new fields, concepts and applications:

- Angular
- Elasticsearch
- Python

It was the first time that I used a web framework for frontend development. Previously, I did not write any web frontend but only backend code. Therefore, I had to learn how to work with Angular from scratch. The first version of our frontend architecture was not ideal. Nearly all application logic was placed in a single file. Although, with the current amount of features this would not be a big problem but as the application grows the more unmaintainable it gets and we decided to restructure the whole project.

I was already familiar with NoSQL as I used MongoDB during the software engineering project. At the beginning it took many hours to write a single Elasticsearch query just because I was unfamiliar with the query syntax of Elasticsearch and its possibilities. As the project progressed I got more familiar and therefore faster in writing the more complex queries.

The last technology I touched in this project was Python. My experience using Python was limited to a few lines of code. My inexperience bit me early: writing the data classes for the consumer was cumbersome and suffered from a lot of code duplication. Here it would have been helpful to ask a more experienced Python developer how you should solve this problem. Eventually I could solve it cleanly.

That Luzia was my project partner was very lucky. While Luzia has more experience in network, I had more in software development. But this was for our best as we could separate the tasks: Luzia was in charge of the Elasticsearch setup, while I was for the development. Luzia always challenged my implementations and ideas what lead to a great team work. One downside had our teamwork: we often neglected the review of the changes the other one made. We were really quick most of the time, but the other did not always know, how you solved a certain problem. In a next project, I would definitely work with merge requests to prevent this problem.

The weekly meetings with our Advisor Thomas and the Darek were always very informative and often funny. I am really thankful for Dareks patience with us, even when we needed a few attempts to understand a certain problem or situation.

G. Declarations

OST
Ostschweizer
Fachhochschule

Eigenständigkeitserklärung

Erklärung

Ich erkläre hiermit,

- dass ich die vorliegende Arbeit selbst und ohne fremde Hilfe durchgeführt habe, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit der Betreuerin / dem Betreuer schriftlich vereinbart wurde,
- dass ich sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben habe,
- dass ich keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt habe.

Ort, Datum:
Rapperwsil, 13.06.2023

Name, Unterschrift:

Luzia Kündig [illegible]

Olivier Lischer [illegible]

OST
Ostschweizer
Fachhochschule

1. Vereinbarung

1.1.1 1. Gegenstand der Vereinbarung

Mit dieser Vereinbarung werden die Rechte über die Verwendung und die Weiterentwicklung der Ergebnisse der Bachelorarbeit «Test Result Viewer» von Luzia Kündig und Olivier Lischer unter der Betreuung von Thomas Corbat geregelt.

1.1.2 2. Urheberrecht

Die Urheberrechte stehen der Studentin / dem Studenten zu.

1.1.3 3. Verwendung

Die Ergebnisse der Arbeit dürfen sowohl von der Studentin / dem Studenten, von der OST wie von Sonova AG nach Abschluss der Arbeit kostenlos und uneingeschränkt verwendet und weiterentwickelt werden.

1.1.4 Beilage/n:

- keine

Rapperswil, den 07.06.2023 [illegible] L. Kündig
Die Studentin/der Student

Rapperswil, den 06.06.2023 T. [illegible]
Der Betreuer / die Betreuerin der Bachelorarbeit

Glossary

ACID In the context of transaction processing, the acronym ACID refers to the four key properties of a transaction: atomicity, consistency, isolation, and durability.

Atomicity: All changes to data are performed as if they are a single operation. That is, all the changes are performed, or none of them are. For example, in an application that transfers funds from one account to another, the atomicity property ensures that, if a debit is made successfully from one account, the corresponding credit is made to the other account.

Consistency: Data is in a consistent state when a transaction starts and when it ends. For example, in an application that transfers funds from one account to another, the consistency property ensures that the total value of funds in both the accounts is the same at the start and end of each transaction.

Isolation: The intermediate state of a transaction is invisible to other transactions. As a result, transactions that run concurrently appear to be serialized. For example, in an application that transfers funds from one account to another, the isolation property ensures that another transaction sees the transferred funds in one account or the other, but not in both, nor in neither.

