



# Test Monitoring Tool Realisation

Bachelor Applied Computer Science  
Specialization Digital Innovation

Syan Delbart

Academic Year 2023-2024

Campus Geel, Kleinhoefstraat 4, BE-2440 Geel

# TABLE OF CONTENTS

<b><i>Table of contents</i></b> .....	<b>3</b>
<b>1 Introduction</b> .....	<b>5</b>
<b>1.1 The company</b> .....	<b>5</b>
<b>1.2 Issue</b> .....	<b>6</b>
<b>1.3 Mission</b> .....	<b>6</b>
<b>1.4 Planning</b> .....	<b>6</b>
<b>2 Used technologies</b> .....	<b>7</b>
<b>2.1 NodeJS</b> .....	<b>7</b>
<b>2.2 NextJS</b> .....	<b>7</b>
<b>2.3 NextAuth (Auth.JS)</b> .....	<b>7</b>
<b>2.4 Ollama</b> .....	<b>7</b>
<b>2.5 TensorFlow</b> .....	<b>8</b>
<b>2.6 Redis</b> .....	<b>8</b>
<b>2.7 BullMQ</b> .....	<b>8</b>
<b>2.8 PostgreSQL</b> .....	<b>8</b>
<b>2.9 Puppeteer</b> .....	<b>8</b>
<b>2.10 Prisma</b> .....	<b>9</b>
<b>2.11 Docker</b> .....	<b>9</b>
<b>2.12 SonarCloud</b> .....	<b>9</b>
<b>3 Features</b> .....	<b>10</b>
<b>3.1 Manage Sources</b> .....	<b>10</b>
<b>3.2 Manage API Keys</b> .....	<b>11</b>
<b>3.3 Manage Prompts</b> .....	<b>13</b>
<b>3.4 Manage Issues</b> .....	<b>14</b>
3.4.1 Integration with external tooling .....	14
<b>3.5 Manage Logs</b> .....	<b>16</b>
3.5.1 Reactivity .....	17
3.5.2 Context AI Models .....	17
3.5.3 Filtering.....	19
3.5.4 Column filtering.....	19
3.5.5 Bulk action buttons .....	20
3.5.6 Preprocessing .....	21
3.5.7 Run through AI (Classification) .....	21
3.5.8 Visual AI model .....	23
<b>4 Application structure</b> .....	<b>24</b>
<b>4.1 Retrieving the logs</b> .....	<b>24</b>
4.1.1 Running the context AI.....	24

<b>4.2 Queue system .....</b>	<b>25</b>
4.2.1.1 Retrieving the logs .....	25
4.2.1.2 Running the context AI.....	25
<b>4.3 Deployment.....</b>	<b>26</b>
4.3.1 Quality insurance.....	26
4.3.1.1 Building the application .....	26
4.3.1.2 Analyzing the code.....	26
4.3.2 Deploying the application .....	27
<b>5 Analyzing Phase.....</b>	<b>28</b>
<b>5.1 Questions logs.....</b>	<b>28</b>
<b>5.2 Questions AI.....</b>	<b>29</b>
<b>6 Conclusion .....</b>	<b>30</b>
<b>7 Glossary.....</b>	<b>31</b>

# 1 INTRODUCTION

## 1.1 The company

Resillion plays an essential role in supporting companies in developing, validating, and optimizing digital products and services, especially in the context of the Internet of Things (IoT). The company offers comprehensive testing and validation services to ensure the reliability, security, and efficiency of digital systems, devices, and content. Their expert team works closely with clients to provide customized solutions that meet the highest industry standards.

Resillion designs and conducts tests to evaluate the performance and security of digital products and offers quality assurance services to identify and resolve potential issues during development and implementation. This helps companies comply with national and international regulations and standards and remain competitive in a changing market.

Additionally, Resillion offers training and consultancy services to improve companies' internal processes and skills, leading to higher product quality and better competitiveness. With a specialized division, Resillion Cyber Security, the company also helps protect digital systems and content against cyberattacks.

In summary, Resillion supports companies throughout the entire software development lifecycle with high-quality testing and quality assurance services, resulting in cost savings, faster time-to-market, and success in the information technology sector and beyond.

During my internship, I worked with colleagues at SQS. Within this department, QA testing is performed, including both manual tests and automation testing. Various types of applications are tested here, including web applications, mobile applications, and more.

## 1.2 Issue

The manual analysis of test logs is a time-consuming process that involves various employees who need to individually review these logs and link them to specific issues. Given that many of these problems are recurring, automating this process offers significant added value. Automation can not only reduce a considerable amount of repetitive manual work but also enable the team to use their available time and resources more efficiently.

## 1.3 Mission

During my internship at Resillion, I was responsible for the "Test Monitoring Tool" project. The primary goal of this project was to develop an advanced tool capable of performing in-depth analyses on complex test logs. A crucial aspect of this project was the tool's ability to make predictions about the most likely errors based on identified patterns and trends. Additionally, it was essential that these predictions be presented to the end-user in a visually compelling manner to enable more efficient processing of the test logs.

Although the tool is officially named a "Test Monitoring Tool," it is designed to work with any type of log file. This means it can handle various log files, not just those related to tests. Therefore, the term "log monitoring" is used more frequently throughout this report, as it more accurately reflects the tool's broader functionality.

## 1.4 Planning

The initial overview plan was the baseline planning utilized at the beginning of the project. However, over time, deviations from this plan occurred. I subsequently adopted Kanban methodologies, which did not fully align with the initial plan due to its rigid structure based on a waterfall model. For documentation purposes, the original planning is included, but for the eventual realization of this project, mainly a Kanban board was used. There was a total of approximately 100 created tickets.

ID	Name	Feb, 2024		Mar, 2024			Apr, 2024				May, 2024				Ju...		
		19 Feb	25 Feb	03 Mar	10 Mar	17 Mar	24 Mar	31 Mar	07 Apr	14 Apr	21 Apr	28 Apr	05 May	12 May	19 May	26 May	02...
1	Onderzoeksfase		█	█	█												
2	AI model					█	█	█	█								
4	Backend + AI Integration									█	█	█					
3	Frontend											█	█	█			
5	Testing & Integration													█	█		
6	Documentation & Handover															█	█

Figure 1 - A representation of the initial planning

Bi-weekly (internal) demos provided an excellent opportunity to gather substantial feedback and observe the progression of the project. The use of Kanban allowed for more flexibility, which was very beneficial as not every task was already defined. It made more room for overcoming certain obstacles and improved the amount of work that could be delivered at each demo.

## **2 USED TECHNOLOGIES**

### **2.1 NodeJS**

Node.js is an open-source, cross-platform JavaScript runtime environment that executes JavaScript code outside of a web browser. It is designed to build scalable network applications and is particularly well-suited for I/O-heavy operations, such as web servers and real-time applications. Node.js uses an event-driven, non-blocking I/O model, which makes it efficient and lightweight. It also has a vast ecosystem of libraries and modules available via the npm (Node Package Manager).

### **2.2 NextJS**

Next.js is a React framework that enables functionality such as server-side rendering and generating static websites for React-based web applications. It simplifies the process of building complex web applications by providing a robust framework with features like automatic code splitting, easy static export, and a rich plugin ecosystem. Next.js is particularly popular for building SEO-friendly websites and applications due to its ability to render pages on the server side.

### **2.3 NextAuth (Auth.JS)**

NextAuth is an authentication provider specifically designed for NextJS. This library supports basic credential authentication (email, password) as well as third-party providers. Since during the internship itself there will not be any access to the required tokens to make third-party providers work, it is convenient to be able to use credentials as login method. The added benefit is that when the tool gets deployed to production, alternative third-party providers can easily be added without having to modify the existing data structure or views.

