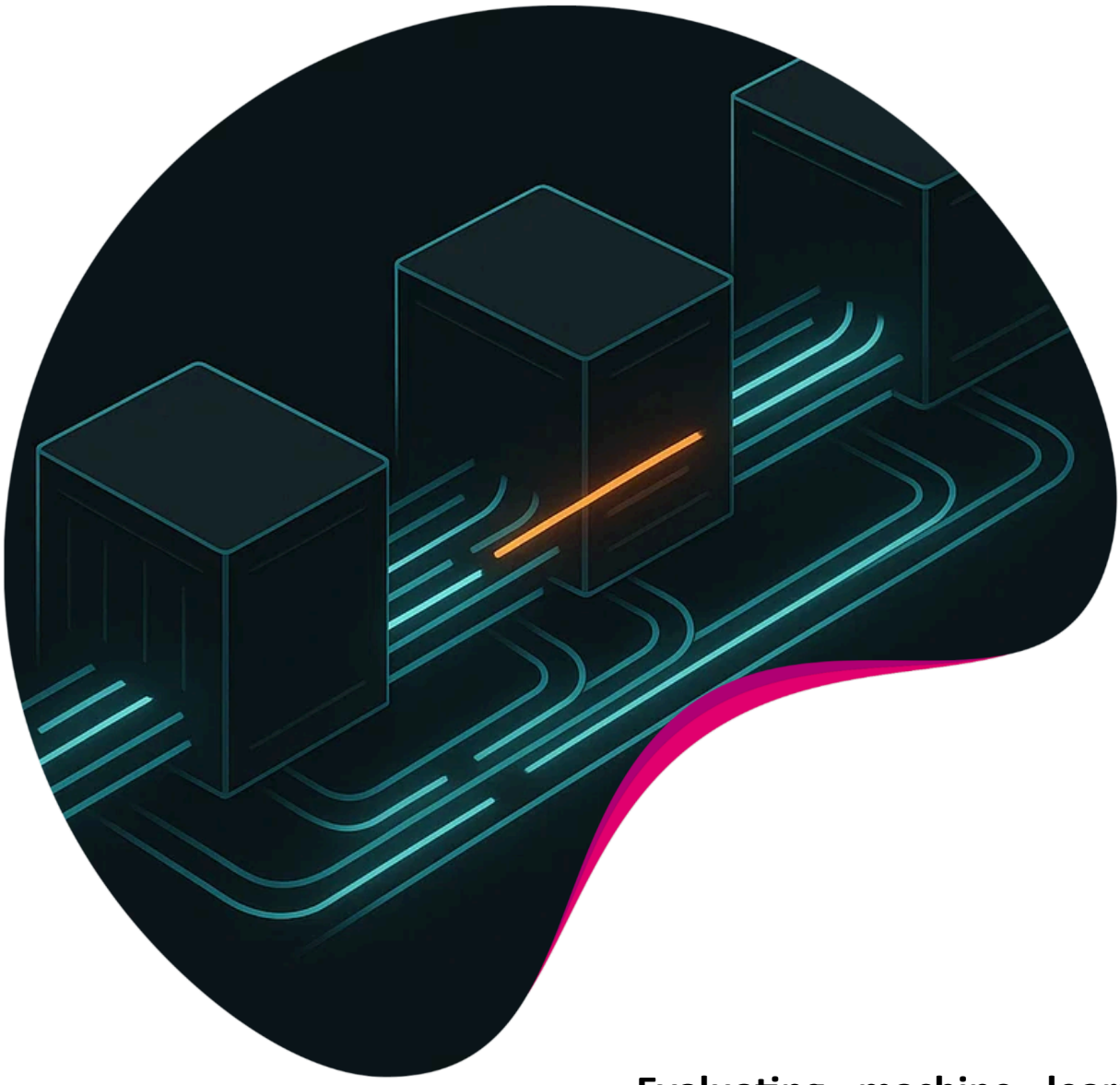
OST
Ostschweizer
Fachhochschule



Evaluating machine learning models in a SIEM environment

Creating a testbed for systematic evaluation of anomaly detection ML models in the ELK Stack

Print:

18.12.2025

Team / Authors:

Martin Arendar und Matteo Mahler

Advisor:

Prof. Dr. Daniel Patrick Politze

Abstract

This project presents the design and implementation of a testbed for detecting user behavior anomalies in web applications using machine learning within the Elastic Stack. Traditional Security Information and Event Management (SIEM) systems rely largely on rule based detection, which offers limited flexibility and struggles with complex behavioral patterns. Modern machine learning approaches address these limitations but introduce new challenges, especially regarding data availability, model evaluation, and integration into operational monitoring systems. The goal of this work is therefore not to build a single high-performing model, but to create an extensible and reproducible environment in which different machine learning models can be trained, deployed, and compared under identical conditions.

Because no suitable dataset for user centric anomaly detection existed, we developed a synthetic data generation pipeline that simulates user interactions in a controlled web application. A custom student management platform was implemented to produce ECS compliant logs representing normal and anomalous behavior. A traffic generator produces user flows and defined anomaly patterns over configurable time ranges, allowing the creation of reproducible datasets. These logs are ingested through Filebeat and Logstash into Elasticsearch, enriched through ingest pipelines (e.g., GeoIP), and transformed into session-level features using Elastic Transforms. This pivot from action-level events to session-level aggregates provides meaningful input for supervised machine learning models.

Two machine learning approaches are evaluated: Elastics built in Data Frame Analytics (DFA) classifier and a custom Random Forest model trained in Python using scikit-learn, imported into Elastic via eland. Both models operate on the same labelled session dataset, enabling direct comparison. A dedicated evaluation dashboard, the foundation of the testbed, visualizes confusion matrices, classification metrics, feature importance, and anomaly distributions. This dashboard allows users to compare models on standardized performance indicators and supports repeated experimentation with newly trained models.

The results demonstrate that the system functions reliably as a testbed. Data generation, ingestion, transforms, and dashboards work as intended, and both models can be evaluated consistently. While Elastic's DFA provides strong baseline performance with minimal configuration, the custom model highlights the challenges of feature engineering, data realism, and training strategy. Several findings indicate opportunities for improvement, particularly regarding model tuning, dataset variability, and automation of dashboard creation. The project concludes that a machine-learning-based anomaly detection workflow in Elastic is feasible and that the developed testbed offers a solid foundation for future research, such as improved model engineering, integration of real-world data, or real-time anomaly tracking.

Management Summary

Initial Situation

Security Information and Event Management (SIEM) systems play a central role in modern cybersecurity by collecting, correlating, and analyzing log data from different systems in once centralized place. Traditional SIEM platforms rely mostly on rule-based detection mechanisms. While these rules are easy to implement, they struggle with dynamic or previously unseen patterns and often generate false alarms.

Machine learning (ML) offers an opportunity to improve anomaly detection by learning past patterns instead of relying on static rules. However, ML introduces new challenges, especially regarding the availability of suitable datasets, the reproducibility of model evaluation, and the integration of ML models into existing operational monitoring systems.

The objective of this project was therefore not to build a single high performing model, but to design and implement an experimental testbed. This testbed enables the training, deployment, and comparison of different machine learning models for detecting user behavior anomalies within the Elastic Stack under standardized conditions.

A key challenge was the lack of suitable datasets for user centric anomaly detection in web applications. As a result, both the data foundation and the evaluation environment had to be developed as part of this work.

Approach and Technologies Used

To address the identified challenges, a structured approach was followed, covering data generation, ingestion, model training, and evaluation.

First, synthetic data generation was implemented to generate realistic log data. A simplified student management web application was created to simulate typical user interactions as well as predefined anomaly patterns. All generated logs follow the Elastic Common Schema (ECS), ensuring compatibility with the Elastic Stack.

A dedicated traffic generator controls user flows and anomaly patterns over configurable time ranges. This allows the creation of repeatable datasets for model evaluation.

The generated logs are ingested into Elasticsearch using Filebeat and Logstash. During ingestion, the data is enriched through ingest pipelines. Elastic Transforms are then used to aggregate action-level events into session-level features, which are more suitable for machine learning based behavioral analysis.

Two different machine learning approaches were evaluated using the same dataset. The first approach uses Elastics built in Data Frame Analytics (DFA) classifier, which provides fast integration and baseline performance. The second approach uses a custom Random Forest model trained in Python with scikit-learn and imported into Elastic via the eland library. This combination allows both ease of use and full model customization to be evaluated.

Results

The implemented system functions reliably as a machine learning testbed. Data generation, ingestion, transformation, and visualization operate as intended and support repeated experimentation.

Both machine learning approaches can be evaluated consistently using a shared dataset and standardized metrics. Elastics DFA classifier delivers strong baseline results with minimal configuration effort. The custom model highlights the importance of feature engineering, dataset quality, and training strategy, especially when working with imbalanced anomaly data.

A dedicated evaluation dashboard was created to visualize confusion matrices, classification metrics, feature importance, and anomaly distributions. This dashboard forms the core of the testbed and enables direct comparison between different models without modifying the underlying infrastructure.

Conclusion and Outlook

This project demonstrates that a machine learning based anomaly detection workflow within the Elastic Stack is feasible and effective. The developed testbed provides a reproducible environment for training and evaluating models under controlled conditions.

While the system already delivers meaningful results, several opportunities for future improvement were identified. These include exploration of more advanced machine learning models, enhanced feature engineering, more realistic data generation, integration of real world data and improved automation of dashboards.

Overall, the project establishes a solid foundation for further research and development in user behavior anomaly detection using machine learning in SIEM environments.

Table of Contents

Section I
Product documentation

1 Introduction

SIEM Systems are software solutions to gather security and event information from different sources, to be able to analyze them. In most cases they use rule based algorithms for anomaly detection. In recent time ML based approaches become more popular due to their superior flexibility in comparison to rule based systems, with the prediction of further growth in the upcoming years ([1] , page 1). This flexibility comes with the risk of performance loss ([2] , page 8). The focus of this work is to provide a testbed in which different ML algorithms can be tested in a unified manner to find out which delivers the best results in a standardized test procedure.

1.1 Problem definition, motivation, research goals

Rule based detection systems are not very flexible and may sometimes detect an anomaly when there is none (false positive), or do not detect one when it should (false negative). ML based systems try to solve this rigidity by learning past patterns and applying historical data to newly incoming events.

Elastic SIEM is an open source SIEM solution. The module landscape of the Elastic stack is so vast that analyzing the whole ecosystem would break the time frame of this project. Because of this the focus will be on the Elastic SIEM Security module, that already has ML capabilities built in, which allows for a fast working proof of concept. Elastic offers its data ingestion pipelines, ML models and analysis procedures commercially and thereby closed source. The problem that arises through this, is that the closed source approach prohibits the user to tune the ML models to their needs. To solve this issue we can use Elastics offering to integrate custom ML models. This is where the testbed becomes relevant. To find the best model, a standardized test procedure (a testbed) is needed for evaluation, which is what the focus of this project will be.

1.2 Related Work

The project consists of two main parts: First, using any kind of dataset to train the model for anomaly detection and integrating this model into the Elastic Stack. Second, building a testbed for the models evaluation.

From the beginning collecting data for model training was a big focus points, as it requires a lot of time and effort [3] , because of this we wanted to build upon existing work instead of starting from scratch.

We found some papers when researching existing datasets and other resources on anomaly detection using machine learning. Multiple of those papers [4] , [5] are using the “CSE-CIC-IDS2018” [6] dataset, which contains raw network traffic data, as well as already processed and labeled data for intrusion detection. The full size of all the data, raw and processed, goes beyond 100GB. We also considered some other datasets like the “KDD Cup 1999 Data” [7] . This dataset is older than the CSE-CIC-IDS2018 [6] but has the same goal and focuses on the intrusion part.

Since we were not working on intrusion detection and analyzing Layer 2 and 3 network traffic but rather anomalies in user behavior on Layer 7, we had to come to the conclusion that those datasets were not viable for our use case. We started searching for datasets that contain web application logs that were also labeled for anomalies. After searching in common places where such datasets are published, like on Kaggle [8] , [9] , GitHub [10] , Hugging Face [11] , and Google Scholar search, we could not find any suitable datasets.

Based on this lack of available datasets it became a necessity to create our own.

In addition to searching for datasets, we also reviewed literature that discusses how anomalies in web applications are defined and categorized. These sources helped us identify realistic behavioral patterns and attack types that later guided the design of our synthetic anomalies.

[12] describes how a ML model can be integrated into a WAF to detect anomalies in real time and is also superior to traditional methods. The following advisory gave some application security mitigation recommendations that we will apply or do exactly the opposite later [13] (page 6) to help simulating anomalies. One example are some common http status codes

that are associated with enumeration attacks. It also encourages to use a SIEM tool "[…] to facilitate active monitoring and threat hunting" [13] (page 6).

1.3 Research Gap

Existing datasets for user-behavior anomaly detection are either unavailable, incomplete, or unsuitable for our purposes, which makes creating a tailored dataset a necessary part of this research. While SIEM platforms like Elastic provide robust pipelines for log collection, processing, and anomaly detection, they typically focus on built in models and do not offer a standardized way to evaluate the performance of custom models in a controlled environment. To address this, our work proposes a testbed in which all variables (data collection, preprocessing, and infrastructure) remain constant, so that any observed differences in results can be attributed solely to the custom model under evaluation. This setup fills a clear gap by enabling systematic benchmarking and comparison of anomaly-detection models within a production-grade SIEM pipeline.

1.4 Approach

To address the research gap identified in the previous section and to create a testbed for evaluating ML models in an anomaly detection SIEM environment, a structured approach is followed.

Beginning by deploying and configuring the ELK stack and familiarizing ourselves with the relevant features of Elastic Security. This provides the technical foundation for data ingestion, processing, and analysis.

The next major step is data generation, which represents one of the most challenging aspects of the project. As discussed in Section 1.2 , no suitable dataset for user behavior anomaly detection was available, which left us with several possible strategies: searching for additional datasets, generating synthetic logs, or simulating real user traffic in a controlled environment. After evaluating these options, we decided to create our own dataset tailored to our use case.

Once the data is generated, it is ingested into Elasticsearch via Filebeat and Logstash. The raw logs require preprocessing and transformation before they can be used for machine learning. This includes pivoting the data towards user sessions through Elastic Transforms. With this prepared dataset, we train a custom ML model for anomaly detection.

Finally, a comparison of the performance of the custom model with the built in models offered by Elastic. For this purpose, an evaluation dashboard that visualizes relevant performance metrics and allows for standardized comparison will be created. These steps allow to answer the research question and assess whether the testbed effectively supports the evaluation of anomaly detection models.

2 Technology Stack and Implementation

This chapter provides a comprehensive overview of the technology stack and implementation details used in this project. It covers the key tools, frameworks, and architectural patterns that enable the collection, analysis, and anomaly detection of user behavior logs. The following sections detail the Elastic SIEM platform, the various system components, the software architecture using the C4 model, and the demo application with its database design and logging integration.

2.1 Elastic SIEM

Elastic supports this work by providing a comprehensive pipeline for collecting, processing, and analyzing log data. Beats and Logstash enable logs from various systems to be captured, parsed, and enriched before being indexed in Elasticsearch. Using Elastic Transforms, raw log events can be converted into structured, user centric summaries that are better suited for behavioral analysis. In addition, Elastic offers built in machine learning capabilities that can model typical activity patterns, detect anomalies, and surface unusual behavior directly in Kibana.

These capabilities form a strong foundation for integrating custom anomaly detection models. Elastic handles data ingestion, preprocessing, storage, and visualization, allowing machine learning models to focus on learning user behavior rather than dealing with low level data management. Predictions generated by custom models are be written into Elasticsearch by pipelines and visualized, enabling a unified workflow for analysis and monitoring.

Elastic SIEM provides third party machine learning solutions. Since these third party solutions get offered commercially they are closed source in most cases and do not provide insight into how the models work or which data they were trained on. It is therefore difficult to assess how well these models generalize to different environments or datasets.

For this project, a minimalistic yet feature complete Elastic SIEM configuration is provided in the project repository. The setup guide in Section 6 describes how to deploy the ELK stack and configure it to work with the demo application used in this project. Detailed explanations of individual components and terminology are introduced when they are first used throughout the documentation.

2.2 Stack

This chapter provides a high level overview over the used components in this project. All components will be described in further detail in the upcoming sections of this documentation.

2.2.1 ELK components

Our ELK Stack consists of the following components:

Component	Description
Elasticsearch	Elasticsearch is the core element of the ELK stack. It stores all data related to Elastic in any way (e.g. pipeline configurations, Transforms, dashboards, trained models, etc.).
Kibana	Kibana is the visual representation of Elastic. Elastic itself is accessible operable just via API calls. Everything displayed graphically is rendered by Kibana.
Logstash	Logstash is responsible for ingesting data into Elastic. It supports a wide variety of input sources, processing steps and output targets.
Filebeat	Filebeat is a lightweight shipper log data. It is typically installed on the servers that generate logs and is responsible for reading log files and sending the data to Logstash or Elasticsearch for further processing.

Tab. 1: ELK stack components

2.2.2 Application

The application used to simulate traffic consists of the following components:

Component	Description
Backend	• TypeScript • Express.js framework • Prisma as ORM • Docker for containerization
Database	• PostgreSQL • Prisma as ORM
Traffic Generator	• JavaScript • Faker.js for data generation

Tab. 2: Application components

The absence of a frontend is a conscious decision as we only need the traffic for our data, which is generated automatically without the need of any user interaction.

2.2.3 ML tooling

Tools for configuring, training and importing the custom machine learning model.

Component	Description
scikit-learn	scikit-learn (sklearn for short) is the library used for the custom machine learning model. It was chosen because this is the library supported by eland to import models into elastic. While lightgbm and xgboost libraries are also supported by eland [14] , the project team has gathered the most experience with sklearn during their studies. Also lightgbm and xgboost provide to much setup overhead when building a custom model.
eland	eland is an Elasticsearch Python client, which enables importing machine learning models into Elasticsearch [15]

Tab. 3: ML tooling components

2.2.4 Infrastructure

Tools and infrastructure that supported the project's development and coordination, but were not directly part of the implemented system.