Durability: After a transaction successfully completes, changes to data persist and are not undone, even in the event of a system failure. For example, in an application that transfers funds from one account to another, the durability property ensures that the changes made to each account will not be reversed [2]. 114

BSON BSON is a computer data interchange format. The name “BSON” is based on the term JSON and stands for “Binary JSON” [Wikipedia]. 114

CAP The CAP theorem applies a similar type of logic to distributed systems—namely, that a distributed system can deliver only two of three desired characteristics: consistency, availability, and partition tolerance (the ‘C,’ ‘A’ and ‘P’ in CAP) [50]. 114

Continuous delivery Continuous delivery is a software development practice that works in conjunction with CI to automate the infrastructure provisioning and application release process. Once code has been tested and built as part of the CI process, CD takes over during the final stages to ensure it’s packaged with everything it needs to deploy to any environment at any time. CD can cover everything from provisioning the infrastructure to deploying the application to the testing or production environment. With CD, the software is built so that it can be deployed to production at any time. Then you can trigger the deployments manually or move to continuous deployment, where deployments are automated as well. [47] 114

Continuous Integration Continuous integration is the practice of integrating all your code changes into the main branch of a shared source code repository early and often, automatically testing each change when you commit or merge them, and automatically kicking off a build. With continuous integration, errors and security issues can be identified and fixed more easily, and much earlier in the development process. [47] 13, 114

Field Programmable Gate Array Field Programmable Gate Arrays (FPGAs) are integrated circuits often sold off-the-shelf. They're referred to as 'field programmable' because they provide customers the ability to reconfigure the hardware to meet specific use case requirements after the manufacturing process. This allows for feature upgrades and bug fixes to be performed in situ, which is especially useful for remote deployments. [46] 114

firmware Firmware is a type of software that is embedded directly in a piece of hardware to make the hardware work as intended. Firmware is programmed by the manufacturer and is installed on a digital device right in the factory. All computing devices have firmware. [14] 14

Kafka Apache Kafka is a distributed event store. 32, 39

Lucene Lucene Core is a Java library providing powerful indexing and search features, as well as spellchecking, hit highlighting and advanced analysis/tokenization capabilities [6]. 33

Minimum Viable Product That version of a new product which allows a team to collect the maximum amount of validated learning about customers with the least effort. This validated learning comes in the form of whether your customers will actually purchase your product. [32] 114

Prepared Statement In database management systems (DBMS), a prepared statement, parameterized statement, or parameterized query is a feature used to pre-compile SQL code, separating it from data[36]. 53

Scrum A way to get work done as a team in small pieces at a time, with continuous experimentation and feedback loops along the way to learn and improve as you go. Scrum helps people and teams deliver value incrementally in a collaborative way. As an agile framework, Scrum provides just enough structure for people and teams to integrate into how they work, while adding the right practices to optimize for their specific needs. [49] 14

Single Component Angular Module An Angular structure pattern. Every component lives in its own module[33]. 114

Topic A log of events in Kafka 32, 39

XML The Extensible Markup Language (XML) is a simple text-based format for representing structured information: documents, data, configuration, books, transactions, invoices, and much more. It was derived from an older standard format called SGML (ISO 8879), in order to be more suitable for Web use. [51] 114

Y-Statement A light template for architectural decision capturing proposed by Prof. Olaf Zimmermann [53]. 39

Acronyms

ACID Atomicity, Consistency, Isolation, Durability (See ACID) 33, 46

BSON Binary JSON (See BSON) 34

CAP Consistency, Availability, Partition tolerance (See CAP) 33

CD Continuous Delivery (See Continuous delivery) 32

CI Continuous Integration (See Continuous Integration) 13, 14, 32

CORS Cross Origin Resource Sharing 38, 45

CPU Central Processing Unit 14

DSP Digital Signal Processor 14

FPGA Field Programmable Gate Array (See Field Programmable Gate Array) 14

KQL Kibana Query Language 5

MVP Minimum Viable Product (See Minimum Viable Product) 17, 18, 63

RDBMS Relational Database Management System 32, 53

SCAM Single Component Angular Module (see Single Component Angular Module) 44, 57

SMART Specific, Measurable, Agreed upon, Realistic, Time-bound 27

TLS Transport Layer Security 28

XML Extensible Markup Language (See XML) 15

List of Figures

List of Tables

Bibliography

[1] *5.2. Performance — Apache CouchDB® 3.3 Documentation*. URL: https://docs.couchdb.org/en/stable/maintenance/performance.html (visited on 03/02/2023).

[2] *ACID properties of transactions*. IBM. URL: https://www.ibm.com/docs/en/cics-ts/5.4?topic=processing-acid-properties-transactions (visited on 05/23/2023).

[3] *Acid Transactions*. RavenDB NoSQL Database. URL: https://ravendb.net/why-ravendb/acid-transactions (visited on 06/05/2023).

[4] *Angular coding style guide*. URL: https://angular.io/guide/styleguide (visited on 03/14/2023).