### **2.4 Ollama**

Ollama is an open-source application designed to enable the running, creation, and sharing of large language models (LLMs) locally on personal hardware such as MacOS and Linux systems. Initially supporting models like Llama2, its library has expanded to include others like Mistral and Phi-2. Ollama provides a straightforward command-line interface (CLI) for users to interact with these models, facilitating easy setup and use without extensive technical knowledge.

Ollama allows users to run LLMs locally, ensuring greater control over data privacy and potentially reducing costs compared to cloud services. It supports its own API and an OpenAI-compatible API, allowing developers to seamlessly integrate LLMs into their applications using various programming languages, including Python and JavaScript. Additionally, Ollama can run within Docker containers, facilitating integration into larger deployment pipelines, and providing an isolated environment for model execution.

By providing a flexible and powerful platform for managing large language models locally, Ollama democratizes access to advanced AI capabilities, making it easier for individuals and small organizations to leverage state-of-the-art AI technology without extensive infrastructure or cloud dependencies.

## 2.5 TensorFlow

TensorFlow is an open-source machine learning library developed by Google. It is widely used for building and training machine learning models, including deep neural networks. TensorFlow provides a flexible ecosystem of tools, libraries, and community resources that enable researchers and developers to push the state-of-the-art in machine learning and easily build and deploy machine learning-powered applications. It supports various platforms and languages, with Python being the most used.

## 2.6 Redis

Redis (Remote Dictionary Server) is an open-source, in-memory data structure store used as a database, cache, and message broker. It supports various data structures such as strings, hashes, lists, sets, sorted sets, and more. Redis is known for its high performance, flexibility, and extensive feature set, making it suitable for use cases requiring low-latency data access, such as caching, real-time analytics, session management, and message queuing.

## 2.7 BullMQ

BullMQ is a Node.js library for creating robust and scalable job queues using Redis. It is an evolution of the popular Bull library, designed to handle background jobs, process them asynchronously, and manage job queues with advanced features like job prioritization, concurrency control, retries, and job scheduling. BullMQ is particularly useful for applications that require background processing, task scheduling, and handling of high-throughput data workflows.

## 2.8 PostgreSQL

BullMQ is a Node.js library for creating robust and scalable job queues using Redis. It is an evolution of the popular Bull library, designed to handle background jobs, process them asynchronously, and manage job queues with advanced features like job prioritization, concurrency control, retries, and job scheduling. BullMQ is particularly useful for applications that require background processing, task scheduling, and handling of high-throughput data workflows.

## 2.9 Puppeteer

Puppeteer is a Node.js library that provides a high-level API to control headless Chrome or Chromium browsers. It is used for web scraping, automated testing, and web performance analysis. Puppeteer allows developers to programmatically interact with web pages, simulate user interactions, and capture screenshots or PDFs of web pages. It is particularly useful for automating browser tasks and running tests in a real browser environment without the need for a graphical user interface.

## **2.10 Prisma**

Prisma is an open-source ORM (Object-Relational Mapping) tool that simplifies database management by allowing developers to define and interact with their database schema in a type-safe manner. It supports various databases and enables efficient data querying, migrations, and seamless integration with modern development frameworks.

## **2.11 Docker**

Docker is an open-source platform that automates the deployment, scaling, and management of applications using containerization. Containers are lightweight, portable units that package an application along with its dependencies, libraries, and configuration files. This ensures that the application runs consistently across different environments. Docker simplifies the development lifecycle by allowing developers to build, share, and run applications in isolated environments, reducing conflicts and enhancing scalability. Docker also integrates with various orchestration tools like Kubernetes, making it easier to manage containers in large-scale, distributed systems.

## **2.12 SonarCloud**

SonarCloud is a cloud-based code quality and security service that performs static code analysis to detect bugs, code smells, and security vulnerabilities in your projects. It supports multiple programming languages and integrates seamlessly with popular CI/CD pipelines. SonarCloud helps ensure that code is clean, maintainable, and secure by providing detailed reports and actionable insights. It also includes features such as code coverage analysis, duplications detection, and technical debt measurement, making it an essential tool for continuous code quality and improvement. SonarCloud's dashboards and visualizations help teams track code quality over time and prioritize areas that need attention.



## 3 FEATURES

The application is designed to be highly manageable. We aim to minimize direct intervention with the application itself, so the system is built to allow adjustments to be made from within the application itself.

### 3.1 Manage Sources

This feature allows users to manually add and organize various data sources. Although the input feeding method cannot be customized, if data is fetched from a specific source, the source program can be automatically created. Currently, this is mainly used as a label and a way to separate and sort the log files.

The screenshot shows the 'Sources' management page in a 'Test monitoring tool'. The page has a navigation bar with 'Logs', 'Sources', 'Prompts', and 'Issues'. There are buttons for 'My API Keys', 'My Profile', and 'Sign out'. A 'Create program' button is located in the top right corner of the table area.

<input type="checkbox"/>	ID ↑	Name ↑	Color	Created At	Actions
<input type="checkbox"/>	170	[blurred]	#808080 ●	2/5/2024, 10:58:26	<a href="#">✎</a> <a href="#">🗑️</a>
<input type="checkbox"/>	207	[blurred]	#808080 ●	2/5/2024, 10:58:26	<a href="#">✎</a> <a href="#">🗑️</a>
<input type="checkbox"/>	53051	[blurred]	#808080 ●	7/5/2024, 11:43:05	<a href="#">✎</a> <a href="#">🗑️</a>
<input type="checkbox"/>	346	[blurred]	#808080 ●	2/5/2024, 10:58:26	<a href="#">✎</a> <a href="#">🗑️</a>
<input type="checkbox"/>	353	[blurred]	#808080 ●	2/5/2024, 10:58:26	<a href="#">✎</a> <a href="#">🗑️</a>
<input type="checkbox"/>	1	[blurred]	#808080 ●	2/5/2024, 10:58:26	<a href="#">✎</a> <a href="#">🗑️</a>
<input type="checkbox"/>	352	[blurred]	#808080 ●	2/5/2024, 10:58:26	<a href="#">✎</a> <a href="#">🗑️</a>

0 of 27 row(s) selected. Rows per page: 10 Page 3 of 3

Data is stored locally. Models run locally, and no data is being sent to any third-party service.  
© 2024 - Syan Delbart

Figure 2 - A representation of the existing sources

The source names are blurred out in this instance. A custom color can be provided to the sources for easier identification.

## 3.2 Manage API Keys

Communication with the application from the outside is done using API keys. In practice, these are JSON Web Tokens that contain the relevant information about the API key, such as the creator, the source for which the API key was created (optional) and an expiry date.

**My API Keys**

Name	Expires At	Token
<input type="checkbox"/> some_api_key_name	10/7/2024, 11:57:03	

0 of 1 row(s) selected. Rows per page: 10 Page 1 of 1

**Create API key**  
Create a new API key to authenticate your external services.

**Name**

**Expires at**

**Source**

**Create**

**Instructions**  
Follow these instructions to use your API key.

To use your API key, include it in the Authorization header of your requests.  
Authorization : Bearer <your-api-key>

**Never expiring API keys**  
If you want your API key to never expire, select 'Never' in the 'Expires at' field.

Data is stored locally. Models run locally, and no data is being sent to any third-party service.  
© 2024 - Syan Delbart

Figure 3 - A representation of the user's API keys.