Component	Description
Docker	Docker is used to run different components across the stack and manage their dependencies in their own environment. This helps to create a transferrable setup which can be deployed anywhere and provides the benefit of not having to install all dependencies locally during development.
GitLab	The version control system for the source code and documentation. Offered for free by OST including build pipelines this was the first choice.

Tab. 4: Infrastructure components

2.3 C4 architecture

The Software architecture diagrams are created using the C4 standard and visualized with IcePanel . The C4 model is an approach to document software architecture at different levels of detail, allowing for a clear understanding of the system's structure and components from a high-level overview down to low-level implementation details. It consists of four main levels:

1. Context Diagram
2. Container Diagram
3. Component Diagram
4. Code Diagram

Every level is described in the beginning of its individual chapter.

2.3.1 Context diagram

Context Diagrams provide a high-level overview of the system and show how it fits into its environment, including users and other systems it interacts with.

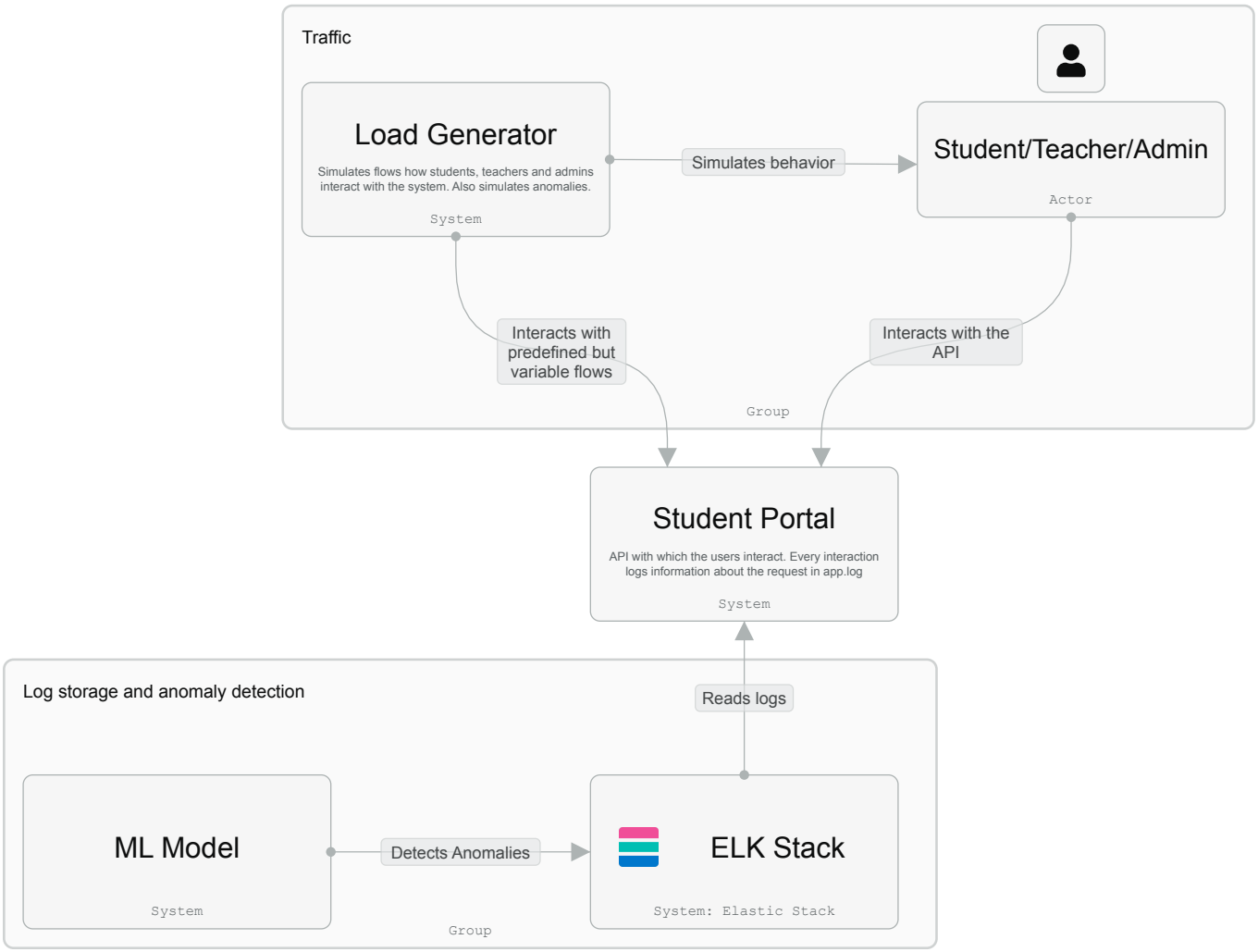


Figure 1: Context Diagram

Figure 1 shows a high-level overview of the systems and its actors. The `Traffic` group consists of the `Load Generator` and the users (Student/ Teacher/ Admin). It is important to mention, that users are a pseudo actor. Since the application is API based and not available to the public, the users are simulated by the `Load Generator`, which sends requests to the API and generates logs based on the requests made. The ELK Stack is a requirement for this project, and is used to collect and visualize the logs generated by the demo application. The custom ML Model is a separate system, that pushes a custom trained model to the `ELK Stack`, to be able to run it inside of Elastic's pipelines. ELK stack and the ML model are bundled together in a group "Log storage and anomaly detection" since they are closely dependent, which is the reason that they can also be seen as one system once it is deployed. The main system is the `Student Portal`. It is the core of this project and the source of the logs to be analyzed for anomalies.

2.3.2 Container diagrams

Container diagrams show an application's high-level technology and involved containers (applications, databases, etc.) and how they communicate with each other.

2.3.2.1 Load Generator

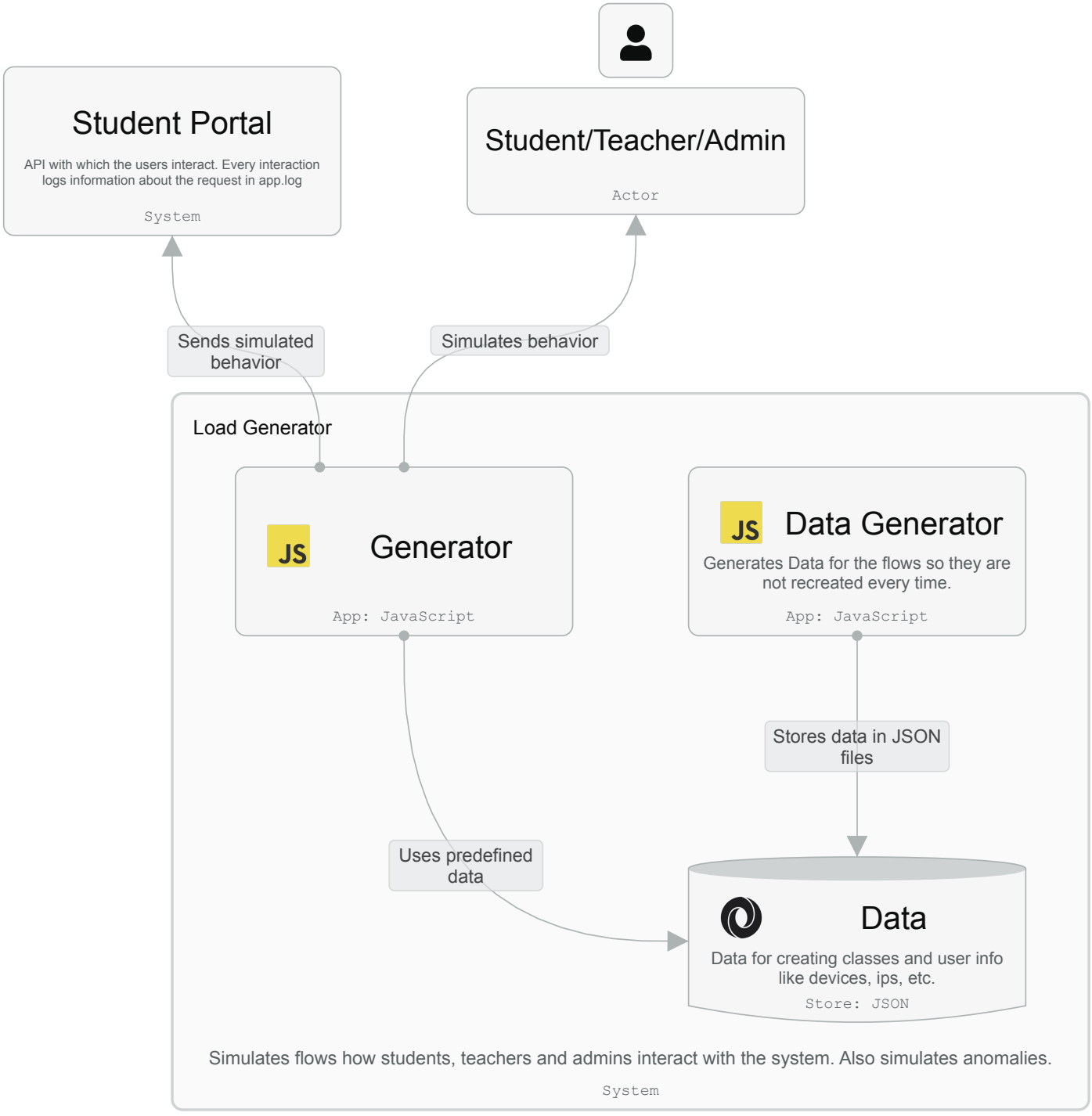


Figure 2: Load Generator - Container Diagram

The `Generator` is an application that simulates traffic to the demo application. It consists of the main container `Generator`, which is responsible for generating the traffic and sending requests to the demo application. A second small folder with scripts is used to create data beforehand, that can then be used during the run. The data is included in the git

2.3.2.2 Student Portal

Figure 3: Student Portal - Container Diagram

The `Student Portal` is a web application that serves as the source for the logs in this project. It is written in TypeScript using the Node.js runtime and the Express framework. All data is stored in a PostgreSQL database.

2.3.2.3 ELK Stack

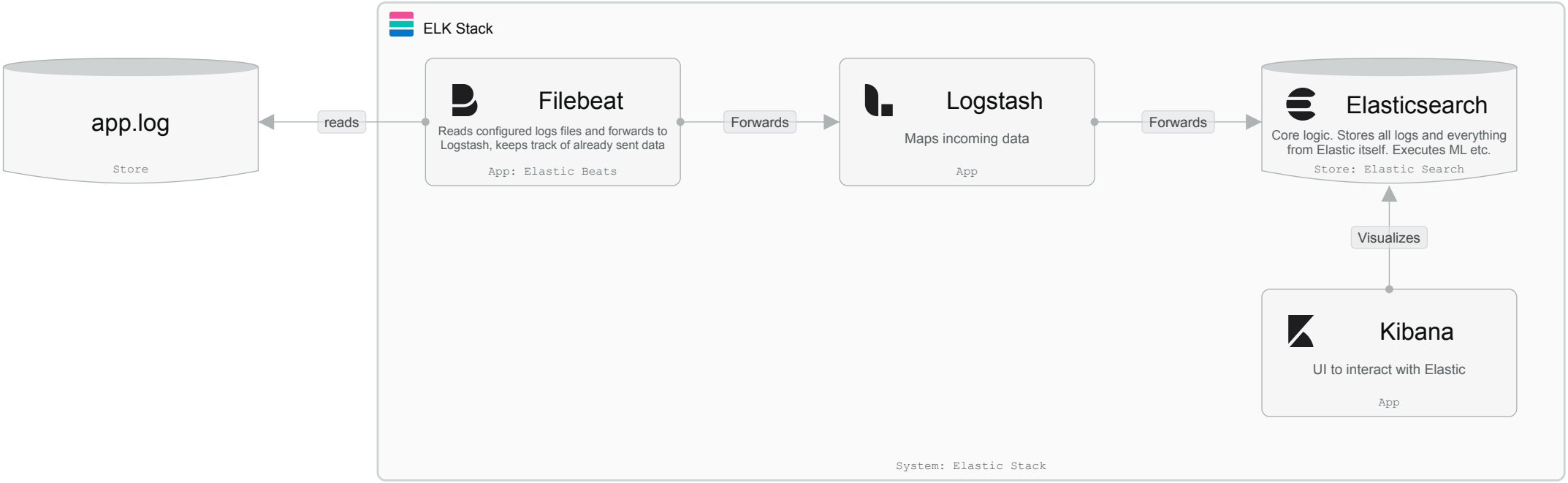


Figure 4: ELK Stack - Container Diagram

The ELK Stack consists of four main containers: a Beat (Filebeat in our case), Logstash, Elasticsearch, and Kibana.

2.3.2.4 ML Model

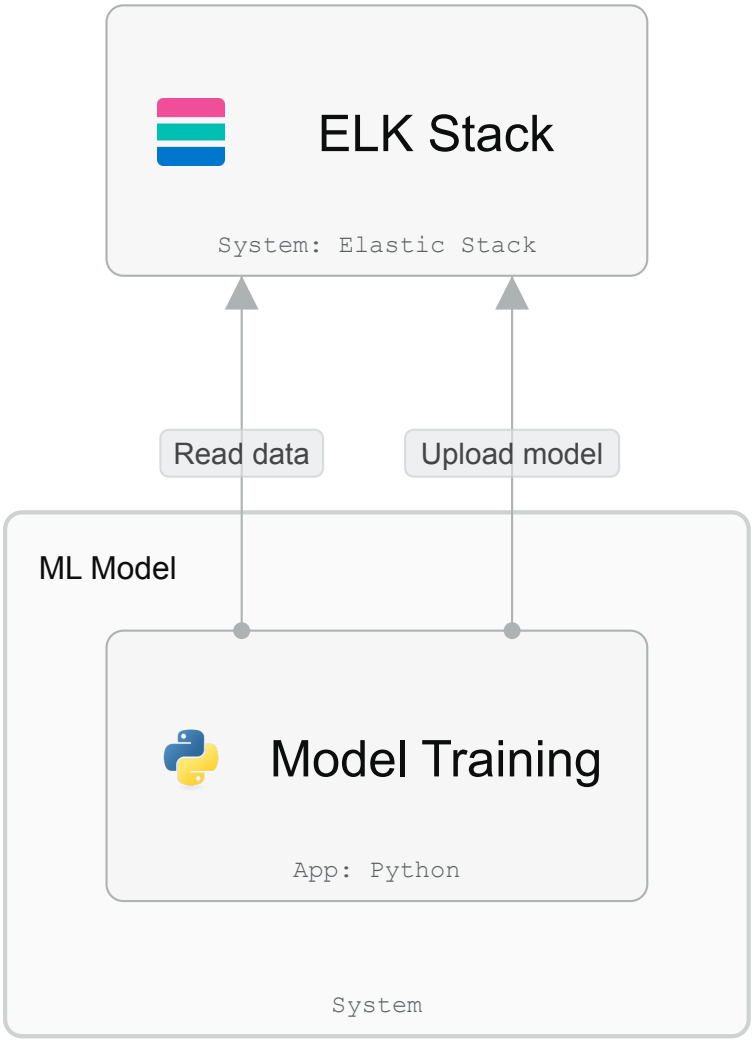


Figure 5: ML Model Container Diagram

The component view of the `ML Model` is very simple. The model reads the data from Elastics index and uploads a trained model.

2.3.3 Component diagrams

Component diagrams breaks down each container into its components. It gives a deeper insight into the internal structure of each container without describing the implementation details.

2.3.3.1 Data Generator

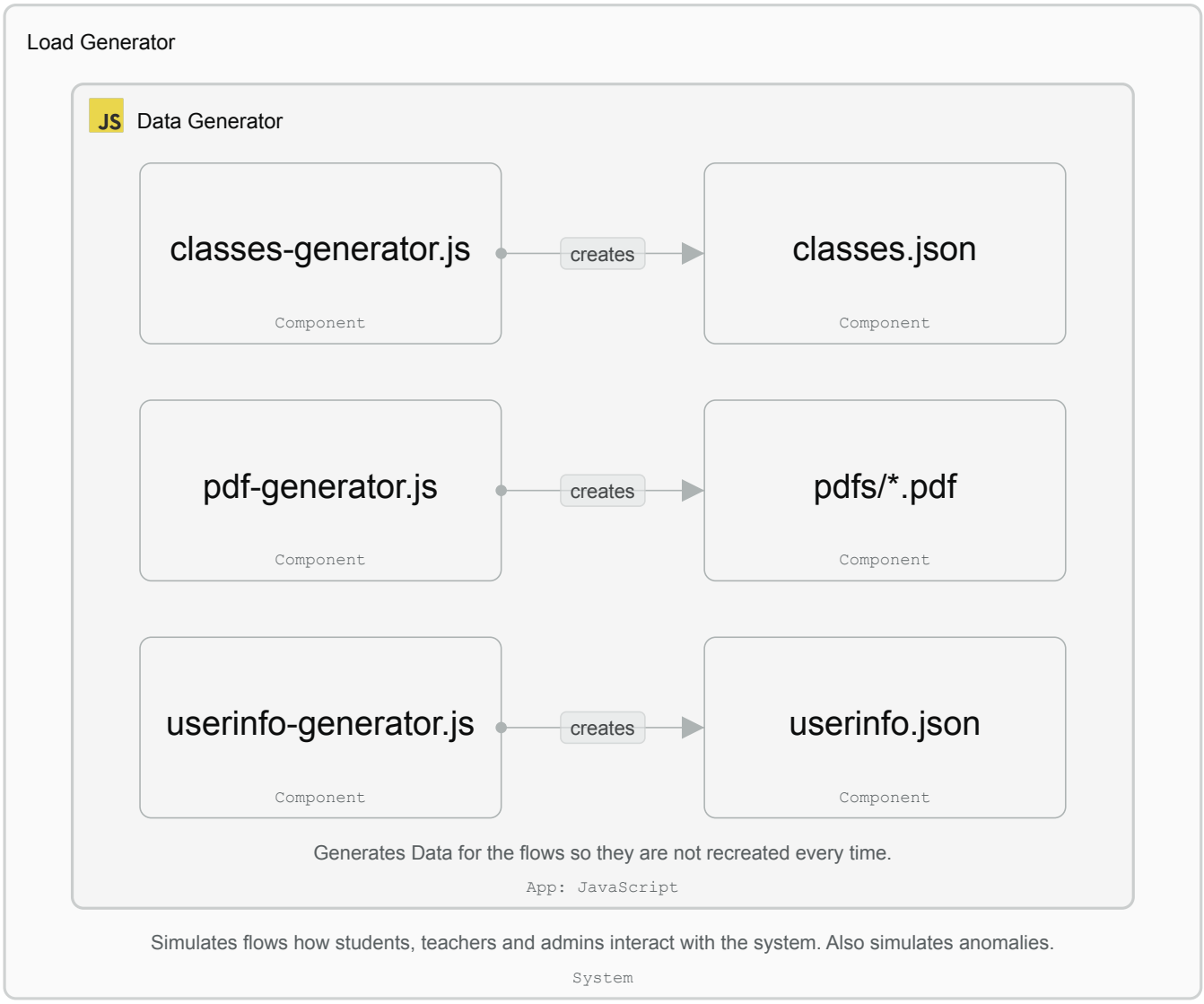


Figure 6: Data Generator - Component Diagram

The `Data Generator` contains a few short scripts, leveraging faker.js , which are used to create data beforehand that can then be used during the run of the `Data Generator`.