[5] *Ansible deployment of Docker*. URL: https://www.digitalocean.com/community/tutorials/how-to-use-ansible-to-install-and-set-up-docker-on-ubuntu-20-04 (visited on 02/25/2023).

[6] *Apache Lucene*. Apache. URL: https://lucene.apache.org/ (visited on 05/23/2023).

[7] Benjamin Anderson and Brad Nicholson. *SQL vs. NoSQL Databases: What's the Difference? | IBM*. June 12, 2022. URL: https://www.ibm.com/cloud/blog/sql-vs-nosql (visited on 02/28/2023).

[8] *Color Vision Simulation — Firefox Source Docs Documentation*. URL: https://firefox-source-docs.mozilla.org/devtools-user/accessibility_inspector/simulation/index.html (visited on 06/08/2023).

[9] *Compodoc - The Missing Documentation Tool for Your Angular A pplication*. URL: https://compodoc.app/ (visited on 06/11/2023).

[10] *Convert Processor*. Elastic. URL: https://www.elastic.co/guide/en/elasticsearch/reference/8.7/convert-processor.html (visited on 06/13/2023).

[11] *CouchDB*. URL: https://couchdb.apache.org/ (visited on 03/02/2023).

[12] *Cross Origin Resource Sharing*. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS (visited on 05/23/2023).

[13] Abe Dearmer. *CouchDB vs. MongoDB: What You Need to Know*. URL: https://www.integrate.io/blog/couchdb-vs-mongodb/ (visited on 05/26/2023).

[14] *Definition of firmware*. URL: https://www.techopedia.com/definition/2137/firmware (visited on 05/23/2023).

[15] Yue Du. "Massive Semi-structured Data Platform Based on Elasticsearch and MongoDB". In: *Signal and Information Processing, Networking and Computers*. Ed. by Yue Wang et al. Lecture Notes in Electrical Engineering. Singapore: Springer, 2021, pp. 877–884. ISBN: 978-981-334-102-9. DOI: 10.1007/978-981-33-4102-9_105.

[16] *Elastic Connectors | Enterprise Search Documentation [8.8] | Elastic*. Learn/Docs/Enterprise Search/Guide/8.8. URL: https://www.elastic.co/guide/en/enterprise-search/current/connectors.html (visited on 05/26/2023).

[17] *Elastic Docs*. URL: https://www.elastic.co/guide/index.html (visited on 02/03/2023).

[18] *Elasticsearch Node.Js Client*. elastic, June 9, 2023. URL: https://github.com/elastic/elasticsearch-js (visited on 06/09/2023).

[19] *Encrypt HTTP client communications for Kibana.* URL: https://www.elastic.co/guide/en/elasticsearch/reference/8.6/security-basic-setup-https.html#encrypt-kibana-http (visited on 03/25/2023).

[20] *Explicit Mapping | Elasticsearch Guide [8.8] | Elastic.* URL: https://www.elastic.co/guide/en/elasticsearch/reference/current/explicit-mapping.html (visited on 06/10/2023).

[21] Estefanía García Gallardo. *RxJS Best Practices.* Medium. May 8, 2021. URL: https://betterprogramming.pub/rxjs-best-practices-7f559d811514 (visited on 06/11/2023).

[22] *Grafana.* URL: https://grafana.com/ (visited on 05/19/2023).

[23] Sheffi Gupta and Rinkle Rani. "A Comparative Study of Elasticsearch and CouchDB Document Oriented Databases". In: *2016 International Conference on Inventive Computation Technologies (ICICT).* Coimbatore, India: IEEE, Aug. 2016, pp. 1–4. ISBN: 978-1-5090-1285-5. DOI: 10.1109/INVENTIVE.2016.7823252. (Visited on 05/26/2023).

[24] *Indexes: Analyzers.* RavenDB NoSQL Database. URL: https://ravendb.net/docs/article-page/5.4/Csharp/indexes/using-analyzers (visited on 06/05/2023).

[25] *Indexes: Map-Reduce Indexes.* RavenDB NoSQL Database. URL: https://ravendb.net/docs/article-page/5.4/csharp/indexes/map-reduce-indexes#indexes-map-reduce-indexes (visited on 06/06/2023).

[26] *Insert Information in Database Using Grafana.* Grafana Labs Community Forums. Jan. 6, 2022. URL: https://community.grafana.com/t/insert-information-in-database-using-grafana/58611 (visited on 03/09/2023).

[27] Jimmix. *Answer to "Update\Insert Data from Grafana to Mysql".* Stack Overflow. May 30, 2020. URL: https://stackoverflow.com/a/62101610 (visited on 03/09/2023).