The API keys can be used for client interaction with the application using the supported API endpoints. Specific endpoints have been created for external connections.

Route	Explanation
/api/external/heartbeat	Used to test if the API key is (still) valid.
/api/external/logs	Used for managing logs externally. Mainly used to fetch the AI context and classification.
/api/external/logs/linkIssue	If an issue is provided, the log classification will be corrected to this provided value.
/api/external/logs/linkIssue/correctPrediction	Sends a signal that the predicted value is correct, which sets the corrected value to the prediction value.

Another use case for these API keys is to add logs to the application from a certain data source. This integrates nicely with technologies that output logs which can call certain API endpoints. A perfect example is a pipeline, which can call the API endpoint and provide it with the relevant information to create a new log.

### 3.3 Manage Prompts

Since different programs might have different interpretations of log messages, there is a necessity for being able to customize the prompts. Existing prompts can be modified to improve the context output of the used AI models. Currently, in the application itself, prompts are applied per log, and not per source. This was a conscious decision to be able to manage the context providing on a log level instead of source level. In the long term, and to further reduce the log monitoring overhead, it might be beneficial to apply prompts per source instead.

In the last week of my internship, specifically for the Gemini AI implementation, a certain prefix was added to the prompt managing. This means that while most context AI models will use the assigned prompt on log level, Gemini uses a general prompt which is inferred from the last added prompt with that certain prefix.

The screenshot shows the 'Prompts' management interface. At the top, there is a navigation bar with 'Logs', 'Sources', 'Prompts', and 'Issues'. A 'Create prompt' button is located in the top right corner. Below the navigation bar is a table with the following columns: ID, Name, Prompt Text, and Actions. The table contains four rows of prompts. Below the table, there is a note: 'When creating prompts, prefixing the prompt with "gemini-system-prompt" will make it the default system prompt of gemini'. At the bottom, there is a footer with the text: 'Data is stored locally. Models run locally, and no data is being sent to any third-party service. © 2024 - Syan Delbart'.

<input type="checkbox"/>	ID	Name	Prompt Text	Actions
<input type="checkbox"/>	2	Default2	Example system prompt text, this should be specific to the program.	<a href="#">✎</a> <a href="#">🗑</a>
<input type="checkbox"/>	1	Default	Example system prompt text, this should be specific to the program. (2)	<a href="#">✎</a> <a href="#">🗑</a>
<input type="checkbox"/>	3	gemini-system-prompt	This will be the first Gemini prompt.	<a href="#">✎</a> <a href="#">🗑</a>
<input type="checkbox"/>	4	gemini-system-prompt-v2	This will be used as this is the latest gemini prompt.	<a href="#">✎</a> <a href="#">🗑</a>

0 of 4 row(s) selected. Rows per page: 10 Page 1 of 1

When creating prompts, prefixing the prompt with "gemini-system-prompt" will make it the default system prompt of gemini

Data is stored locally. Models run locally, and no data is being sent to any third-party service.  
© 2024 - Syan Delbart

Figure 4 - A representation of the existing prompts

## 3.4 Manage Issues

Before the log monitoring tool, issues were managed manually. This means, when an issue was identified, this needed to be manually searched for in a certain table and modified accordingly. If it didn't exist, it needed to be added manually. This search process has a certain overhead and takes quite some time. It requires the person monitoring to already know the name of the issue.

### 3.4.1 Integration with external tooling

The current process required quite some manual work. The issue needs to be identified manually and needs to be documented manually. For this internship, it was important to be able to integrate the test monitoring tool with the current tooling. Eventually, the idea is that this tool can be used for any external tooling. This is where the use of the API keys is necessary for client-to-application interaction from the external tooling.

To integrate with this external tooling, a modal was added. This modal makes it possible to see the current identified issue, given there is one. Upon selecting another issue from the dropdown list, the corrected issue is modified through the API endpoint. Alternatively, a new issue can be created, which will take the user to the log monitoring tool itself. Ideally, the prediction the AI made is correct, and the user simply clicks the "Prediction correct" button.

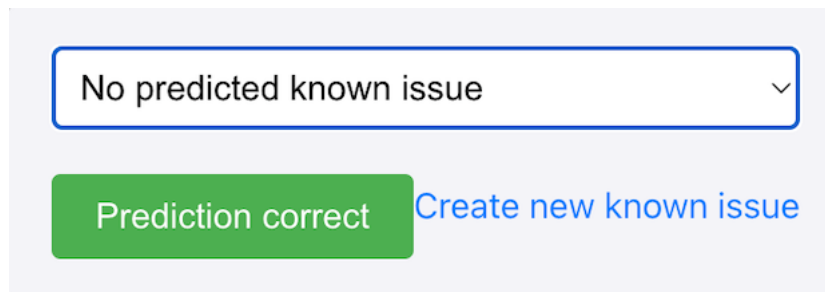
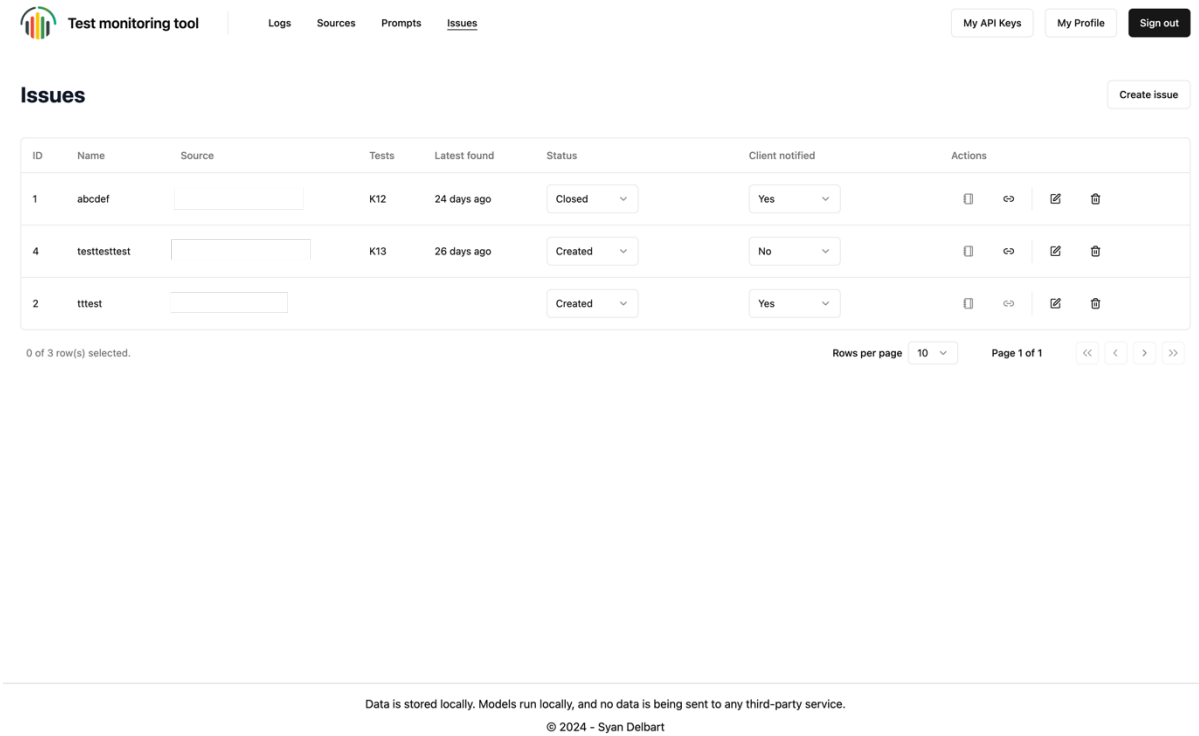


Figure 5 - A modal displaying the external interaction with the issues.