2.3.3.2 Generator

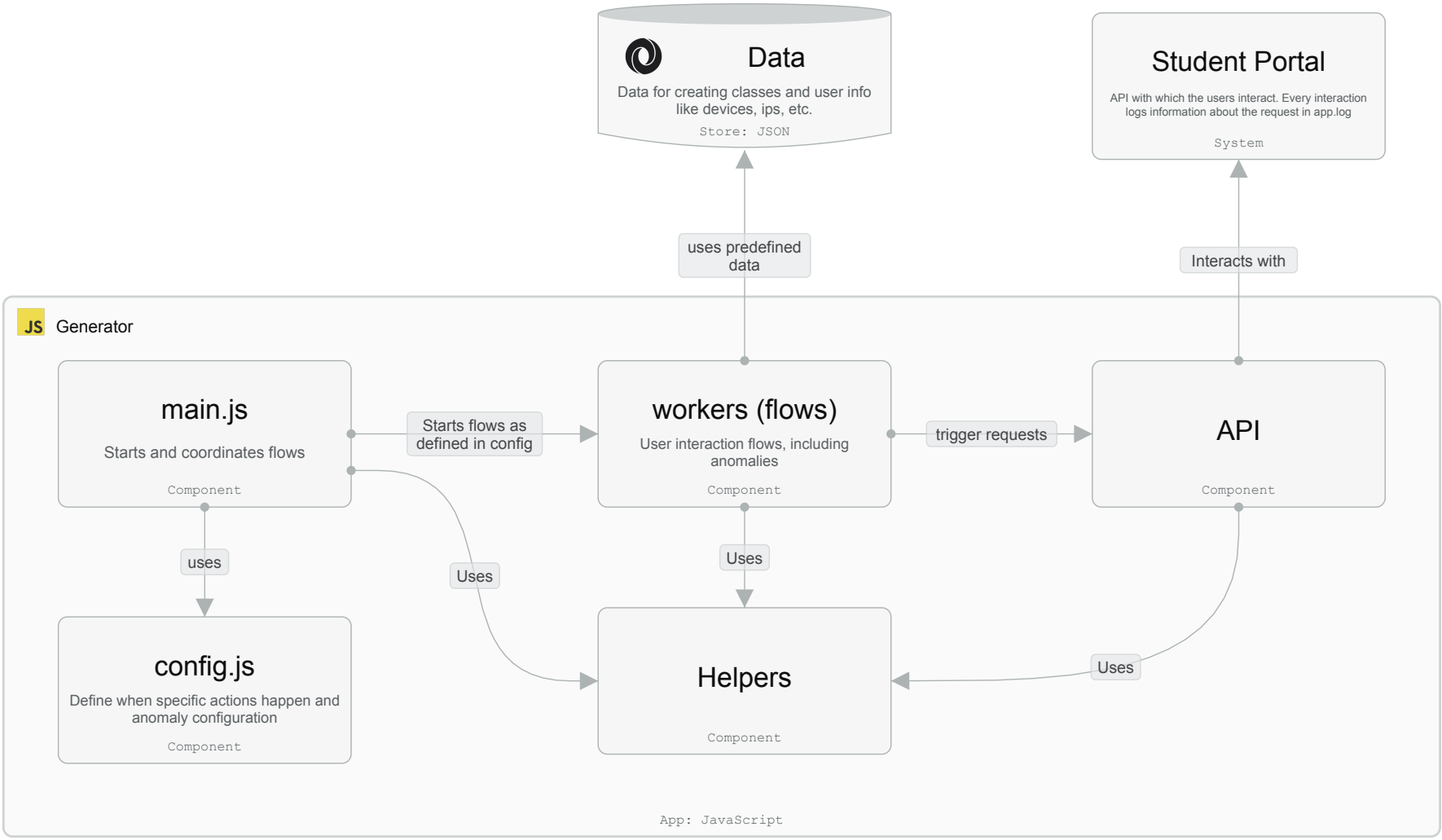


Figure 7: Generator - Component Diagram

The components of the `Generator` have the same structure as the code in the GitLab repository. There are 2 main files in the root directory and the rest is organized in sub-folders. `Helpers` consists of various utility functions that are used throughout the codebase. This includes functions to select random entries from the generated data, as well as the most important functionality, a helper to simulate time. Time simulation allows the creation of logs, that date over a span of weeks, in a short amount of time by speeding up the time internally in the application. The API folder contains all the logic to send requests to the demo application.

2.3.3.3 Student Portal

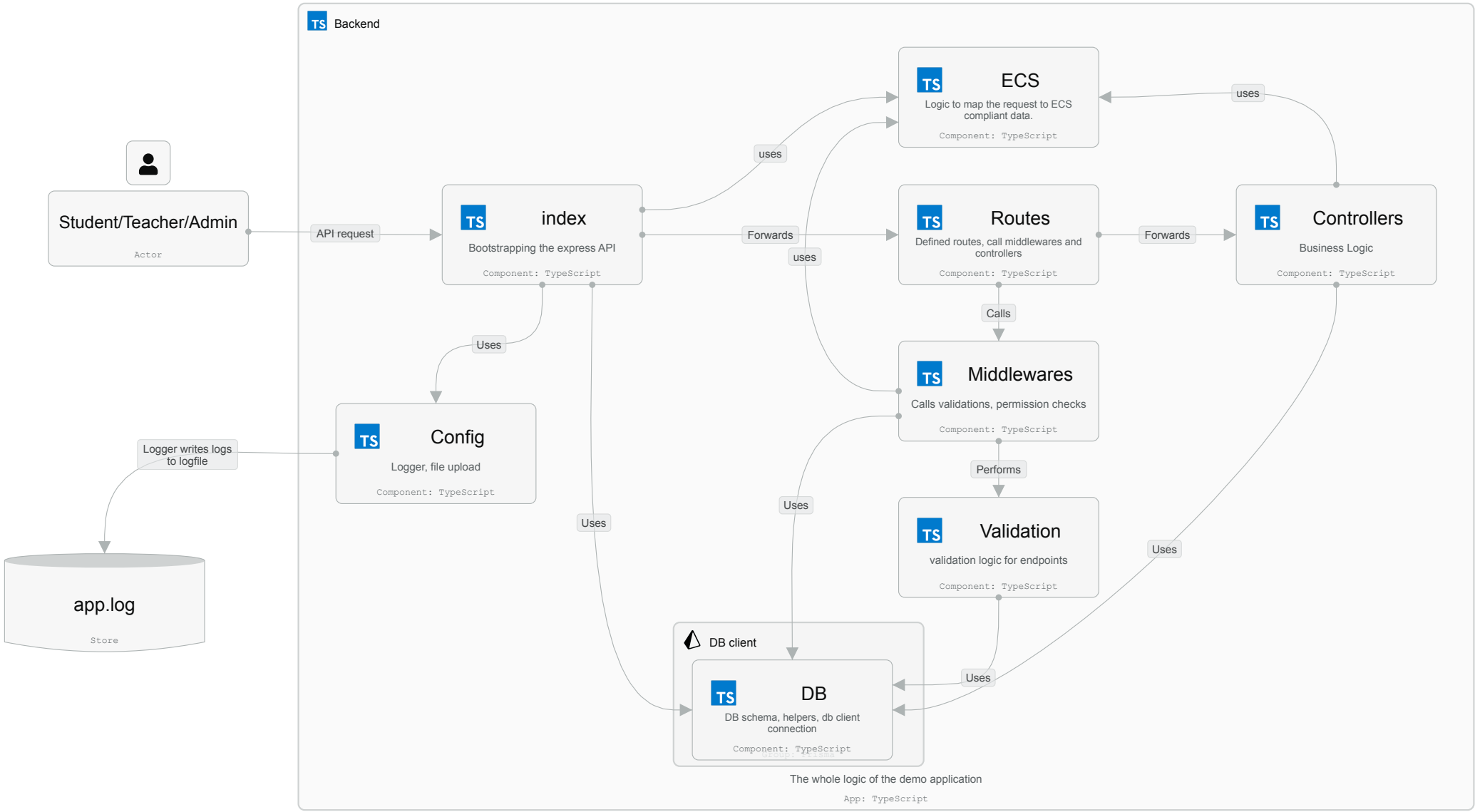


Figure 8: Student Portal - Component Diagram

Analog to the components of the `Generator` shown in Figure 7 , the components of the `Student Portal` have the same structure as the code in the repository. Their responsibilities are as described on the diagram.

2.3.3.4 ML Model

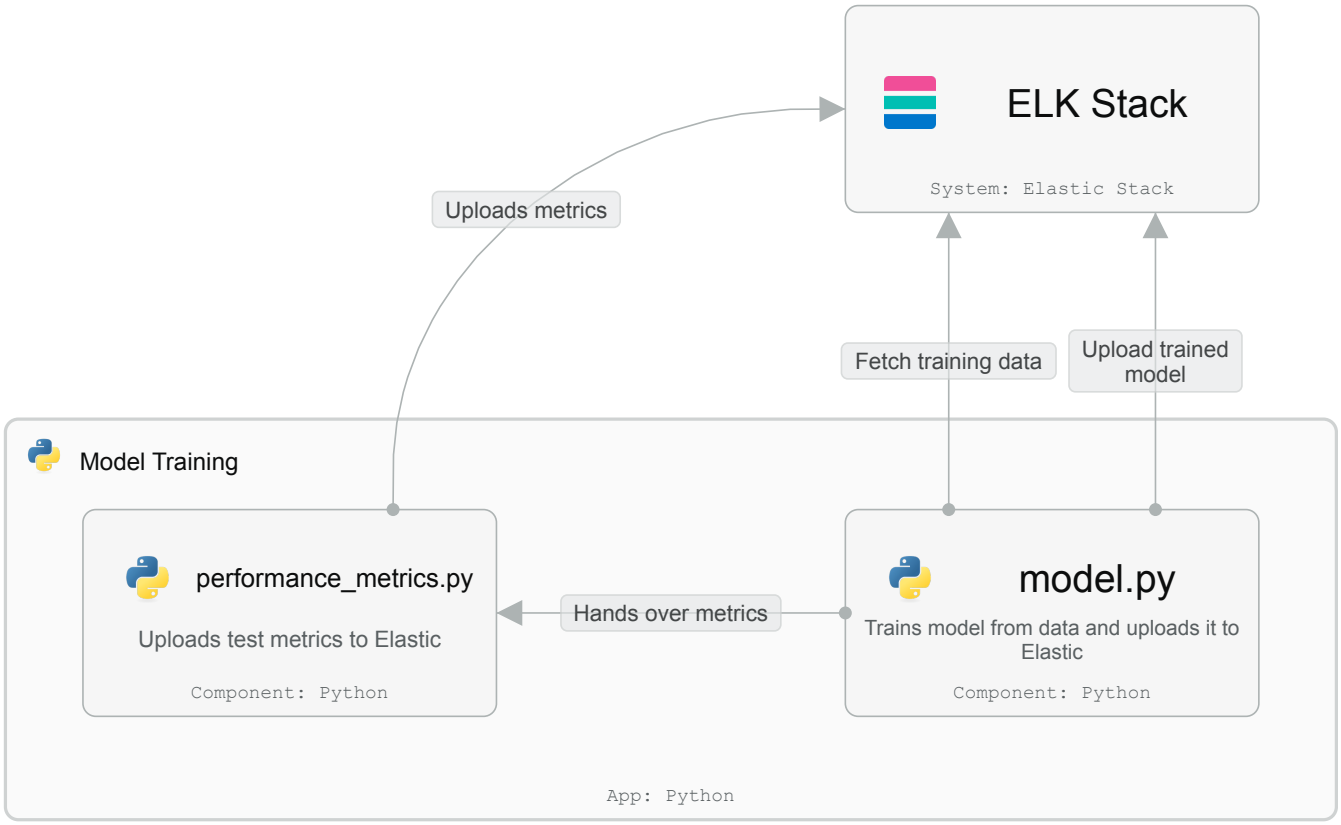


Figure 9: Custom ML Model Component Diagram

The ML model container is divided in two components. `model.py` pulls the data from Elastic, trains the model on the data and uploads the model to the Elastic instance. `performance_metrics.py` takes the test performance of the model, which was trained by `model.py` and uploads them to the same Elastic instance, just to a different index.

2.3.4 Code diagrams

Code diagrams provide the lowest level of detail, showing concrete code elements, such as classes, methods, and relationships within a component.

Since the complete source code will be handed-in with this documentation and is also available on the OST GitLab, the decision has been taken to not document on this level.

2.4 Demo Application

The application simulates a very simple student management platform that is build with TypeScript and Express.js as the web framework. To follow best practices as encouraged in [13] (page 6), all user inputs are validated. This is important to prevent invalid data from being logged and messing up the models training and evaluation. Postgres is used as a database to store the applications data. For the connection between the application and the database, Prisma is used as the ORM to make database access easier and quicker to setup. The decision to use that stack was based on the existing experience of the project team with those technologies and it allowing for a quick setup of a simple web application. Which enables spending more time on the rest of the testbed.

2.4.1 System Features

The system features describe the different actions the users can perform in the application. Every action performed by a user is logged and can then be used to detect anomalies in the users behavior.

2.4.1.1 Administrator Functionality

Administrators are responsible for the overarching management of the systems user base. Their primary task is to create and configure new user accounts. This includes registering both teachers and students and assigning the appropriate role-specific permissions. By handling user onboarding, administrators ensure that all actors can access the system with the correct authorization level.

2.4.1.2 Teacher Functionality

Teachers are provided with features that enable them to organize and manage their classes. They can create new classes by supplying essential information such as the course title and times. Furthermore, they are able to manage class enrollment by adding or removing students as needed.

In addition to administrative class control, teachers can upload instructional materials, including documents relevant to lectures, assignments, or supplementary learning resources. Uploaded documents can also be deleted if they become outdated or require replacement. To support academic assessment, teachers can record and update student grades directly within the system.

2.4.1.3 Student Functionality

Students are granted access to all information and resources necessary for participating in their classes. They can browse all available classes and view detailed information for the ones in which they are enrolled. The system allows students to download documents provided by teachers, such as lecture notes or assignments.

To support academic planning, students can consult their personal schedule. Additionally, the application offers access to the student's personal grades as entered and maintained by the respective teachers.

2.4.2 Database

The demo applications data is stored in a simple PostgreSQL database, this adds a bit of realism to the generated data, while keeping the setup overhead as low as possible.

Here is an overview of the DBs entities and their relations.

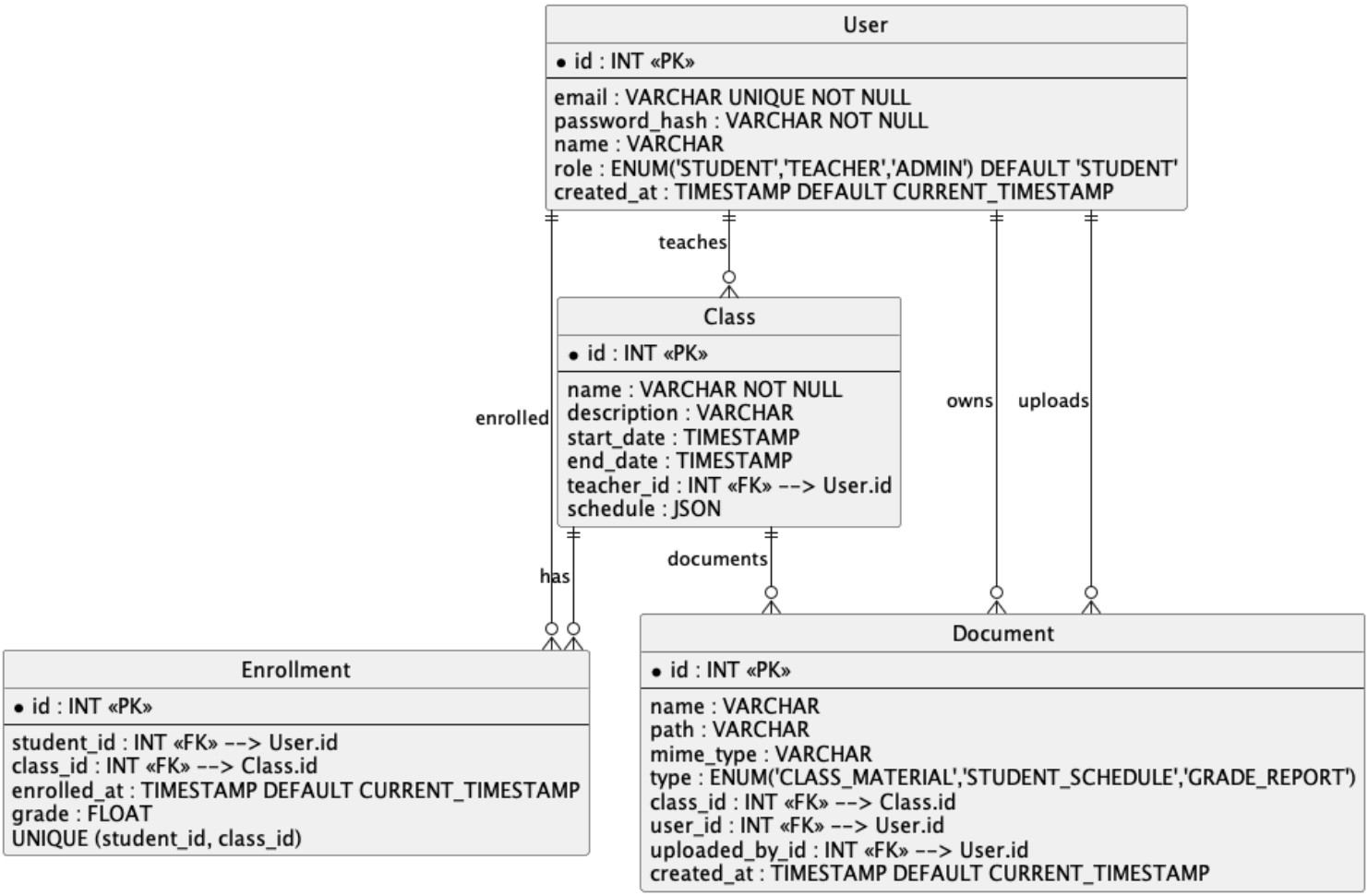


Figure 10: ERD of the demo application.