[28] *JUnit-Schema.* Windy Road Technology Pty. Limited, Mar. 1, 2023. URL: https://github.com/windyroad/JUnit-Schema/blob/cfa434d4b8e102a8f55b8727b552a0063ee9044e/JUnit.xsd (visited on 04/24/2023).

[29] *Kibana Docs.* URL: https://www.elastic.co/guide/en/kibana/current/introduction.html (visited on 05/19/2023).

[30] Lauren Schaefer. *NoSQL Vs SQL Databases.* MongoDB. URL: https://www.mongodb.com/nosql-explained/nosql-vs-sql (visited on 02/28/2023).

[31] *Mapping | Elasticsearch Guide [8.8] | Elastic.* URL: https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping.html (visited on 06/10/2023).

[32] *Minimum Viable Product, Glossary.* URL: https://www.agilealliance.org/glossary/mvp/ (visited on 05/23/2023).

[33] *Module SCAM Pattern.* URL: https://angular-training-guide.rangle.io/modules/module-scam-pattern (visited on 06/08/2023).

[34] *MongoDB.* URL: https://www.mongodb.com/ (visited on 03/02/2023).

[35] Judy Nduati. *Elasticsearch vs MongoDB - A Detailed Comparison of Document-Oriented Databases | SigNoz.* https://signoz.io/blog/elasticsearch-vs-mongodb/. Jan. 2023. (Visited on 05/26/2023).

[36] *Prepared Statement.* In: *Wikipedia.* Aug. 11, 2022. URL: https://en.wikipedia.org/w/index.php?title=Prepared_statement&oldid=11%2003939861 (visited on 06/10/2023).

[37] *Query and filter context.* Elastic. URL: https://www.elastic.co/guide/en/elasticsearch/reference/8.7/query-filter-context.html.

[38] *Run API Requests | Kibana Guide [8.6] | Elastic.* URL: https://www.elastic.co/guide/en/kibana/current/console-kibana.html (visited on 03/09/2023).

[39] *Search Templates | Elasticsearch Guide [8.8] | Elastic*. URL: https://www.elastic.co/guide/en/elasticsearch/reference/current/search-template.html (visited on 06/10/2023).

[40] *Setup ElasticSearch with example Docker Compose File*. URL: https://www.elastic.co/guide/en/elasticsearch/reference/current/docker.html#docker-compose-file (visited on 03/25/2023).

[41] The NUnit Project. *Test Result XML Format | NUnit Docs*. URL: https://docs.nunit.org/articles/nunit/technical-notes/usage/Test-Result-XML-Format.html (visited on 04/23/2023).

[42] The NUnit Project. *Testresult Schema 2.5*. URL: https://nunit.org/files/testresult_schema_25.txt (visited on 04/23/2023).

[43] *The State of JS 2022: Front-end Frameworks*. URL: https://2022.stateofjs.com/en-US/libraries/front-end-frameworks/ (visited on 03/09/2023).

[44] *This Is Sonova | Sonova International*. URL: https://www.sonova.com/en/sonova-0 (visited on 05/07/2023).

[45] *Virtual memory*. Elastic. URL: https://www.elastic.co/guide/en/elasticsearch/reference/current/vm-max-map-count.html (visited on 03/12/2023).

[46] *What Is an FPGA?* URL: https://www.arm.com/glossary/fpga (visited on 05/18/2023).

[47] *What is CI/CD? - RedHat*. URL: https://about.gitlab.com/topics/ci-cd/ (visited on 05/23/2023).

[48] *What Is Elasticsearch?* Elastic. URL: https://www.elastic.co/what-is/elasticsearch (visited on 03/02/2023).

[49] *What is Scrum?* URL: https://www.scrum.org/learning-series/what-is-scrum (visited on 05/22/2023).

[50] *What Is the CAP Theorem? | IBM*. https://www.ibm.com/topics/cap-theorem. URL: https://www.ibm.com/topics/cap-theorem (visited on 05/26/2023).

[51] *XML Essentials*. URL: https://www.w3.org/standards/xml/core (visited on 05/23/2023).

[52] Mani Yangkatisal. *The Battle of the NoSQL Databases - Comparing MongoDB and CouchDB*. July 2020. (Visited on 05/26/2023).

[53] Olaf Zimmermann. *Architectural Decisions — The Making Of*. URL: https://ozimmer.ch/practices/2020/04/27/ArchitectureDecisionMaking.html#y-statements-and-other-templates.

[54] Olaf Zimmermann. *SMART criteria*. URL: https://socadk.github.io/design-practice-repository/activities/DPR-SMART-NFR-Elicitation (visited on 05/27/2023).