After the issue has been corrected, it will already be added to the issue table. Relevant information such as the test number, whether it was a functional test or KPI test and when the issue has been found latest, are all shown in the log monitoring tool already, without anyone needing to add that information manually. This already automates a large part of the issue documentation.



This process of identifying and correcting the issues adds the relevant information automatically in the log monitoring tool. This provides a structured way to keep track of all the issues, and already decreases a lot of manual work.

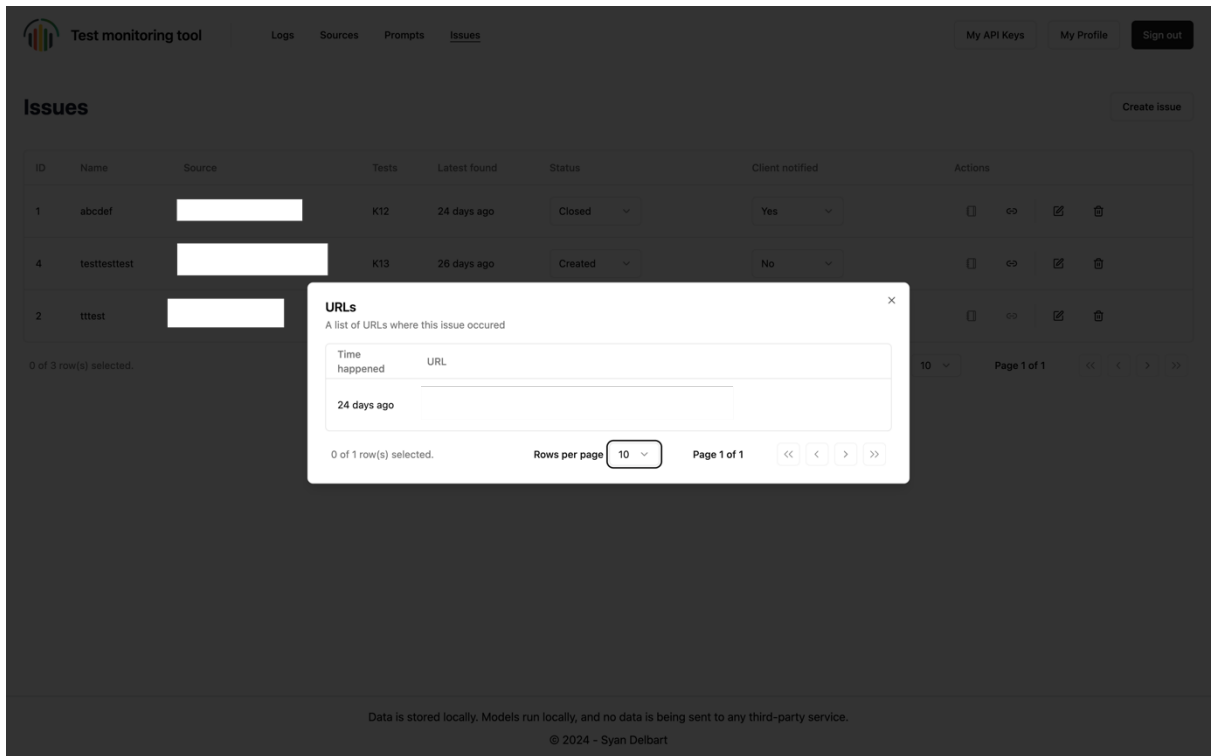


Figure 6 - A modal displaying the URLs where a certain issue has been found.

Even the URLs where the issue has occurred are being added automatically. When the manual approach was in use, these URLs needed to be added manually, and usually only some of them were included. With the automated method, all the occurrences are being logged.

### 3.5 Manage Logs

The most important part of the tool is being able to manage the logs. This “central” point of the tool is where everything comes together. A log is linked to the following:

- Prompt
- Source
- Issue (identified, corrected)

This means being able to manage each of the previously mentioned features are all important to correctly manage the logs.

The screenshot shows the 'Test monitoring tool' interface. At the top, there are navigation tabs for 'Logs', 'Sources', 'Prompts', and 'Issues'. On the right, there are buttons for 'My API Keys', 'My Profile', and 'Sign out'. Below the navigation, there are several action buttons: 'Refresh', 'Create log', 'Columns', 'Working on: Idle', 'In queue: 0', 'Selected: 0', 'Analyze', 'Reprocessing', 'Reanalyze', and 'Delete'. The main part of the interface is a table with the following columns: ID, Test Name, Start, Type, Progress, Identified Issue, and Actions. The table contains 10 rows of log entries. The first five rows have a 'Type' of 'ERROR' and a 'Progress' of 'COMPLETED' (8/8). The next three rows have a 'Type' of 'ERROR' and a 'Progress' of 'NOT\_STARTED' (7/8). The last two rows have a 'Type' of 'DEFAULT' and a 'Progress' of 'FAILED' (0/7). At the bottom of the table, there is a status bar that says '0 of 1000 row(s) selected.' and a pagination control showing 'Rows per page: 10' and 'Page 1 of 100'.

ID	Test Name	Start	Type	Progress	Identified Issue	Actions
30130		14 days ago	ERROR	COMPLETED 8 / 8	Select an issue	▶ ⌂ 📄 🗑️ ⚙️
30129		14 days ago	ERROR	COMPLETED 8 / 8	Select an issue	▶ ⌂ 📄 🗑️ ⚙️
30128		14 days ago	ERROR	NOT_STARTED 7 / 8	Select an issue	▶ ⌂ 📄 🗑️ ⚙️
30127		14 days ago	ERROR	NOT_STARTED 7 / 8	Select an issue	▶ ⌂ 📄 🗑️ ⚙️
30126		14 days ago	ERROR	NOT_STARTED 7 / 8	Select an issue	▶ ⌂ 📄 🗑️ ⚙️
30109		21 days ago	DEFAULT	FAILED 0 / 7	Select an issue	▶ ⌂ 📄 🗑️ ⚙️
30108		21 days ago	DEFAULT	FAILED 0 / 7	Select an issue	▶ ⌂ 📄 🗑️ ⚙️
30107		21 days ago	DEFAULT	FAILED 0 / 7	Select an issue	▶ ⌂ 📄 🗑️ ⚙️
30106		21 days ago	DEFAULT	FAILED 0 / 7	Select an issue	▶ ⌂ 📄 🗑️ ⚙️
30105		21 days ago	ERROR	COMPLETED 7 / 7	Select an issue	▶ ⌂ 📄 🗑️ ⚙️

Figure 7 - A representation of the existing logs.

The above figure is a representation of the logs table. This table is more advanced than the others. The reason for this is because the AI gets executed at the log level. Another fact is that there are simply a lot of logs being added

daily, which makes being able to a sort and filter the information more important and necessary.

### 3.5.1 Reactivity

The log table refreshes at a given interval. This means it fetches newly added logs. To make the user-experience optimal, these logs are not automatically added to the table. Rather, the table must be refreshed manually before the new logs will be appended to the table.



Figure 8 - The refresh button, disabled, as there are no new logs.

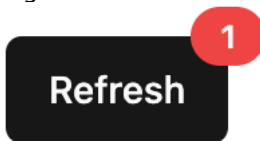


Figure 9 - The refresh button, enabled, as there are new logs found.