While designing the database it had to be decided whether to use UUIDs as primary keys for the tables or simple integers, that auto increment. Since it makes no difference application wise, it was decided to use integers as primary keys so an anomaly could be simulated with random access to ids that do not exist or the user has no permission to access. It's also the exact opposite of best practices described in [13] (page 6) mentioned earlier. This can be used to better simulate anomalies.

2.4.3 Interaction with ELK pipeline

To make the tracking of users easy (since we focus on user behavior) all endpoints require authentication via JWT token. A user has to login first in order to receive a token which has to be sent with every request. The token contains a sessionId which allows us to track a user over multiple requests that are made with the same token.

Every HTTP request that the application receives is appended to a log file on disk. These logs are in ECS (Elastic Common Schema) which is required for the default ML rules available in Elastic. The exploration of rule based anomaly detection is not in scope for this project, but you can find more information about the functionality in the official documentation . The logs are then read by Filebeat and sent to the data ingress pipeline.

2.4.3.1 ECS logs

Elastic provides predefined security rules that can be used to detect anomalies in the data. They are based on the Elastic Common Schema (ECS) which is a standardized way of structuring data in Elastic. To follow best practices the ECS format was adopted for the logs of the demo application.

Another option would have been to use OpenTelemetry for logging and tracing which can be mapped to ECS. However the project team has no experience with this technology, so it was decided to keep the setup as simple as possible.

2.4.3.2 Headers

To be able to log all the necessary data in ECS format, http headers are used to fill in values to enrich the data. The following headers are used:

Header	Description
`user-agent`	The user agent of the client making the request. This is nothing new, but since we make the calls via the traffic generator, we can simulate different devices by setting this header. Browsers usually set this header automatically.
`x-demo-no-log`	If this header is set to true, the request will not be logged. This is useful for api requests that the traffic generator has to make in order to work and dispatch other requests.
`x-demo-ip`	The ip address of the client making the request. This is used to simulate different clients with different ip addresses.
`x-demo-labels`	Custom labels to enrich the data, represented in stringified json structure. This is used to attach sessionIds of the clients. We also set a label if an anomaly is simulated in the request. This is required to train and evaluate models with supervised learning.
`x-demo-timestamp`	The timestamp of the request. Our traffic generator can simulate time and speed it up to generate data over a longer period of time in a short amount of time. This header is used to set the correct timestamp for the request.

Tab. 5: Application Headers for Log Enrichment

3 Data Generation

In order to evaluate the models, a purpose built dataset was created. Multiple attempts were made to find an appropriate, already existing dataset, but while there are a couple of existing options for low level network traffic (OSI layers 2 and 3), there is no dataset available which depict a users flow through an application. This makes sense because every application is different and it is almost impossible to create a universally applicable dataset.

The data generator is responsible for creating realistic data that can be used to train and evaluate the anomaly detection models. It requires some configuration. The most important one being the start and end time for the data generation in combination with the duration in seconds, in which this time should be simulated. This results in a scale factor that is used to speed up the time simulation. For example a scale factor of 60 means that 1 real second equals 1 simulated minute. There is a time helper that contains all of the logic to simulate it. One has to be careful though to not set the scale factor too high, because otherwise the generator and or the backend might not be able to keep up with the generated requests. This would lead to unrealistic timestamps in the logs because the next user actions get dispatched later than they would without such a high scale factor.

The settings shown in Listing 1 were chosen for the data generation to not overload the host machines. Delayed requests were observed on the host systems (MacBook Pro 16”, M1 Max) when the scale factor was >1000 with the given numbers of workers. This config resulted in 57’370 request simulated over a period of 33 days in around 2 hours of real time.

```
# ====== DATE CONFIG ======
START_TIME=2025-11-02T00:00:00Z
END_TIME=2025-12-05T21:59:59Z
RUN_DURATION_SECONDS=7000

# ====== TRAFFIC CONFIG ======
CLASS_WORKERS_COUNT=30
STUDENT_WORKERS_COUNT=500

# ====== ANOMALY CONFIG ======
ANOMALY_403_COUNT=60
ANOMALY_MANY_FILES_COUNT=45
ANOMALY_GEO_COUNT=30
```

Listing 1: “Data Generation Configuration”

3.1 Flows

Flows define how a user behaves while using the application. The application simulates one flow for the students and how they interact with the application and one flow for the teachers. These two flows are responsible for most of the data that is generated.

In addition to these normal flows, three anomaly flows were implemented that simulate specific anomalous behavior. These flows are responsible for generating around 5% to 10% of the data to have enough anomalies to be able to train and evaluate the models properly. New flows can be added easily if needed.

3.1.1 Student

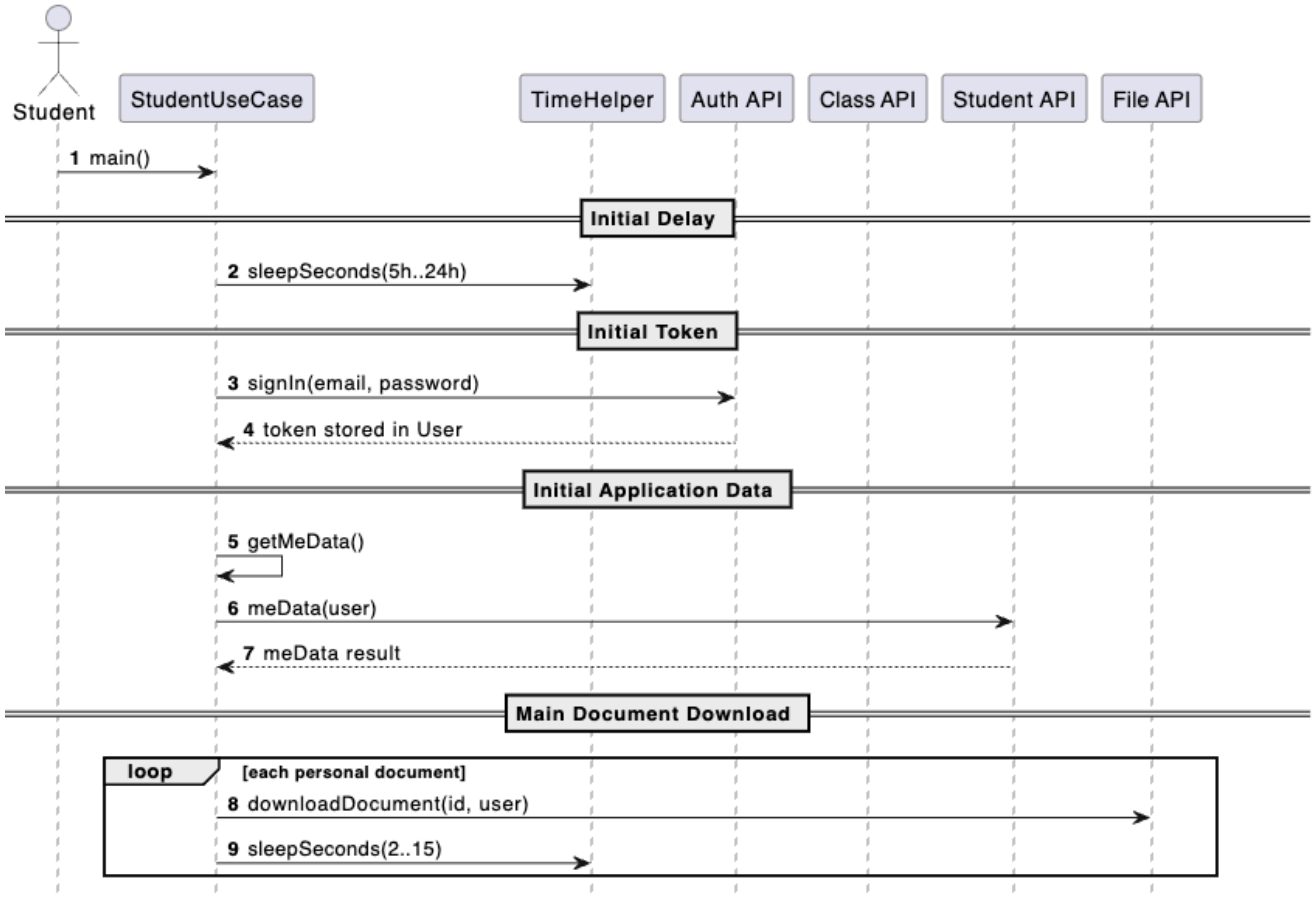


Figure 11: Student Flow - 1

The student flow in Figure 11 is by far the most complex and contains all tasks a student can perform. It also depends on the amount of classes the student is enrolled in. The more classes the student has, the more tasks will be performed. The user flows are started once when the application starts and then loop until the application shuts down.

It starts by the initial setup, containing signing in and fetching personal data and documents as seen in Figure 11 .

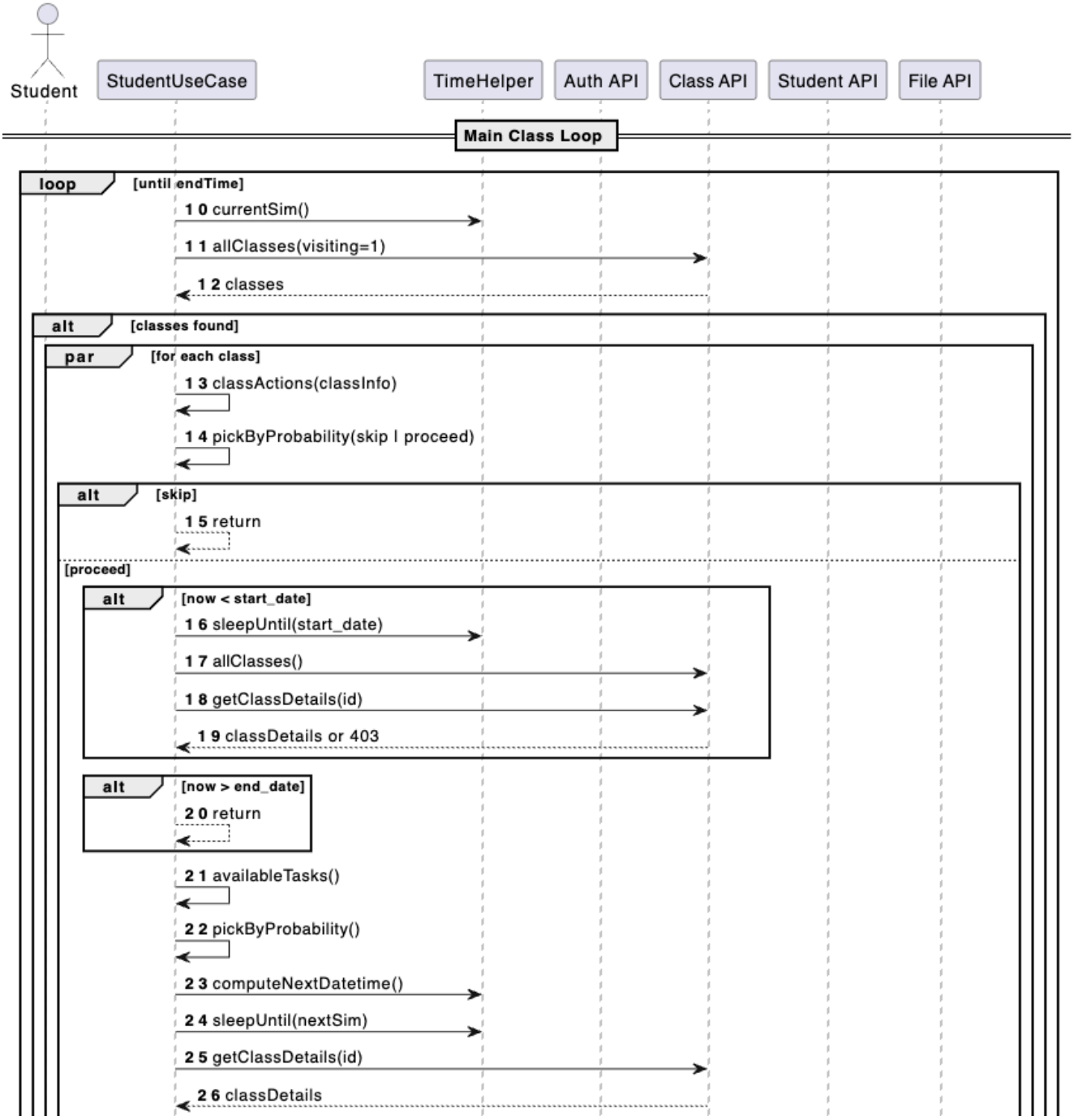


Figure 12: Student Flow - 2

It continues in Figure 12 with fetching all classes the student is enrolled in and selects the task to perform next.

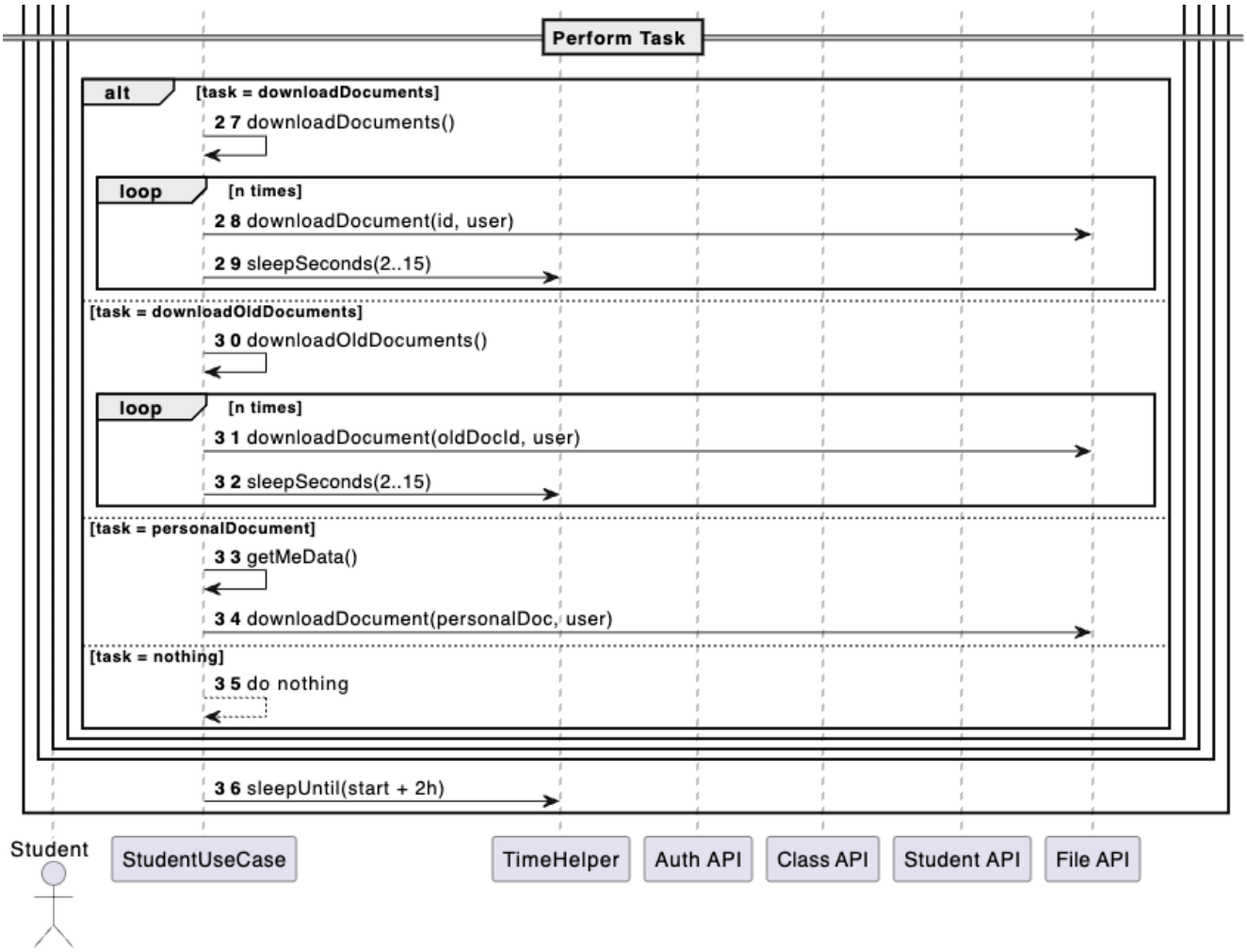


Figure 13: Student Flow - 3

The student performs the task in Figure 13 for each class he/she is enrolled in.

3.1.2 Professor

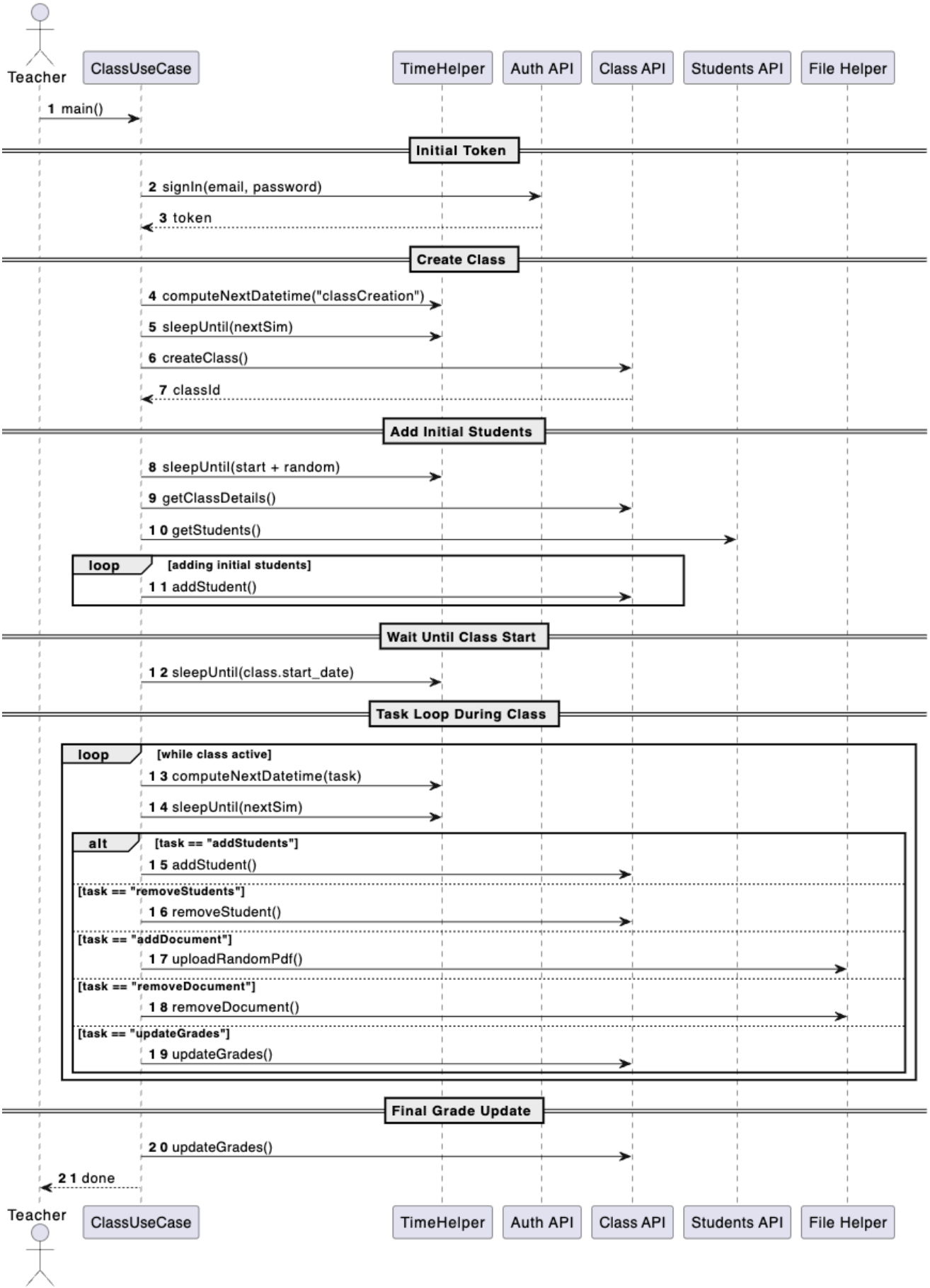


Figure 14: Teacher Flow

Professors perform their tasks over the lifetime of a class. It always starts with creating it and ends with the final grade update. The tasks in between must happen in specific time slots, but can be in any order and quantity within the lifetime of the class.

3.2 Anomalies

The anomalies are designed to simulate specific types of anomalous behavior that the model needs to be able to detect. Three different anomaly flows that simulate different types of anomalies have been implemented. The anomalies are inspired by the following source [16] (page 4) and its “User Behavior Analytics”chapter, that describes different anomalies typical in user behavior analytics context and their common detection patterns.

3.2.1 Geo changes

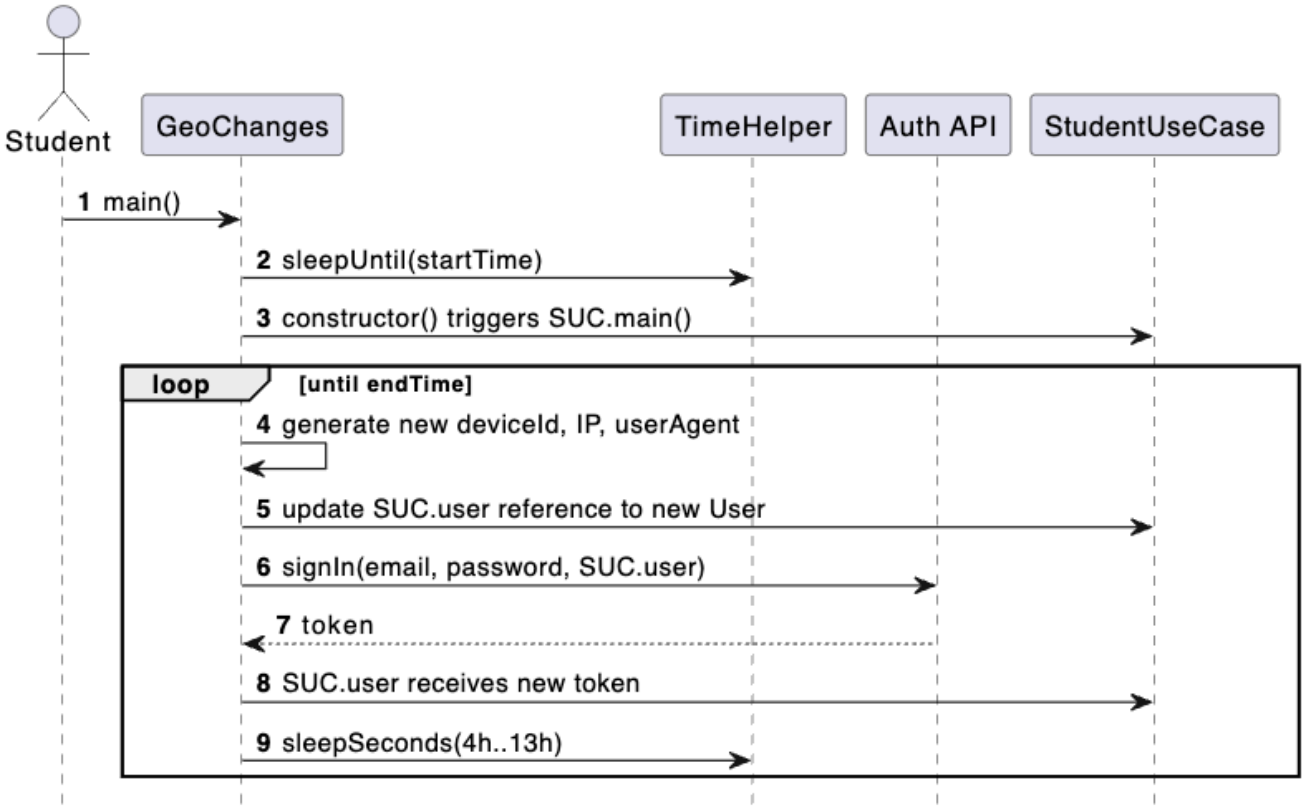


Figure 15: Geo Location Change Anomaly

The geo location change anomaly simulates a user that suddenly accesses the application from a different location and using a different device. It is exactly the same flow as the normal student flow but with the addition that the user signs in using a random device and ip address every 4 to 13 hours. Figure 15 only shows the important part of the flow, the normal student flow can be found in Section 3.1.1 .

3.2.2 Many Files

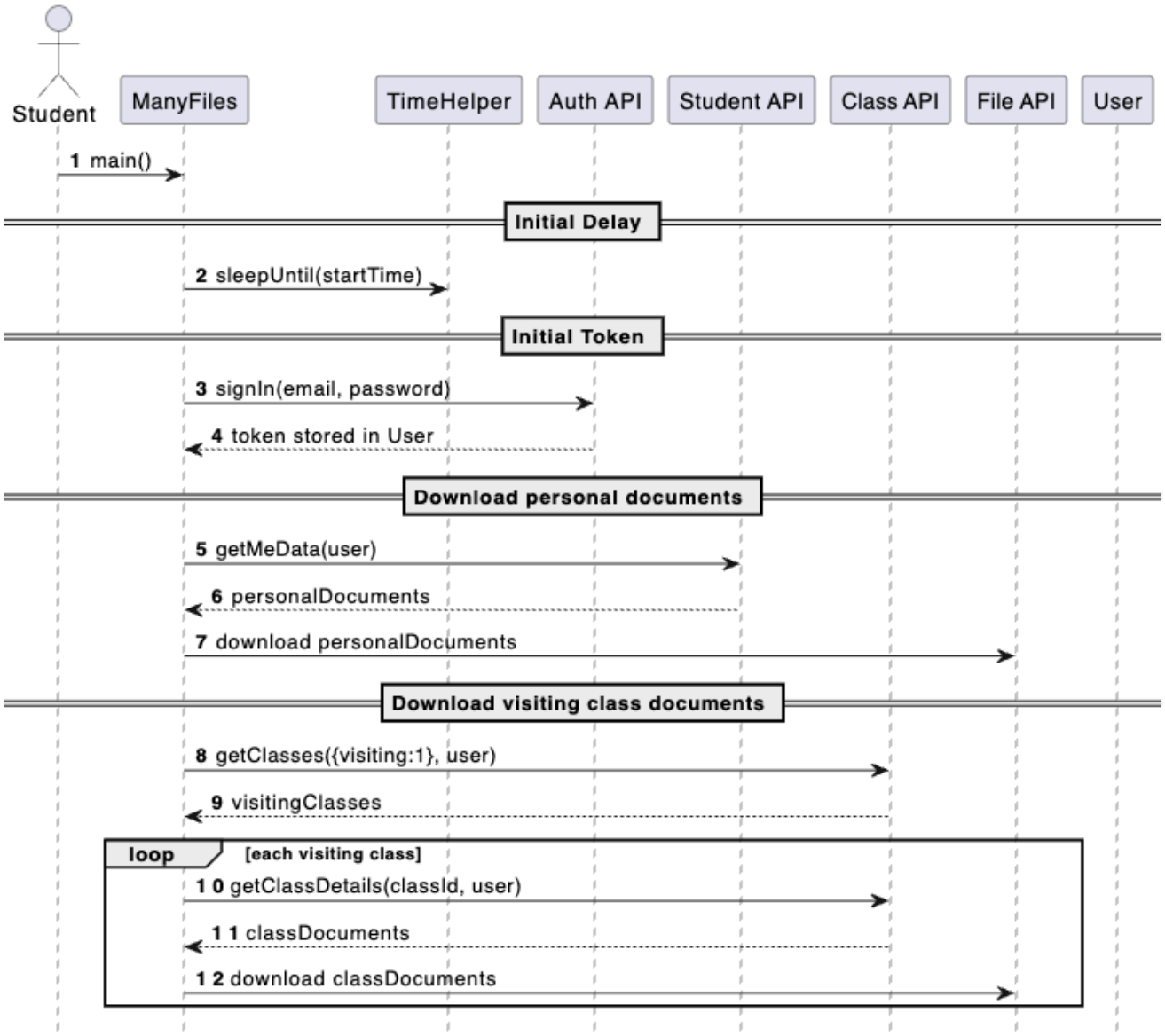


Figure 16: Many Files Anomaly

The anomaly shown in Figure 16 runs only once. The flow start at its configured starting time and then signs in the student, who downloads all available files. First the personal documents (schedule and grading sheet) and then all documents from all classes the student is enrolled in.

3.2.3 Student 403 Forbidden

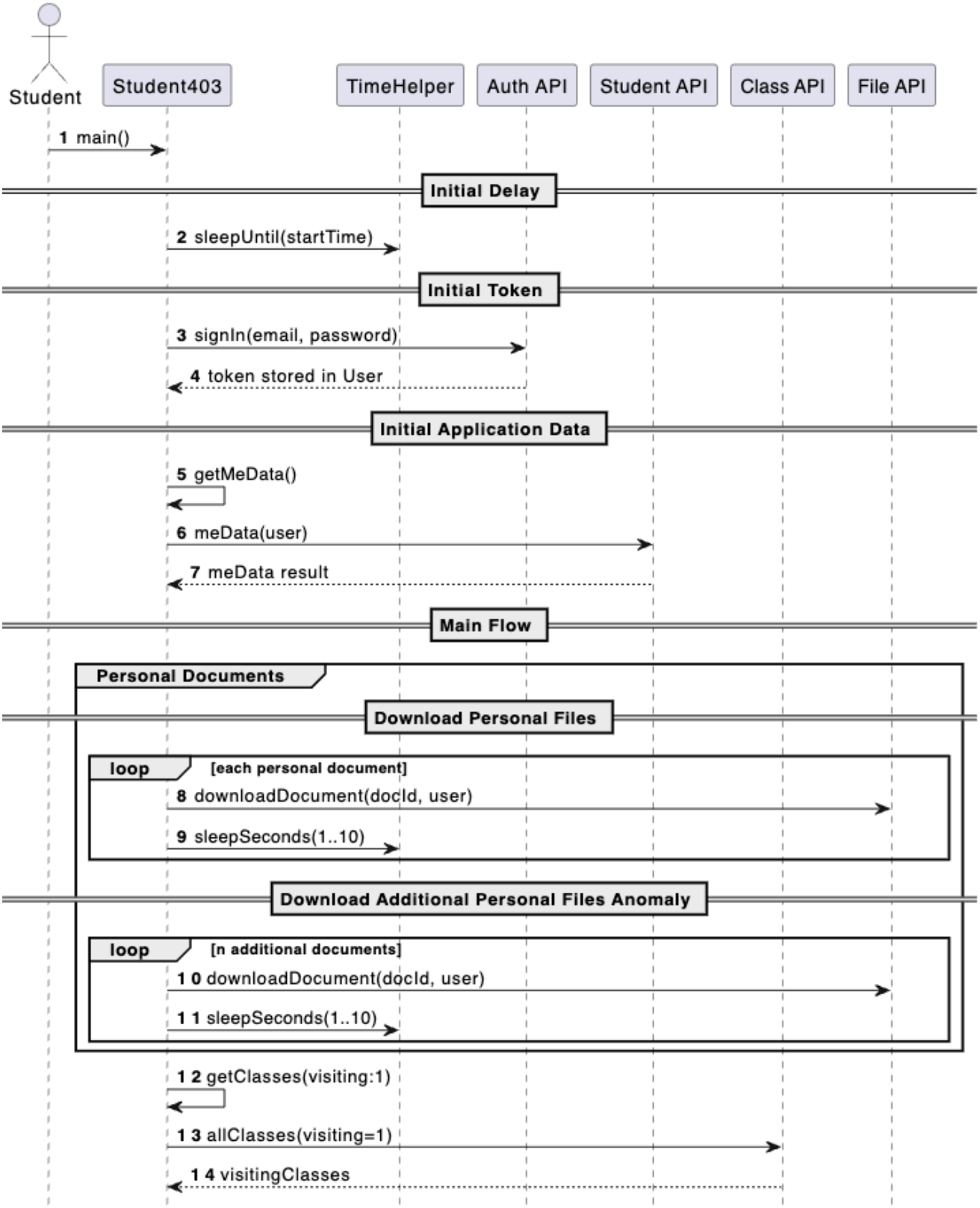


Figure 17: Student 403 Forbidden Anomaly - 1

Figure 17 shows the first part of the random access attempt flow. Instead of just requesting random resources, the flow starts regularly by signing in the student and getting his/her personal data and documents. After that, the student tries to download some more documents using random ids that do not belong to him/her.

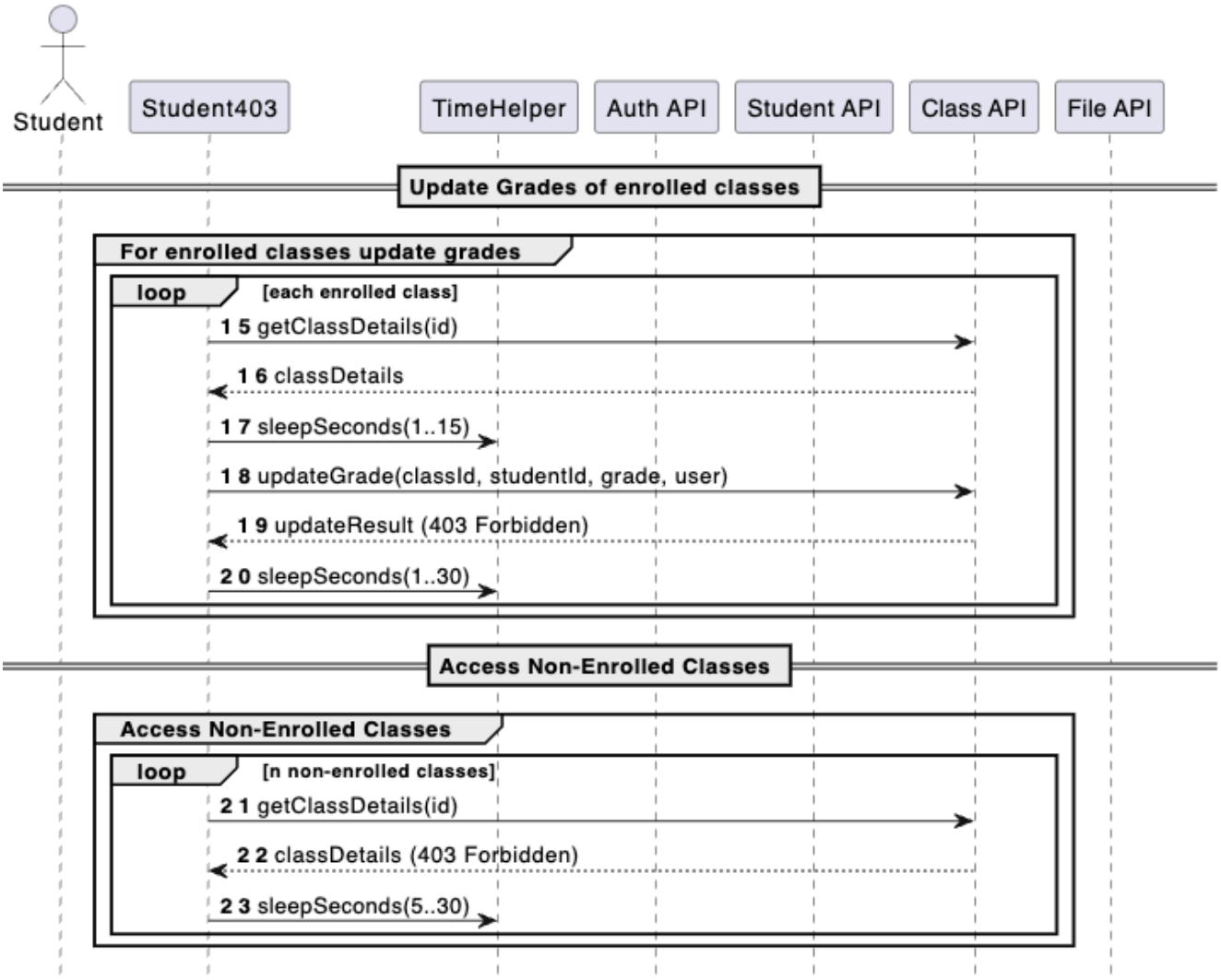


Figure 18: Student 403 Forbidden Anomaly - 2

Figure 18 continues with the student trying to update grades of classes he/she is enrolled in which is only possible for the respective teacher and will therefore also result in 403 Forbidden responses from the backend. Lastly, the student tries to access some classes he/she is not enrolled in which results in some more 403 responses.