Not only are newly created logs being added in this way. Fetching the state of the AI progress is also done reactively. This means the progress can be observed in real-time.

All this is done through simple, periodic, fetch calls. An alternative was to use Web Sockets. Whilst this method is more lightweight, the implementation in this specific framework was quite tedious. Therefore, the choice was made for a more stable method which has somewhat more overhead. If this application would be used by a large amount of people at the same time, using WebSocket would be the preferred way.

### 3.5.2 Context AI Models

The so called "context" AI models are essentially text2text models. These take in the log information and output a context which might contain a clearer idea of what the issue is, and/or might contain a possible solution. Currently, multiple models are being used. This is mainly so because there are a lot of advancements regarding AI, and new models are being trained almost daily. Being able to identify the most performant model depends on a lot of factors. This feature makes it possible to use multiple models, without having to test them one by one.



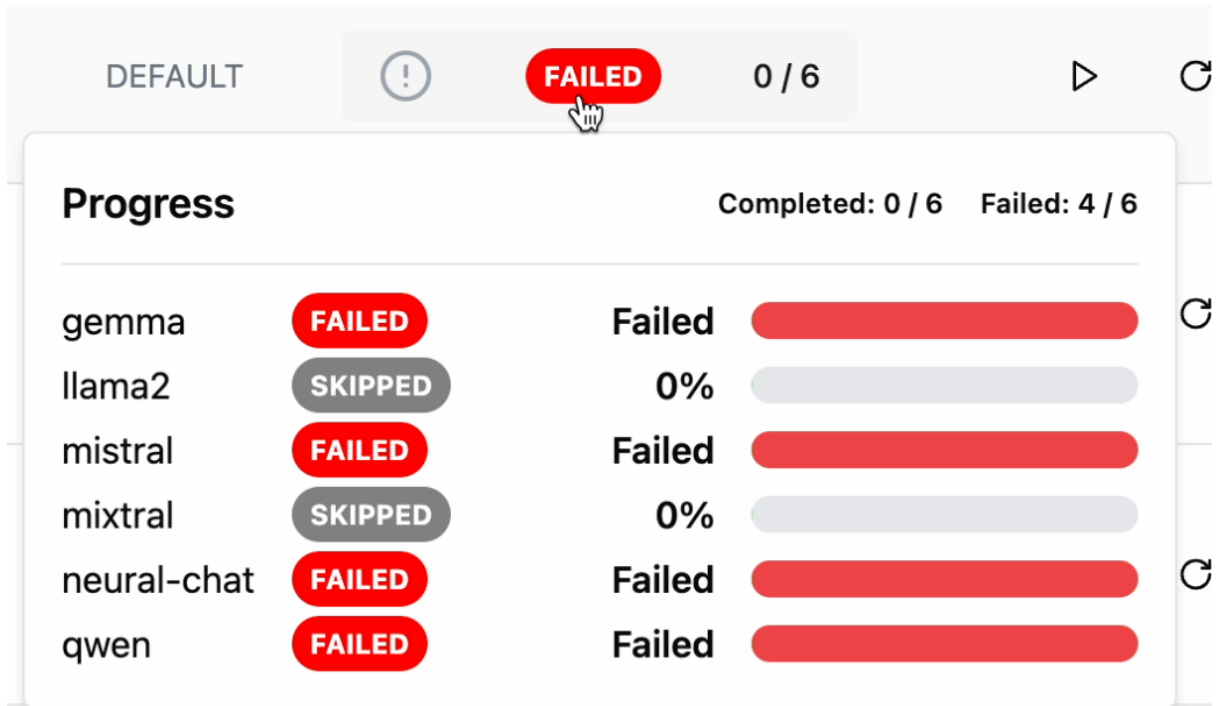


Figure 10 - A popover menu showing the progress of each individual context AI model.

Clicking on the Progress column within a log entry will open a popover. The individual progress per model can be observed there. The combination of all these states will be used to show the current state of the log context providing.

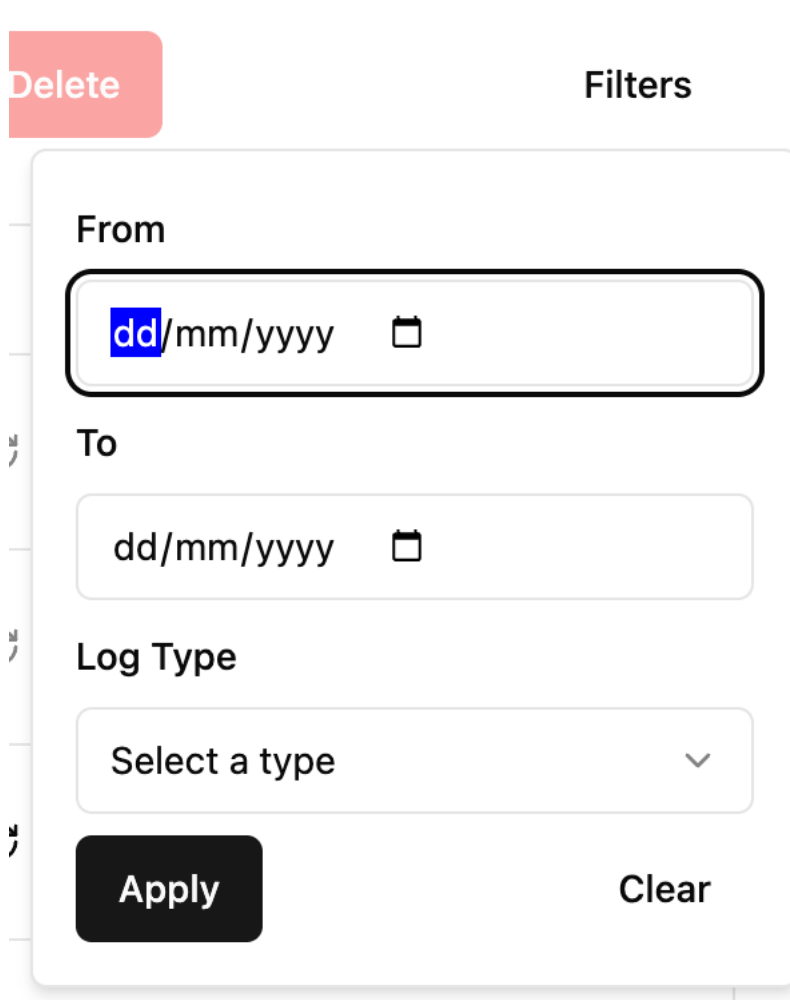
There are 5 different states:

State	Explanation
Failed	The analyzing failed, and an error message will be shown on hover. This is caused by the following issues: response provider crashed, the model is not installed
Skipped	This happens when the model is in the “skip” list. This list is especially useful on development, since not all models can be run on the development machine.
In queue	This log will be analyzed once it has reached first place in the queue.
In progress	The log is currently being analyzed.
Complete	The log has been analyzed successfully.

These states are determined per AI model per log, as each model is executed separately.

### 3.5.3 Filtering

The filtering for these logs is done through a dedicated button.



The image shows a popover menu titled "Filters" with a "Delete" button in the top left corner. The menu contains three input fields: "From" (with a text input containing "dd/mm/yyyy" and a calendar icon), "To" (with a text input containing "dd/mm/yyyy" and a calendar icon), and "Log Type" (a dropdown menu with "Select a type" and a downward arrow). At the bottom of the menu are two buttons: "Apply" and "Clear".

Figure 11 - A popover menu displaying the filter possibilities.