3.3 Difficulties with data generation

Before the start of this thesis, the data engineering part of the project was already known to posing a great challenge. During the studies the project team has heard multiple times, across different modules, that around 80% of the time and budget in a machine learning project is spent on data preparation. This article [3] supports this claim.

As the time frame for this projects is tight, there is not enough time to first analyze every dataset in detail to find the best fitting one, implement the ingestion, labelling and feature engineering. With the chosen approach the need of analyzing the datasets and implementing dataset specific solutions are eliminated. In addition, the kind of simulated anomalies and their properties can be fully managed by the project team.

The problems described above and the resulting approach are the main deciding factors for the final designs of the regular and anomalous flows described in Section 3.1 and Section 3.2 , respectively. Some inspiration was gathered from an article [16] that describes different types of anomalies in web application logs.

There is a dockerized version of the generator, which is technically limited by thread handling and timing behavior.

4 Data Processing and Ingestion

After the data is written to the `app.log` file by the demo application, it gets ingested into Elasticsearch for further processing and analysis. Multiple components are involved in this process, from ingesting the data to processing it into a different structure for analysis.

At a high level, the system processes application logs in two phases. First, log events are collected from the log file and shipped through the ingestion pipeline (Filebeat, Section 4.1.1 and Logstash, Section 4.1.2) into Elasticsearch. Second, the ML models consume the data for training.

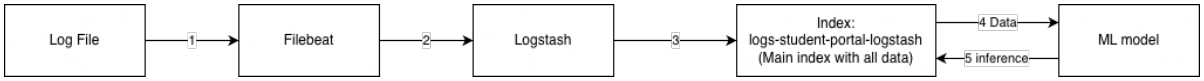


Figure 19: Data Flow Concept

Figure 19 shows the conceptual data flow through the whole application.

The detailed data flow concept specifically for ML is shown in Figure 22 .

4.1 Ingestion Pipeline

The ingestion pipeline is responsible for getting the data from the source (the log file) into Elasticsearch. It consists of multiple components that work together to achieve this goal.

4.1.1 Filebeat setup

Filebeat is running on the same client/ host that is producing the data. In this project Filebeat is used to read the log file and send the data to Logstash for further processing. Filebeat is configured to use the “filestream” input type which takes an array of file paths to read log files from, as shown in Listing 2 .

```
filebeat.inputs:
  - type: filestream
    id: student-portal-logs
    paths:
      - /var/log/student-portal/*.log
```

Listing 2: Filebeat: Log File Configuration

Filebeat keeps track of which log lines it has already read and sent, ensuring data integrity. So even after a reboot, it knows the offset from where to continue. Filebeat guarantees that the data is delivered at least once by storing its state in a registry file. It achieves this by sending the events to the configure output until it receives an acknowledgment that the data has been received successfully.

For performance reasons, Filebeat batches multiple events together before sending them to the output. This reduces the overhead of network communication and improves throughput.

The only way to secure the communication between Filebeat and Logstash is via TLS. There is no support for username/ password or API keys according to the official documentation .

Filebeat and Logstash are running on the same host with our given configuration and do not have TLS setup for secure communication and data will be sent directly to logstash. The output configuration for Logstash is shown in Listing 3 .

```
output.logstash:
hosts: ["logstash:5044"]
```

Listing 3: Filebeat: Output Configuration

4.1.2 Logstash processing

Logstash acts as a processing pipeline that sits between Beats and the Elasticsearch storage layer. It has three main functions: input, filter, and output:

Input: Defines where the data will be received from like in Listing 4 .

```
input {
  beats {
    port => 5044
  }
}
```

Listing 4: Logstash: Input Configuration

Filter: A series of processing steps that parse, clean, and fully transform the incoming data.

Output: Specifies where the processed data should be sent to as visible in Listing 5 . This will in most cases be an Elasticsearch index or data stream. One can also specify pipelines that should be applied to the incoming data for further processing in Elasticsearch.

```
output {
  elasticsearch {
    hosts => [ "${ELASTIC_HOST}" ]
    api_key => "${ELASTIC_API_KEY}"
    pipeline => "geoip-pipeline"
    ssl_verification_mode => "none"
    ssl_enabled => "true"
    data_stream => "true"
    data_stream_type => "logs"
    data_stream_dataset => "student-portal"
    data_stream_namespace => "logstash"
    ecs_compatibility => "v8"
  }
}
```

Listing 5: Logstash: Output Configuration

Even though Logstash has only these three main functions, it supports conditions which allows to create complex pipelines based on the content of the data.

While Filebeat must run on the same system that is producing the data (unless this system already sends it to somewhere), Logstash could be run on a separate server collecting data from multiple sources. So only one Logstash instance is needed for multiple Filebeat instances reducing resource consumption. But like mentioned in Section 4.1.1 , both will be run on the same host.

4.1.3 Data Organization

Within Elasticsearch, data is stored as JSON documents. Where these documents end up is determined by either an index or a data stream.

Indices are the basic storage abstraction. An index defines its structure through mappings that specify field names and types.

Data streams provide a higher level abstraction for time series data and internally organize data into multiple indices. They are especially useful for continuously generated logs, metrics, or events, because data streams automatically manage index creation. This simplifies ingestion and supports retention policies while keeping search efficient as data grows.

In summary, data streams are advanced indices with additional automation and optimizations for time based data.

In the setup of this project, Logstash writes application logs into an Elastic data stream (configured via `data_stream => true` in Listing 5). This fits the log use case well, because new log events are appended continuously over time.

4.2 Transforms

Each row represents a single user action, but the objective is not to analyze individual actions. Rather, the goal is to determine whether an entire session is anomalous. This is where Transforms are helpful. Transforms are components in Elastic where a user can define aggregation functions on different fields or group rows based on a fields value in order to pivot the data to

represent a new entity [17] . This new featureset, then gets saved in a new index (explained in Section 4.1.3), which is used by the different machine learning models. The two upcoming chapters show the final configurations of the Transforms but do not include step by step instructions, those can be found in Section 6.1.3.1 .

4.2.1 Labelled Transforms

First a labelled Transform was created. It groups all datasets by their session ID and performs aggregations on relevant session-level metrics. It also includes a label that is used as the ground truth for testing and validation after the models training:

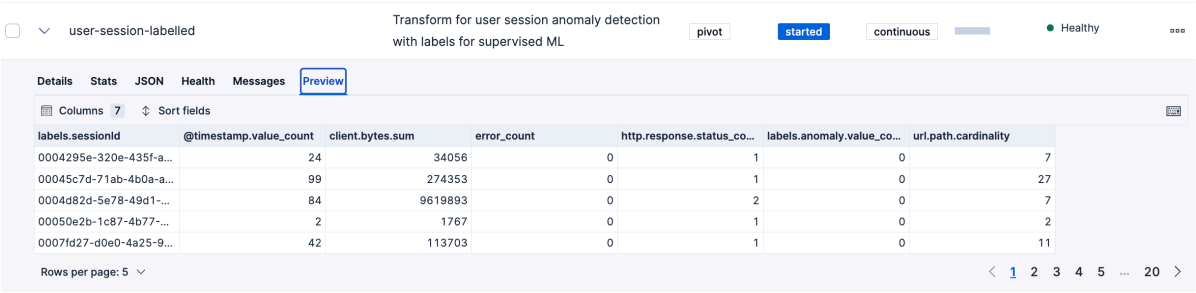


Figure 20: Labelled Transform Features

A quick explanation for every selected feature:

Field	Explanation
labels.sessionId	The unique identifier for a session
@timestamp.value_count	The number of user actions during the session
client.bytes.sum	The sum of bytes sent by the client
error_count	The number of errors the server responded with during the session
http.response.status_code.cardinality	The number of different http status codes sent to the client during the session
labels.anomaly.value_count	The number of anomalies in the session
url.path.cardinality	The number of different url paths requested by the client during the session

Tab. 6: Labelled Transform Fields

The number of anomalies in the session is always equal to the number of actions in a session. This is based on the decision taken in the concept phase to classify a whole session as “anomalous” or “not anomalous”, and not differentiate between single requests.

The selected features are the result of the project team brainstorming for meaningful features and then consulting a LLM for the reasonability of the ideas. There are multiple mathematical and algorithmic approaches for feature selection, but this is out of scope for this work and would use up the time budget disproportionately to the benefits. The optimal selection of features can be explored in a future project, where the focus would be to get the best performance out of a ML model. As a proof of concept, for a ML model testbed this selection of features delivers good enough results.

Before, one row in the generated dataset was a singular user action. After applying the Transform the entity is changed to a session, which includes multiple actions.

4.2.2 Unlabelled Transforms

The unlabelled Transform includes the same features as the labelled one, but is missing the label.

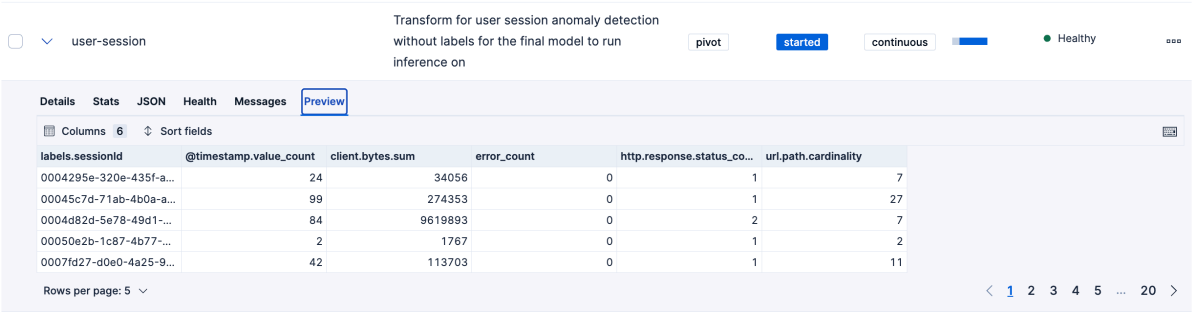


Figure 21: Labelled Transform

This Transform is running in continuous mode, which means that new incoming data will be taken in by this Transform and put into its new index.

5 Machine Learning

The machine learning part is one of the core elements of this project. As defined in Section 1.4 , two main ML components are explored in this work. The first being the out of the box Elastic ML capabilities, the second being a self trained model. Elastics model is used as a base line to determine how close the custom model can get to its performance and maybe find some gaps that could potentially be explored in future projects.

5.1 Data Frame Analytics

Data Frame Analytics are the built in functionalities to run classification models on indices in Elastic. This feature is used in order for comparison to the custom model.

Elastics models are closed source, so not much is known about the training or the parameter tuning process. The step by step configuration can be found in Section 6.1.3.2

5.2 Custom ML model

There are use cases where running the out of the box Data Frame Analytics in Elastic is not enough and one needs to implement a custom training loop or make other changes to the model. For this case, it is possible to get data from Elastic, train a model and import it in Elastic for potential use in inference pipelines. The implementation can be split up in three steps. The following chapters explain each step and show the final configuration. Analog to Section 4.2 , the detailed step by step instructions to setup a custom model can be found in Section 6.1.3.3 .

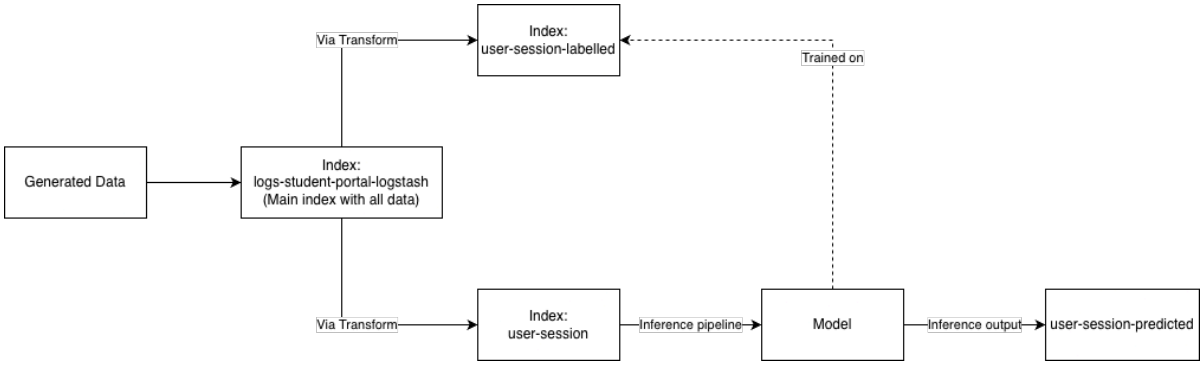


Figure 22: ML Data Flow Concept

5.2.1 Training the model on our data

Before defining the model itself data needs to be get from the Elastic index. Elastic provides a python client which can be used to access the Elastic instance and run different actions.

One of those actions is to pull data from a defined index

```
es = Elasticsearch(ES_HOST, api_key=ES_API_KEY)
ES_INDEX = "user-session-labelled"

for hit in helpers.scan(es, index=ES_INDEX):
  flat = flatten(hit["_source"])
  rows.append(flat)
```

Listing 6: Pull data from Elastic index

`helpers` is an imported package from the elastic client and `.scan()` runs the search. The user-session-labelled index is used for the training of our supervised model. `.scan()` is safer on larger datasets than regular queries, because it uses scrolling to avoid timeouts, as results returned by the regular search function time out after a period of time (default: 1 minute). The return is a nested JSON, so `flatten` is used to put the returned values into the one-dimensional `rows` array.

After the data was pulled a data frame is created and split it into a label vector y and the feature matrix X.

```
LABEL_COUNT_FIELD = "labels_anomaly_value_count"
feature_cols = [
    EVENT_COUNT_FIELD,
    BYTES_SUM_FIELD,
    STATUS_CARDINALITY_FIELD,
    URL_CARDINALITY_FIELD,
    ERROR_COUNT_FIELD,
]

df = pd.DataFrame(rows)
y = (df[LABEL_COUNT_FIELD].fillna(0).astype(float) > 0).astype(int).to_numpy()
X_df = df[feature_cols].copy()
```

Listing 7: Create label vector and feature matrix

The line beginning with `y = ...` is the line that turns the data frame column named "label_anomaly_value_count" (our label) into its own vector. The next line copies every feature column into another data frame. This is done as to not accidentally have the label remain in the feature matrix and the model train on it. This is referred to as "Data Leakage" in machine learning.

The next step is the training of the models.

```
X_train, X_test, y_train, y_test = train_test_split(
  X,
  y,
  train_size=0.7,
  shuffle=True,
  stratify=y,
  random_state=42,
)

# Train
clf = RandomForestClassifier(
  n_estimators=200,
  class_weight"balanced",
  random_state=42,
  n_jobs=-1,
)

clf.fit(X_train, y_train)
```

Listing 8: Training the model

The used sklearn package is very helpful in this step as it has two high level functions `train_test_split()` and `fit()` which solve the whole data splitting into training and testing sets and training the model on the training data automatically. While this is not best practice when working with machine learning, the model itself is not a core part of the project, so it has been decided to take this shortcut in order to save time.

5.2.2 Importing the model in Elastic

Importing the model into Elastic happens with a code snippet. The procedure to import the model is documented in Section 6.1.3.3 .

5.2.3 Running the model

The custom model can be executed inside a pipeline in Elastic. Pipelines run on an index and can be run in two ways:

1. Once on a whole index, and output new data into another index, this process is called "reindexing"
2. The pipeline runs consistently, every time a new document is uploaded to the index

5.3 Model Evaluation

Two different models are analyzed in this testbed, the built in DFA Job and the custom ML model. Both need a different approach in the validation phase.

5.3.1 DFA Evaluation

After the DFA Job has finished its classification, an evaluation page is created, that can be found under “Machine Learning” > “Results Explorer”. This simplifies evaluating the built in functionality, by not having to create a dedicated dashboard for it.

When viewing the evaluation page under Machine Learning > Results Explorer, a page opens with 5 collapsible sections.

The second section is the key section. It displays a confusion matrix and an ROC curve

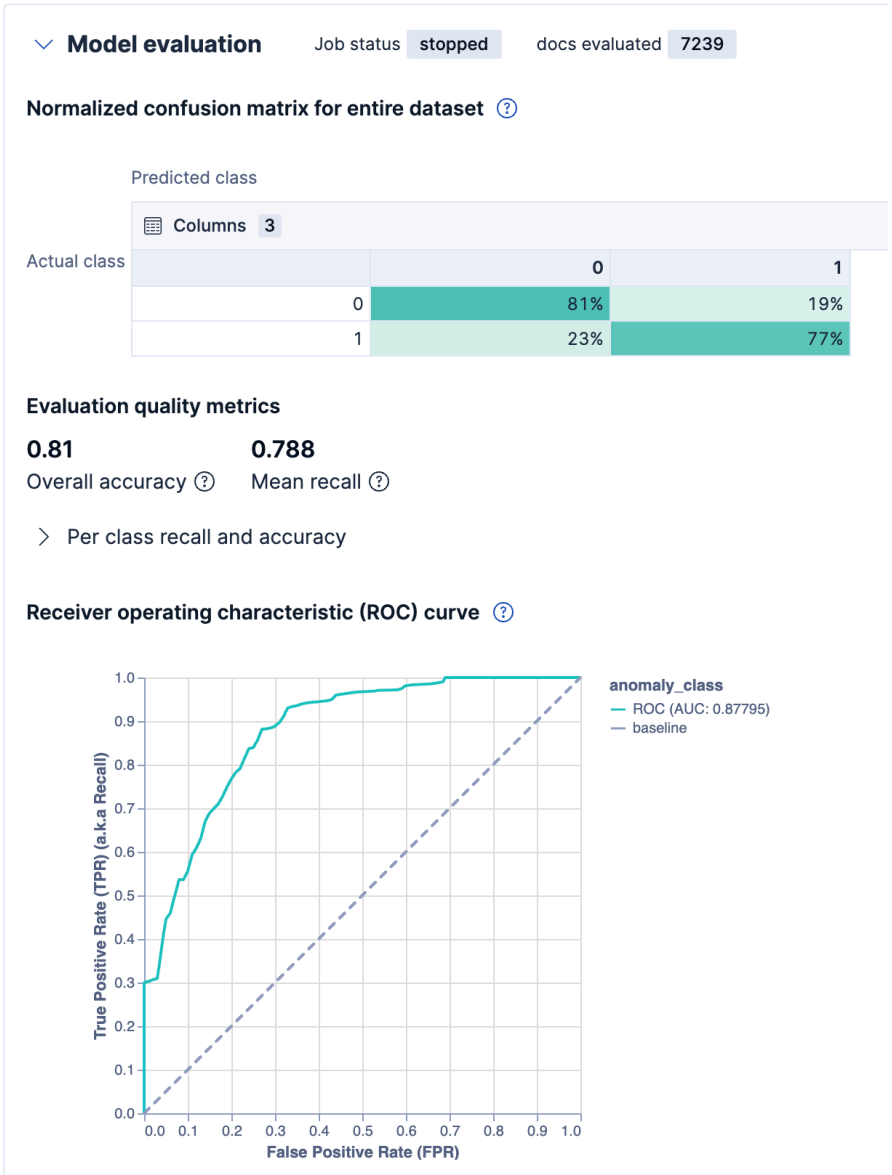


Figure 23: DFA Results Explorer - Model Evaluation

The third section shows the importance of each feature in a bar chart. The model calculates the importance of each feature for every datapoint, this chart is the result of averaging every features importance and displaying the averages in a single graph. This helps with feature engineering and selection.

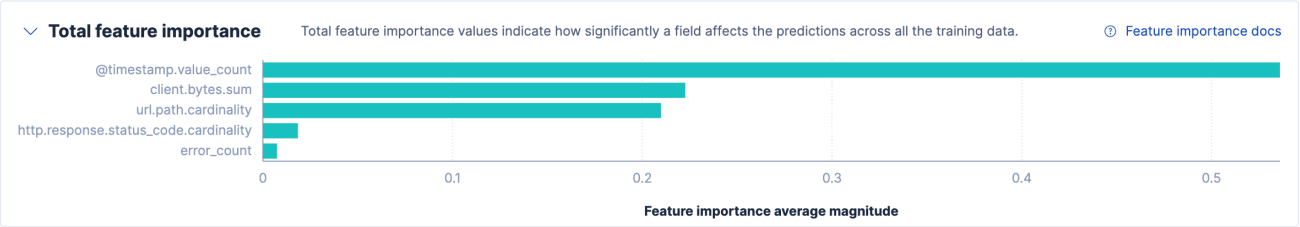


Figure 24: DFA Results Explorer - Feature Importance

The fourth section is the scatterplot matrix. In the “Fields” input the user can configure what features he wants to see the graphical relation of.

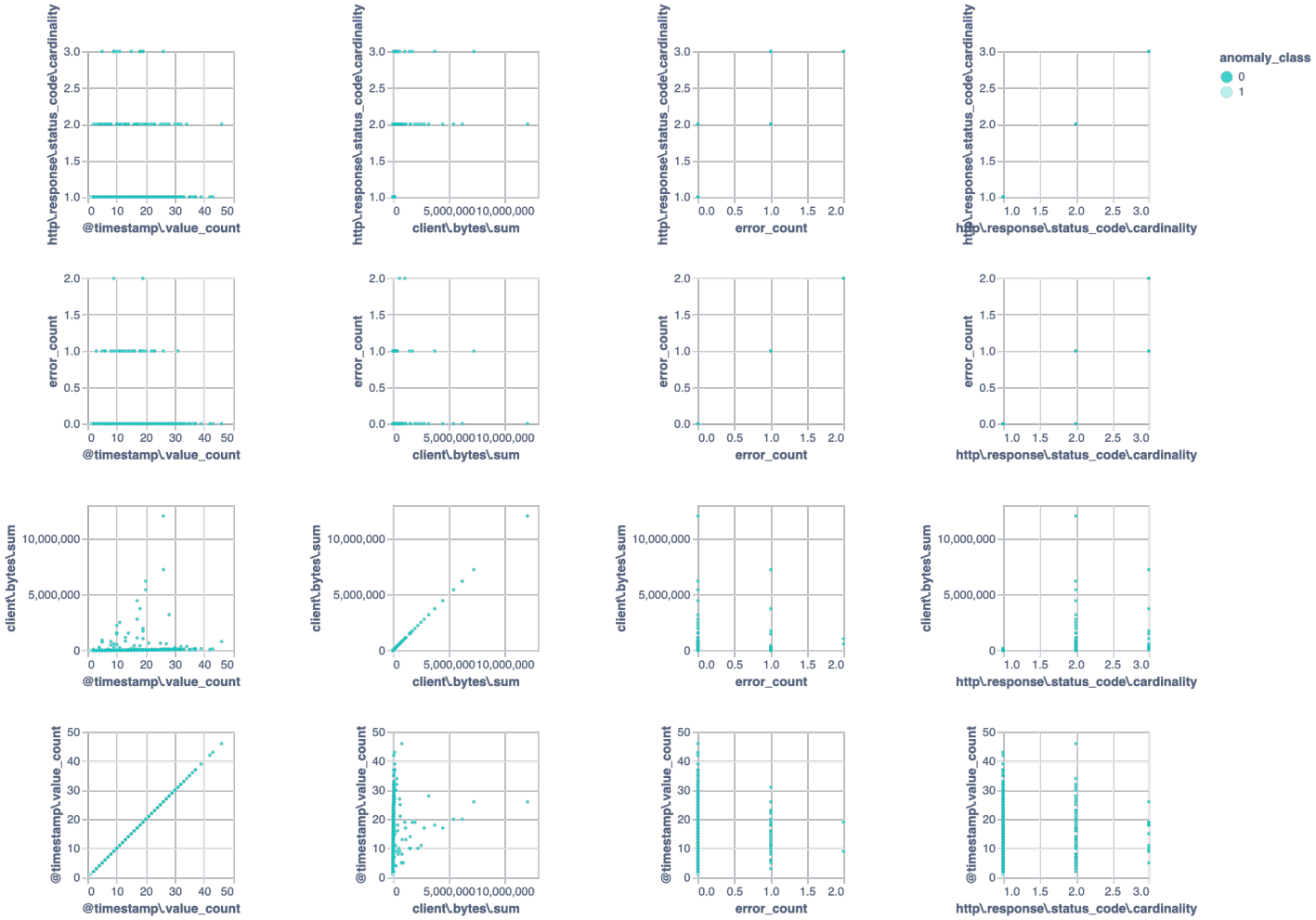
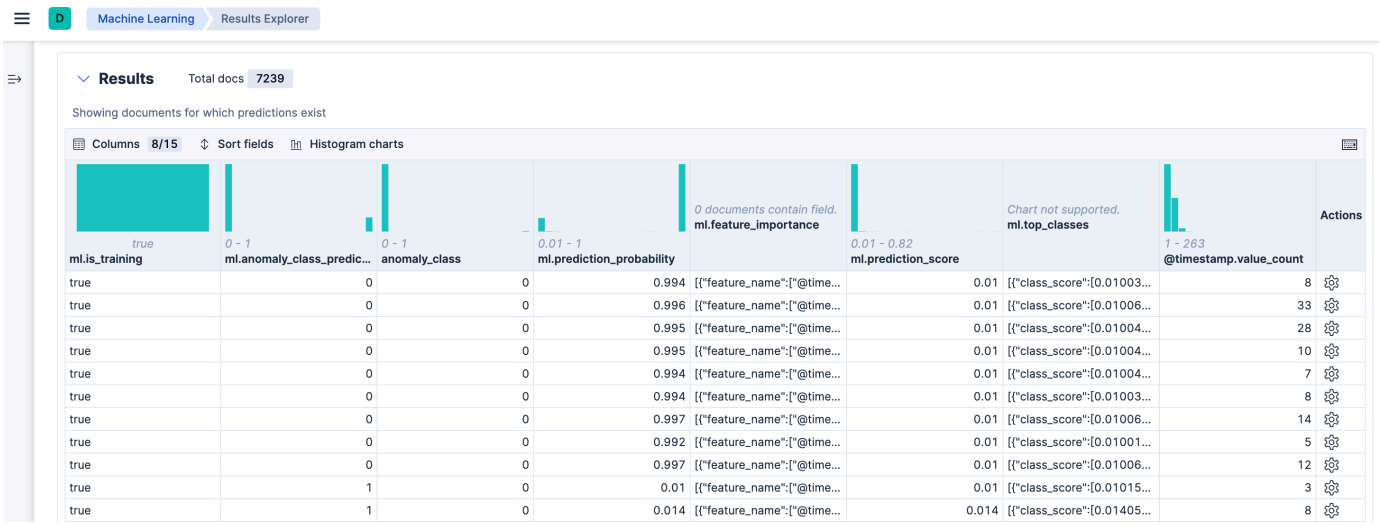


Figure 25: DFA Results Explorer - Scatterplot Matrix

Not every feature combination in this matrix will make sense, but it is helpful in a cherry-pick manner where individual plots can be selected, based on relevance.

Section five displays a preview of the dataset including the models predictions in tabular form. All new fields, added by the model are prefixed with “ml.”



ml.is_training	ml.anomaly_class_predic...	anomaly_class	ml.prediction_probability	ml.feature_importance	ml.prediction_score	ml.top_classes	@timestamp.value_count
true	0	0	0.994	[{"feature_name":["@time...	0.01	[{"class_score":[0.01003...	8
true	0	0	0.996	[{"feature_name":["@time...	0.01	[{"class_score":[0.01006...	33
true	0	0	0.995	[{"feature_name":["@time...	0.01	[{"class_score":[0.01004...	28
true	0	0	0.995	[{"feature_name":["@time...	0.01	[{"class_score":[0.01004...	10
true	0	0	0.994	[{"feature_name":["@time...	0.01	[{"class_score":[0.01004...	7
true	0	0	0.994	[{"feature_name":["@time...	0.01	[{"class_score":[0.01003...	8
true	0	0	0.997	[{"feature_name":["@time...	0.01	[{"class_score":[0.01006...	14
true	0	0	0.992	[{"feature_name":["@time...	0.01	[{"class_score":[0.01001...	5
true	0	0	0.997	[{"feature_name":["@time...	0.01	[{"class_score":[0.01006...	12
true	1	0	0.01	[{"feature_name":["@time...	0.01	[{"class_score":[0.01015...	3
true	1	0	0.014	[{"feature_name":["@time...	0.014	[{"class_score":[0.01405...	8

Figure 26: DFA Results Explorer - Results

A quick overview over the fields:

- “ml.is_training”, is true when the datapoint was used for training
- **“ml.anomaly_class_prediction”**, the prediction of the model whether this datapoint is an anomaly or not
- “anomaly_class”, the truth label, to be able to compare if the model is correct
- “ml.prediction_probability”, the models assigned probability for its prediction
- “ml.feature_importance”, the importance of features for this one specific datapoint
- “ml.prediction_score”, when configuring the job the user can define a “class_assignment_objective”. The ml score shows how strongly the model prefers the predicted class after taking into account the objective
- “ml.top_classes”, class scores and probabilities for every possible class the model could assign so the user can backtrace the decision

In summary the built in DFA evaluation workflow is convenient and fast to use, but it does not expose all metrics needed to completely standardize in comparison to the custom evaluation views.

5.3.2 Custom Model Evaluation

The custom model does not create its own evaluation dashboard after it is finished. Because of this one has to be created manually. This is done by storing the models testing performance metrics in its own index and then creating a custom dashboard based on that index.

The custom model integration works end-to-end (training outside Elastic, importing, and evaluating), but its performance is sensitive to dataset imbalance and the amount of time invested in feature engineering and training strategy. Importing multiple custom models requires a traceability strategy. In practice this means ensuring unique model identifiers (Elastic does not allow duplicate model names) and a consistent approach for storing per-run performance metrics so results remain attributable to the correct model.

5.3.2.1 Index Structure

The script that trains the custom model includes a part that saves the metrics of the test run and pushes them into a new index.

```
metrics = {
    "@timestamp": datetime.now(timezone.utc).isoformat(),
    "report": report, # Classification report generated by sklearn
    "confusion_matrix": matrix,
    "importance_table": importance_table,
}

index_name = "ml-test-performance-metrics"

client = self.es_client

# Create if not exists
if not client.indices.exists(index=index_name):
    client.indices.create(index=index_name)

doc = metrics

client.index(index=index_name, document=doc)
```

Listing 9: Storing model performance metrics

Every row in the “ml-test-performance-metrics” index is a new training/ testing run and saves the data we need to show in the custom dashboard. The relevant field data in the index is stored like this.


```
...
"@timestamp": "2025-12-08T15:52:32.721678+00:00",
"report": {
  "anomaly_precision": 0.6,
  "anomaly_recall": 0.15789473684210525,
  "anomaly_f1-score": 0.25,
  "anomaly_support": 19,
  "accuracy": 0.9917431192660551
},
"confusion_matrix": {
  "all_samples": 2180,
  "true_negatives": 2159,
  "false_positives": 2,
  "false_negatives": 16,
  "true_positives": 3
},
"importance_table": {
  "client_bytes_sum": "0.477",
  "@timestamp_value_count": "0.241",
  "url_path_cardinality": "0.193",
  "error_count": "0.053",
  "http_response_status_code_cardinality": "0.036"
}
...
```


Listing 10: Performance Metrics Example Document

5.3.2.2 Confusion Matrix

The confusion matrix is the “base layer” of the models evaluation, based off the confusion matrix a group of other important metrics can be calculated, which are described in the next chapter. A confusion matrix is a 2x2 matrix, where one axis depicts the true classes and the other axis shows the predicted classes. Figure 27 shows a confusion matrix from a models test run.

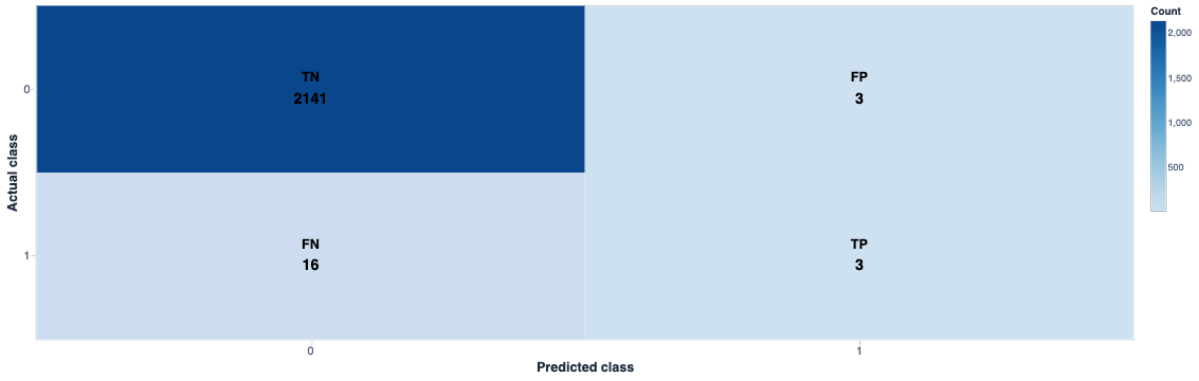


Figure 27: Confusion matrix

This confusion matrix describes a run with 2163 samples (sum every quadrant together). A quick explanation for each value:

- TN = True Negative, the datapoint was not an anomaly and the model correctly predicted “no anomaly”
- TP = True Positive, the datapoint was an anomaly and the model correctly predicted “anomaly”
- FN = False Negative, the datapoint was an anomaly but the model incorrectly predicted “no anomaly”
- FP = False positive, the datapoint was not an anomaly but the model incorrectly predicted “anomaly”

This chapter explains the general procedure of how the models will be evaluated and is therefore kept as general as possible. The detailed analytics of our models can be found in Section 7 .

5.3.2.3 Used Metrics

There are a couple of standard metrics used to evaluate a machine learnings performance.

Metric	Definition	Mnemonic/ Remark
Accuracy	Relative amount of correctly identified sessions	“How often is the model right over all? (Anomalies and no anomalies)”
Precision	Amount of correctly predicted anomalies relative to all predicted anomalies	“How often are anomaly predictions correct?”
Recall	Amount of correctly predicted anomalies relative to all actual anomalies	“Does the model find all anomalies?”
F1-Score	Balance between precision and recall	Helpful in scenarios where the dataset is imbalanced
Support	Absolute number of anomalies (positive class) in the test set	Useful metric to understand the dataset

Tab. 7: Used Metrics

5.3.2.4 Custom Dashboard

As explained in Section 5.3.1 Elastic offers pre-made result explorers only for their built in Data Frame Analytics jobs. For this reason a custom dashboard had to be created.