This button opens a popover menu which allows the user to specify the dates and log type of choice. It might be relevant to only show all the failing logs in the previous 30 days. The filtering is done on server-side. First this feature executed on the client-side, but this was not scalable as the number of logs is increasing daily.

### 3.5.4 Column filtering

This feature allows the user to select only certain information regarding the log.

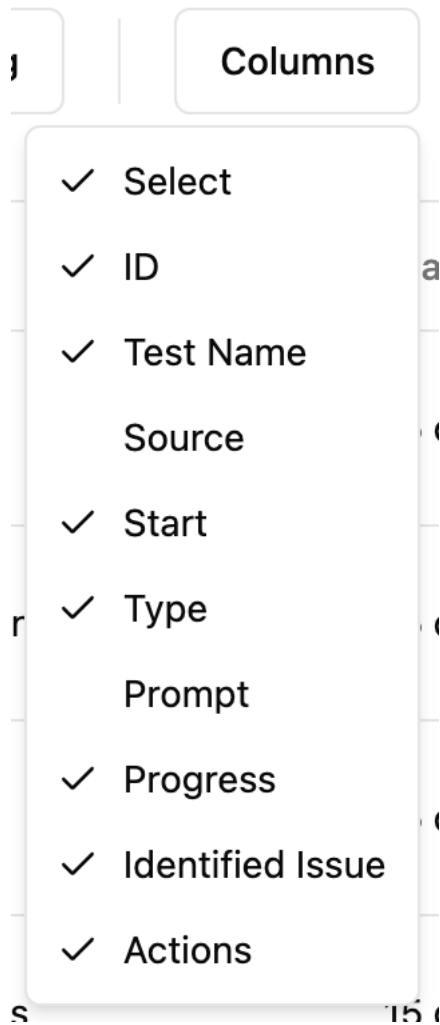


Figure 12 - A dropdown list consisting of all columns in the log table, can be enabled/disabled per column.

The state of these columns is stored in the local storage of the user. This means that the state remains on reload, as it is stored in the browser data storage. This is different from the filtering feature, which modifies the URL and does not store its state on the client-side.

### 3.5.5 Bulk action buttons

These buttons allow the user to do bulk operations on selected logs. These were especially useful while developing the tool, since changing the underlying code might require re-analyzing some logs or reprocessing them. The difference between analyzing and re-analyzing is that re-analyzing will only analyze jobs that have failed/skipped.

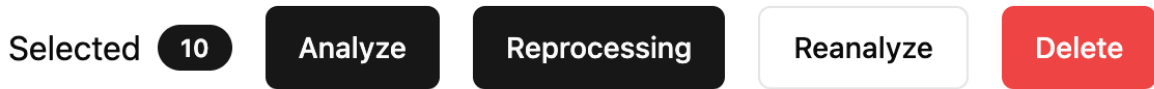


Figure 13 - Bulk action buttons, these are buttons that can be used for controlling multiple logs at the same time.

### 3.5.6 Preprocessing

Preprocessing of the log message is important. This is different from AI preprocessing but is necessary to remove unnecessary characters from the message before inputting it to the models.

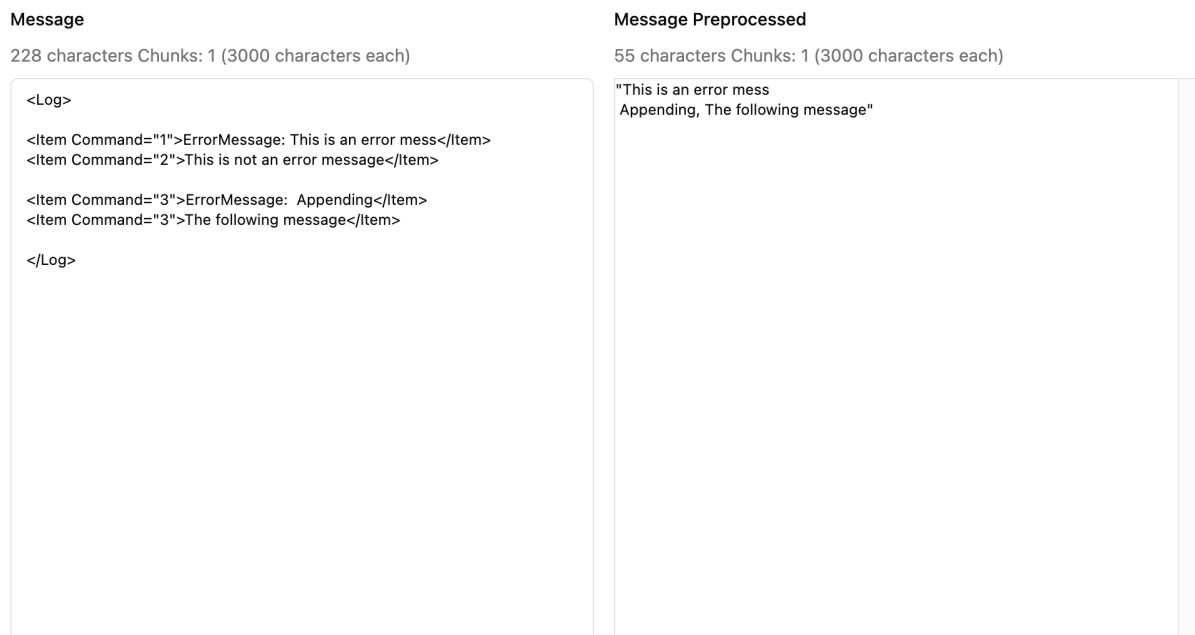


Figure 14 - A display of the initial message (left) and the preprocessed message (right), which displays the transformation of the message.

As seen in the figure above, this is where the preprocessing happens. The client can see the preprocessed message real-time and can modify the message if necessary. The preprocessing function executes on the client-side as well as server-side. As seen in this example, the input is a certain XML input. The output is condensed to only contain the error messages. For the current implementation, only XML input was relevant. This can however be expanded as need be. Input such as CSV, HTML, ... can also be supported if this is required in the future.

### 3.5.7 Run through AI (Classification)

The AI classification is done with the use of TensorFlow. Training the model is currently done in the following flow:

1. Create dictionary from all words in all logs  
= convert words to a number, an "index")
2. Tokenize all logs  
= convert all log words to the corresponding index in the dictionary
3. Take the first and last 500 tokens of each log, creating a vector of 1000 tokens.
4. Combine this vector with the "corrected issue"  
the corrected issue = the manually selected issue
5. Fit the dataset on the model.

This results in a trained model. This model can then be used to predict new issues. Over time, the identified issues will be compared to the corrected issues. The number of times the identified issue is the same as the corrected issue will be the success factor. The idea is to maximize this success factor over time and improve the model.

The model automatically improving based on correcting the wrongly identified issues is not a feature that is currently implemented but can be implemented once there is enough training data. In the current implementation, there was simply not enough training data to benchmark and finetune the model optimally.

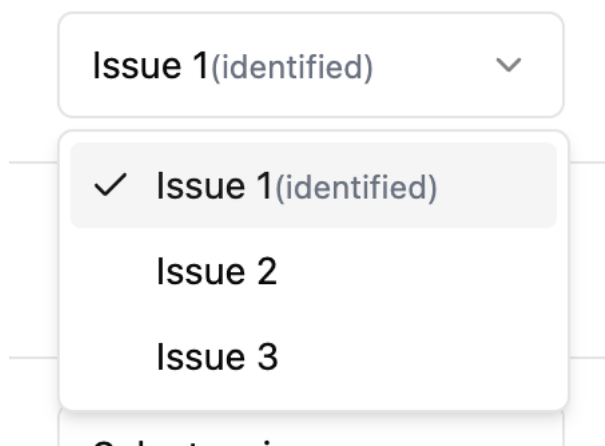


Figure 15 - A dropdown list for linking a log to an issue, the identified issue is already selected, and the corrected can be selected alternatively.

In the dashboard itself, this is represented in the "Identified Issue" column, where the identified issue is selected by default. This is the issue that is identified by the trained AI model. Clicking another issue will make the corrected issue different from the identified issue, and this information will then be used in the training of the model.

### 3.5.8 Visual AI model

The currently used context AI models could take the log message, but nothing more. As often, the logs are a textual representation of a certain video file, it was beneficial if the AI model could take in this information as well. Using open-source software, this would have been a challenge, too large to fit in the scope of the internship. However, using Google's Gemini AI model, it was possible to pass the video file as well as the log file. Gemini AI can not only tokenize text, but videos as well. Therefore, the Gemini AI was added to the application. This required some modifications, but most of the foundations were already existing. The support for multiple AI models is already implemented, and fetching assets from the internal service was also supported already.

Essentially, this means that the only step being added is that the video file should be fetched along with the log message.

The Gemini AI has an existing library for NodeJS, which made it easier to interact with the AI itself. The steps were as follows:

- Download the video file (save in temporary folder)
- Upload the video file.
- Await ACTIVE state on video file instead of PROCESSING.
- Save the video file reference for later use in API calls.
- Send the log message along with the video file to the Gemini AI model.

As previously mentioned, Gemini uses its own "global" system prompt, this is because the video files should all be processed in the same way. Asking the AI to first explain step-by-step what happens in the video proved to be more useful than directly requiring it to make the connection between the video file and the log file. Instructions like these can all be provided in the system prompt.

## 4 APPLICATION STRUCTURE

### 4.1 Retrieving the logs

There is no streamlined method to retrieve the logs. The source from which the logs must be retrieved is in this case an internal service. There are API endpoints that can be used to retrieve the latest logs; however, these do not contain all the relevant information. This endpoint responds with information such as the log name, log type, and so on, but is missing one crucial aspect. The log message itself. This log message is an artifact which is provided on another service, which can only be accessed when the user is logged in. Using the existing API access this asset cannot be retrieved. This means the only viable way to retrieve the log message is to log-in and navigate to the asset's URL path. This requires the use of a library called "Puppeteer". This library simulates a real browser, in which there is in fact a real client.

For each log, the same process was repeated:

1. Start a browser instance.
2. Navigate to the login page.
3. Login to the internal service with user credentials
4. Navigate to the log message's URL path.
5. Download the visible text on the webpage.

Once all the visible log message text is downloaded, it can then be added to the log itself. Sadly, this process is not very optimal, as starting a browser is a relatively long process.

In the application itself, a BrowserPool was created which is essentially a global defined browser pool with a set number of browsers that are already logged in and ready to be used. Due to instability (browsers closing, login sessions invalidating, ...) this method was not very optimal, even though it was way faster.

The current state of the project starts a new browser for each individual log, but optimally this would be a browser that gets reused. Even more optimal would be an API endpoint from which the log message can directly be retrieved.

#### 4.1.1 Running the context AI

Running the AI models require quite some time and computational power. For this, a separate service is being used. This service is called "Ollama". This allows LLMs to be ran locally on either CPU or GPU power. After downloading the relevant models, these can be accessed from an API endpoint. The previously retrieved log message can then be provided together with some other relevant

metadata. The API responds with a context response. This can only start once the log message has been retrieved.

## 4.2 Queue system

Retrieving the logs as well as running the AI model are tasks that take quite some time. It cannot be calculated before being executed; therefore, a queue is necessary. Another reason why this is important is because running the AI context models can only happen sequentially, not in parallel. It is important to make sure only one model runs at the same time. These things can be specified in a queue.

First, the queue service "Quirrel" was being used. This service was specifically created for NextJS but was optimized to run in an edge environment. Even though it was possible to run locally, there were some performance issues which made working with this service not ideal, as well as the service not receiving future support. Eventually, BullMQ was used. Compared to Quirrel, BullMQ could be integrated directly in the application itself, whilst Quirrel needed its own server. This added stability made BullMQ even more suitable.

### 4.2.1.1 Retrieving the logs

The amount of queue jobs running concurrently depends on the number of available browsers. Since a new browser is created for each log independently, this should be limited to a reasonable number. This number can be modified depending on the available computational power.

### 4.2.1.2 Running the context AI

For this internship, it is assumed that only one GPU is available. The amount of GPU power available will essentially determine the number of jobs that can be ran concurrently. In this case, this is limited to one.

For each of these tasks, separate, isolated, queues are being used.



## 4.3 Deployment

### 4.3.1 Quality insurance

For quality assurance, Azure Pipelines were utilized. Azure Pipelines are automated scripts that run on the uploaded code. These pipelines can be configured with specific conditions, such as requiring successful script execution before allowing the code to be integrated into the production codebase. During this internship, Azure Pipelines were consistently used to ensure that the latest code met certain standards.

#### 4.3.1.1 Building the application

One of the tasks of the Azure Pipeline was to build the web application. This involved specifically building the Dockerfile related to the Next.js web application. If this build failed, it indicated an error in the code, such as a typing error or another issue that would prevent the application from being built in production.

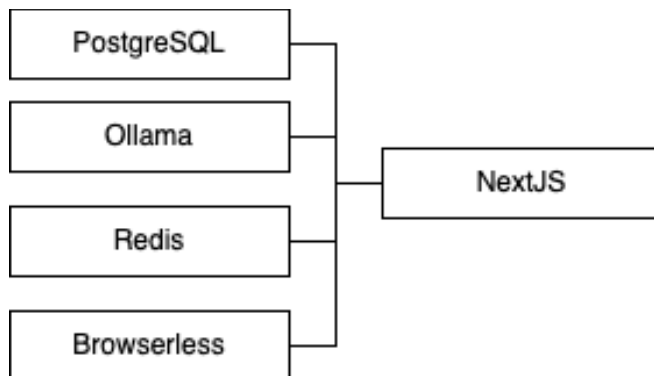
#### 4.3.1.2 Analyzing the code

For code analysis, a third-party provider called SonarCloud was used. SonarCloud ensured that the code met a certain quality standard. If the code did not pass this analysis, the pipeline would fail, and the code needed to be improved before it could be merged into the production codebase.

### 4.3.2 Deploying the application

To deploy the application, Docker was used. There were several different services:

- NextJS application (frontend + API)
- PostgreSQL (database)
- Redis (database for queues)
- Browserless (spawns browsers containerized)
- Ollama (Running AI models locally)



*Figure 16 - A simple representation of the application architecture*

These containers connected with NextJS directly. Running this required the installation of GPU passthrough so that the AI service could use the GPU power.

To use all these services together, an appropriate Docker Compose file was utilized. All services, except for Next.js, were based on existing Docker images. The Next.js application had a dedicated Dockerfile that needed to be built before use.

An internal Nginx proxy manager was available, ensuring that the application was reachable through a dedicated URL instead of just an IP address. This provided multiple benefits, including the provision of a valid SSL certificate, which enabled encrypted traffic.

## 5 ANALYZING PHASE

In the first weeks of the internship, it was crucial to ask questions, primarily related to AI and data. Understanding that AI requires specific instructions and clear data input was essential. Therefore, it was important to inquire about the inputs being used to accurately define the expected outputs.

While there were several specific questions, they are not detailed in this document as they pertained more to the project's implementation rather than the initial analysis. The answers have been summarized because they encompass a variety of information, including screenshots, which do not necessarily clarify the outcomes of these questions.

### 5.1 Questions logs

#### 1. Which logs need to be monitored?

The logs requiring monitoring for this internship are limited to an internal service that hosts several different programs. The goal is to make the tool as broad as possible, allowing it to connect with other services in the future.

#### 2. How much data is available, and is it enough?

The internal service stores all logs indefinitely, ensuring enough data is available for analysis.

#### 3. Where are the patterns and trends located in these logs? How can an issue be identified?

The log structure remains consistent, with specific failure messages differing between programs. For instance, a web application might report issues related to page loading, while a video stream could have frame rate drop issues.

#### 4. Are the outcomes of these logs already available?

Yes, the outcomes of each log are accessible via an API endpoint, using indicators like Passed/Failed. False positives and false negatives are also tracked, but these must be manually adjusted.

#### 5. Is there a formally determined connection between the log and the issue?

Often, the log describes the issue, but sometimes this is vague, especially for issues like video flickering that are only visible in a video stream. To clarify this connection, the video stream should be tokenized and analyzed alongside the log.

#### 6. Is there a formally determined connection between the issue and the solution?

Not always. It is not guaranteed that each issue has a solution that AI can infer. While documented solutions can help, making this comprehensive is challenging. Different services might have vastly different issues and solutions. For the scope of this internship, it is assumed there is no formally determined connection yet.

#### 7. Do the logs need to be isolated from each other for actual issue identification?

Isolation does not appear necessary for the implementation. The sources include video streams, web applications, and mobile applications, which often have related issues. The log output is quite generic across different

programs. However, as the tool expands to other services with different log structures, isolation might become necessary.

**8. Can the logs from different sources be combined in one model? Is a separate model required for each log?**

For this internship, it is assumed that logs from different sources can be combined in one model. Given the similarity in log outputs, a single model is used for all sources. In future expansions, using multiple models might be beneficial, offering improved accuracy at the cost of reduced training data. This should be benchmarked and cannot be decided beforehand.

## 5.2 Questions AI

**1. Are the (currently existing) tools to monitor logs and analyze them?**

While there are existing tools available, they are mostly proprietary and often tied to specific platforms, offering limited control over the models. Although these tools exist, they are not ideal for this use case.

**2. Are these tools “good enough”? Is it necessary to make a custom solution?**

As mentioned in the first question, the existing tools are not sufficient for this use case. They tend to be costly and lack the flexibility to be modified. While using third-party AI models is an option, no tool currently integrates well with existing services. Additionally, since it is preferable to have the entire application on-premises, existing tools do not meet these requirements.

**3. How can there be a guarantee that the used AI does not share/store information?**

This can be ensured by training and using models locally. However, this approach requires consideration of factors such as hardware power (GPUs) and scalability. Some third-party model providers offer options to disable data usage in training and prevent storage, but these measures do not provide absolute guarantees.

## 6 CONCLUSION

The primary goal of this project was to develop a fully functional application. Initially, the freedom in architectural and design choices was daunting, but it led to a deeper understanding of key technical concepts. I gained extensive hands-on experience with various technologies, particularly in distinguishing between cloud-based and on-premises applications. Key considerations included deployment machine performance, application scalability, and system isolation.

The project, although a Proof of Concept, felt more like a real product. Prioritizing functionality over optimization was necessary due to time constraints, especially during bi-weekly demos. This approach highlighted the need for better scalability testing and a more integrated workflow.

Necessary optimizations include:

- Scalability: This aspect wasn't fully tested, especially in scenarios involving multiple GPUs or AI services.
- Workflow: The code flow could be more seamless, as some elements felt appended rather than naturally integrated, partly due to dynamic project scope adjustments.

Working in an agile environment allowed for many adjustments but also presented challenges, emphasizing the importance of better scope estimation for future projects.

Overall, the project successfully met its objectives and provided significant insights into developing robust, scalable applications with a clear focus on practical implementation over theoretical optimization.

## 7 GLOSSARY

<b>Term</b>	<b>Definition</b>
Kanban	A project management methodology that focuses on visualizing work, limiting work in progress, and optimizing flow. It uses a board with columns and cards to represent tasks and their status, allowing teams to manage work more flexibly and continuously improve their processes.
Proof of Concept (PoC)	A demonstration or prototype used to verify that certain concepts or theories can be successfully applied in practice. It helps validate the feasibility and potential of an idea before committing to full-scale development.
AI Models	Computational algorithms designed to recognize patterns and make decisions based on data. These models can perform tasks like classification, regression, and clustering, and are used in applications such as natural language processing and image recognition.
API (Application Programming Interface)	A set of rules and definitions that allows different software applications to communicate with each other. APIs specify how software components should interact and enable the integration of different systems and services.
CLI (Command-Line Interface)	A text-based user interface used to interact with software and operating systems by typing commands. CLIs provide a way to perform tasks, run programs, and manage system resources using text input.
WebSockets	A communication protocol that provides full-duplex communication channels over a single TCP connection. WebSockets enable real-time, bidirectional communication between a client (like a web browser) and a server, making it ideal for applications like live chat and online gaming.
Tokenization	The process of converting words or text into numerical tokens that a machine learning model can process. Tokenization is a fundamental step in natural language processing, where text data is transformed into a format suitable for analysis by algorithms.
Local Storage	A web storage method that allows websites to store data on a user's browser. Local storage retains data across browser sessions, enabling web applications to save and retrieve information even after the browser is closed and reopened.

GPU Passthrough	A technique that allows a virtual machine to directly use a physical GPU. This enhances computational performance for tasks like training AI models and running high-performance applications that require significant graphical processing power.
Bi-weekly Demos	Regular demonstration sessions held every two weeks to showcase progress on a project. These demos provide opportunities to gather feedback, make adjustments, and ensure the project is on track.
Automated Testing	The use of software tools to execute pre-scripted tests on an application before it is released into production. Automated testing helps identify defects, ensure functionality, and improve the quality of software by running tests quickly and repeatedly.
Quality Assurance (QA)	A systematic process to ensure that products and services meet specified requirements and are reliable, maintainable, and fit for purpose. QA involves activities like testing, inspection, and review to detect and prevent defects in software development.
Docker Compose	Docker Compose is a tool that allows you to define and manage multi-container Docker applications. Using a YAML file, you can configure the services, networks, and volumes required for your application, enabling you to deploy and manage all your containers with a single command. It simplifies the orchestration and scaling of complex applications composed of multiple services.
Dockerfile	A Dockerfile is a text file that contains a set of instructions for building a Docker image. It specifies the base image to use, the necessary dependencies, configuration settings, and the commands to build and run the application. The Dockerfile enables the creation of a custom Docker image that can be consistently reproduced and shared across different environments.
Nginx proxy manager	Nginx Proxy Manager is a tool that simplifies the setup of Nginx as a reverse proxy server. It provides a user-friendly web interface to manage proxy hosts, SSL certificates, and redirects. It allows you to route incoming requests to different backend services, enabling seamless domain management and load balancing.
SSL Certificate	An SSL (Secure Sockets Layer) certificate is a digital certificate that authenticates the identity of a website and enables an encrypted connection. SSL certificates ensure that data transferred between the web server and browsers remains private and secure. Websites with SSL certificates use HTTPS protocol, providing a secure communication channel and building trust with users.