Figure 28 shows a mockup idea of the model evaluation dashboard. Going by rows (left to right):

- Row 1: Confusion matrix, Ratio of identified vs all anomalies, tile with metrics listed in Tab. 7
- Row 2: Processed documents over time, Anomalies over time, Anomalies to “No anomalies” ratio

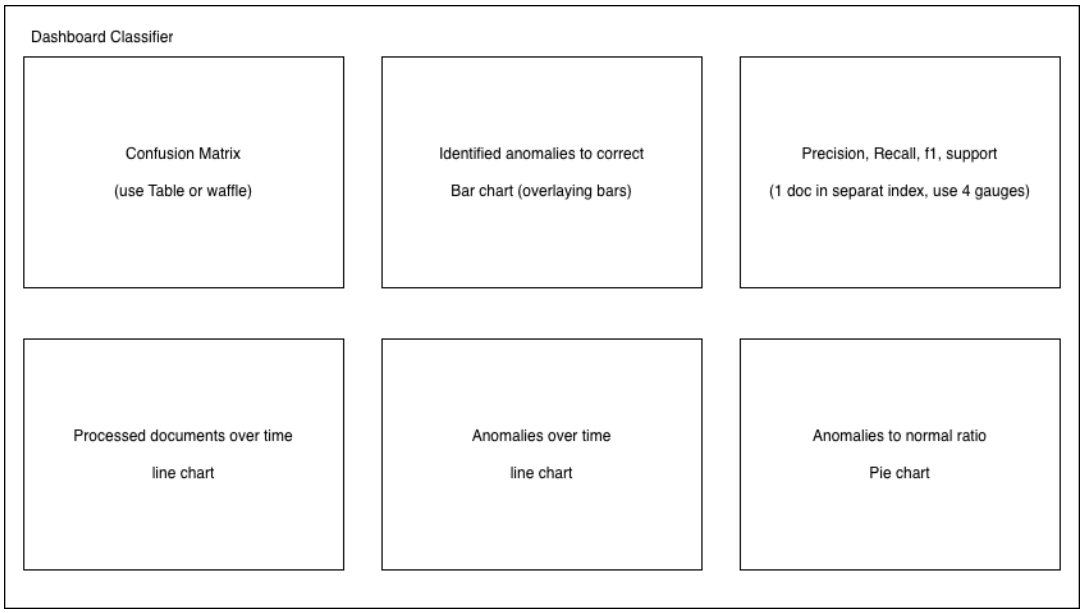


Figure 28: Kibana Dashboard Mockup

The initially conceptualized “tiled” view was deemed impractical to implement without substantial effort. Therefore, the design was revised to adopt Elastics default layout with collapsible vertical sections.

That resulted in the following sections, shown in Figure 29 :

- Row 1: Confusion Matrix, Metrics calculated off the confusion matrix with slider chats for better visuals
- Row 2: Line-Graphs for total documents vs anomalies, Pie chart for total documents vs anomalies

On first sight there are a couple of metrics missing, which were planned in the mockup. Those can be explained. The middle metric in the top row “Ratio of identified anomalies vs all anomalies” already is the Recall metric which is displayed anyways. To save space the “Processed documents over time” and the “Anomalies over time” graphs have been merged into one. By making these changes the final version of the dashboard ends up with a 2x2 matrix design, that you can see below.

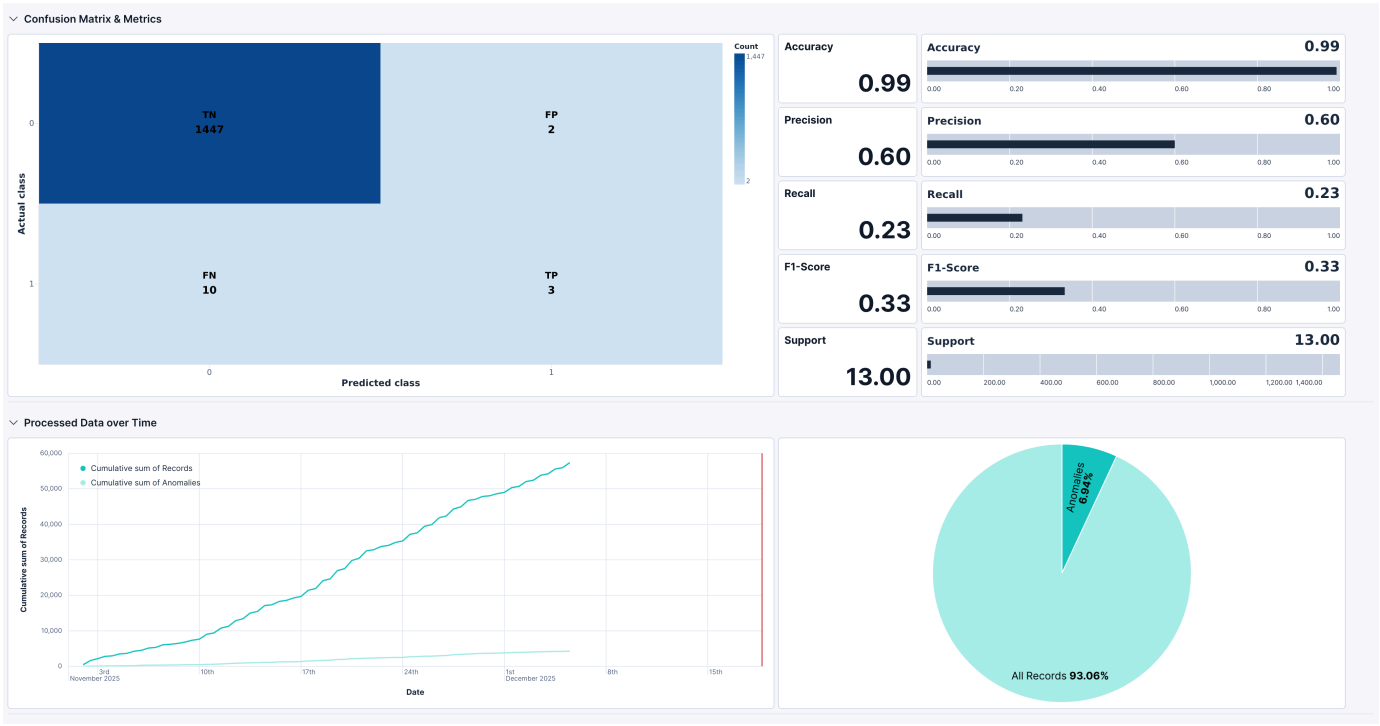


Figure 29: Final version of Kibana Dashboard

The dashboard enables repeatable comparison of model runs once the performance metrics are stored in an index, but requires manual steps to set up. Currently, users have to create a new Data View and adjust existing visualizations when switching to a new model. A potential improvement would be to store all model runs in a single metrics index and add a model identifier that can be filtered in the dashboard, reducing the need to create new indices and dashboards per model.

6 Setup Guide

The setup guide contains all detailed steps necessary to set up the project. The code for this project can be cloned from the git repository under the following link: https://gitlab.ost.ch/matteo.mahler/sa-codeconfig .

6.1 Elastic Setup

This section explains all setup steps in regards to Elastic.

6.1.1 Linked configurations

First, clone the repository and navigate to the `services/elastic/` directory. After that, either follow the quick setup guide in the README.md file located in the same directory or continue reading here.

1. `docker-compose.yml`: This file contains the necessary services to run the ELK stack with Docker.
2. Rename `env.example` to `.env` and fill in the XPACK_ENCRYPTEDSAVEDOBJECTS_ENCRYPTIONKEY. Use the following command to generate a random 32 character long key:

   ```
   openssl rand -base64 32
   ```

   Additionally define `ELASTIC_PASSWORD` and `KIBANA_PASSWORD`.
3. Start the stack with `docker-compose up -d`.

You have a running Elasticsearch and Kibana instance with the necessary configurations to run our setup. The Kibana UI is available at `http://localhost:5601` and logging in is possible with the `elastic` user and the password set up in step 2.

The `docker-compose.yml` sets low memory limits for Elasticsearch which are limiting to its ability to train and run ML models. If enough memory is available on the host machine, it is recommended to increase the memory limits in the `docker-compose.yml` file. Do this by changing the `ES_JAVA_OPTS` environment variable in the `elasticsearch` service section. For example, you can set it to `-Xms2g -Xmx2g` to allocate 2GB of memory to Elasticsearch or even higher. Giving Elastic 4GB on the development machines produced good results.

6.1.2 Setup: PWS, ingress pipelines, API keys, etc.

6.1.2.1 First login

After the stack has started you will be greeted with the login screen.

Use the `elastic` user and the password you set earlier to log in. It is good practice to create a new user that does not have superuser permissions. You can do that by navigating to “Stack Management” > “Management” > “Security” > “Users”.

6.1.2.2 Create API Key

Sign in with the new user you just created. Create an API key that will be used by Logstash to send data to Elasticsearch. To create an API key, navigate to “Stack Management” > “Security” > “API Keys” and click on “Create API Key”.

You will see the generated API key once its created. There are different formats available as visible in Figure 30 . The Beats and Logstash format are the same.

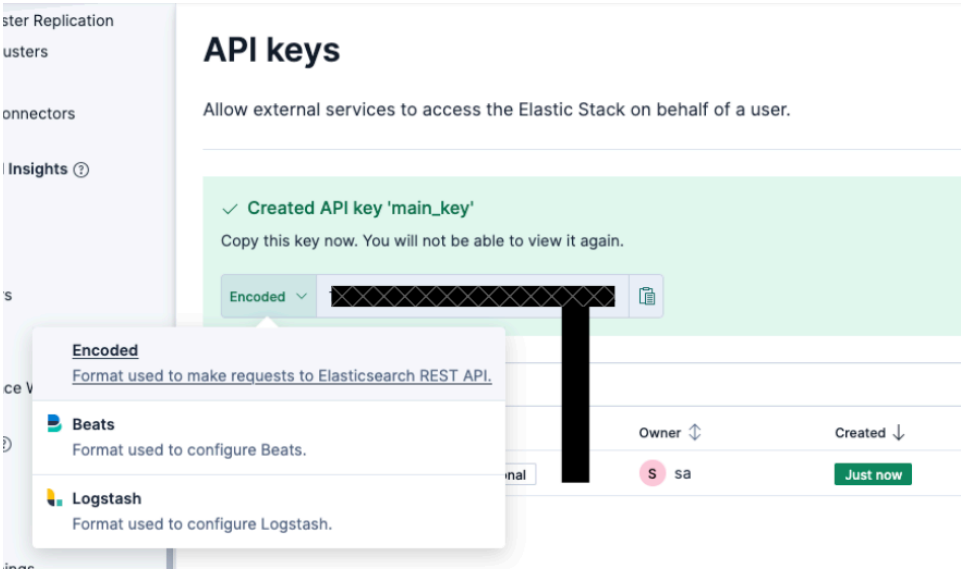


Figure 30: API Key

Write down both values, the encoded one as well as the Beats format.

The Beats format for Filebeat and Logstash and the Encoded version for the ML model so you can upload the model later directly in the python script.

6.1.2.3 Import Saved Objects

The import of saved object only works when using the provided data from Section 6.1.2.4 . If you use your own data you need to adjust the Student Portal Data View to use your own index.

The validation dashboard we have created to visualize the training result and the model performance is stored inside the git repository as a saved object export. To import it, navigate to “Stack Management” -> “Saved Objects” and click on “Import”. All saved objects are located in the `services/elastic/saved_objects/` directory.

This will import:

- The validation dashboard
- ML Test Performance Metrics Data View
- Student Portal Data View

6.1.2.4 Import Data Dumps

If you use the imported data dumps you do not need to run our data generator or student portal. If you want to generate your own data you need to finish Section 6.2 and Section 6.3 first. The data dumps are like an existing dataset you can just import in Elastic. These dumps are not in the code repository because of their size. They will be handed in zip file in the `elastic_data_dumps/` directory.

The indices containing data are exported for training into json files using elasticsearch-dump . Install the tool and follow the install instructions in the basic http auth note to setup authentication.

To import them, navigate to “Stack Management” > “Index Management” and create new indices with the name: “student-portal” and “ml-test-performance-metrics”. After that, the dumps can be imported using elasticsearch-dump. First the mapping has to be imported as shown in Listing 11 .

```
NODE_TLS_REJECT_UNAUTHORIZED=0 elasticdump \
--output=https://localhost:9200/student-portal \
--input=./student-portal_mapping.json \
--type=mapping
```

Listing 11: Import index mapping of Student Portal Index

```
NODE_TLS_REJECT_UNAUTHORIZED=0 elasticdump \
--output=https://localhost:9200/ml-test-performance-metrics \
--input=./ml-test-performance-metrics_mapping.json \
--type=mapping
```

Listing 12: Import index mapping of Performance Metrics Index

In a second step, the data can be imported as shown in Listing 13 .

```
NODE_TLS_REJECT_UNAUTHORIZED=0 elasticdump \
--output=https://localhost:9200/student-portal \
--input=./student-portal_data.json \
--type=data
```

Listing 13: Import Student Portal index data

```
NODE_TLS_REJECT_UNAUTHORIZED=0 elasticdump \
--output=https://localhost:9200/ml-test-performance-metrics \
--input=./ml-test-performance-metrics_data.json \
--type=data
```

Listing 14: Import Performance Metrics index data

For exporting the data from Elasticsearch, use the same commands by switching the input and output parameters.

6.1.2.5 Create Ingest Pipelines

The setup requires Elasticsearch to have a GeoIP ingest pipeline to enrich incoming logs with geo information based on the client IP address. For that, copy the code block below and paste it into the Dev Tools Console in Kibana to create the GeoIP ingest pipeline. The Dev Tools Console is located in the navigation under “Elasticsearch” > “Home”, in the dark gray footer of the page, click on “Console”. See Figure 31 for where to paste the code block. When pasting the codeblock enter a new line after “`PUT _ingest/pipeline/geoip-pipeline`”. After clicking the “Play” button you should see a success message and the pipeline is created.

```
PUT _ingest/pipeline/geoip-pipeline
{
  "description": "Add geo info from client.ip",
  "processors": [
    {
      "geoip": {
        "field": "client.ip",
        "target_field": "client.geo"
      }
    }
  ]
}
```

Listing 15: Create GeoIP Ingest Pipeline

Figure 31: GeoIP Ingest Pipeline

6.1.3 Elastic ML (required license) capabilities in depth

To enable Elastics Machine Learning / AI capabilities the instance needs to run on a platinum license [18] . This includes the usage of Data Frame Analytics (Section 5.1) and importing custom models into the Elastic instance (Section 6.1.3.3). The trial is automatically activated if you follow this setup guide.

6.1.3.1 Setup Transforms

Before beginning this step you need to have data in your Elastic instance. Either follow the instructions in Section 6.1.2.4 to import existing data or Section 6.2 and Section 6.3 to generate your own data.

To setup a Transform navigate to the “Stack Management” > “Transforms”.

After clicking on “Create your first transform”, select a source for the Transform. Use the “Student Portal”, as this is the main index where all the generated data gets stored.

In the next step a “pivot” Transform is needed since the data needs to be pivoted from an action centric entity to a user session centric entity. For the time frame to include in the pivot, select “Use full data” since all of the data is relevant for training.

To create a user session pivot, group the data by `terms(labels.sessionId)`. The following aggregate fields were defined in this project which are all configurable through the GUI without the need to adjust them in their JSON format, which would also be possible if one needed complex logic behind an aggregation field.

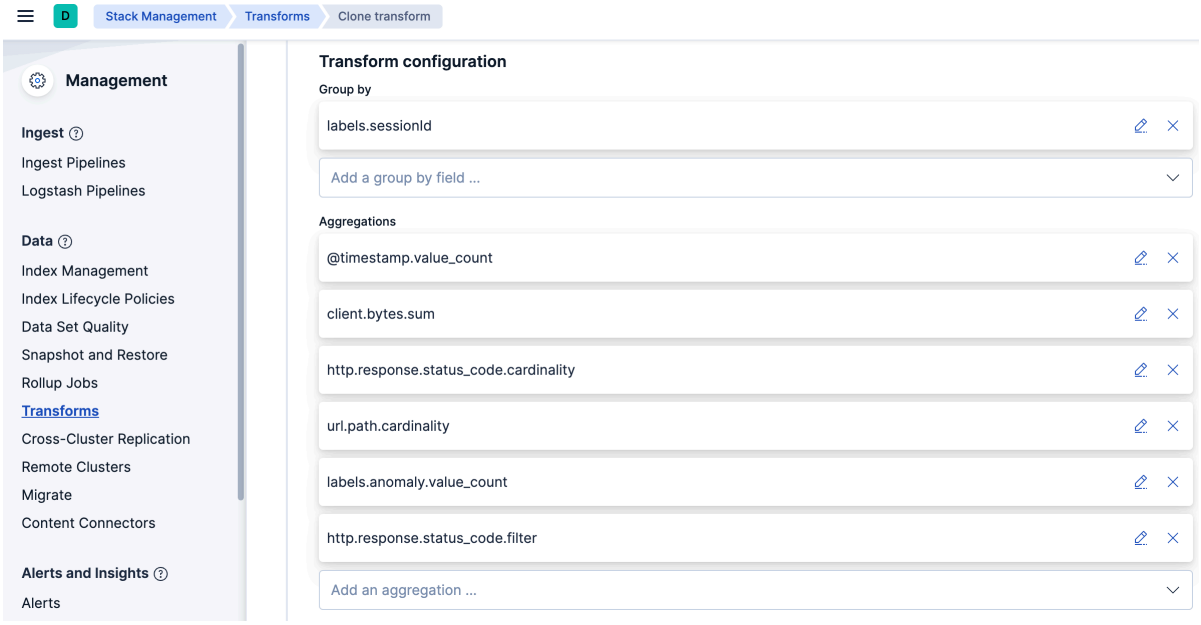


Figure 32: Configure Transform Groups and Aggregates

After grouping the data it makes sense to create aggregates such as counts, sums and cardinalities. The aggregated values compress the individual values in their groups into one meaningful feature, this reduces noise from single user actions and provides a more compact representation. As described in Section 4.2.1 it was decided to use the following features. Here is a quick step-by-step on how to set them up.

Field	Aggregation	Setup
@timestamp	value_count	Type “timestamp” into the “Add an aggregation” input field and select “value_count(@timestamp)”
client.bytes	sum	Type “client.bytes” into the “Add an aggregation” input field“ and select “sum(client.bytes)”
http.response.status_code	cardinality	Type “http.response.status_code” into the “Add an aggregation” input field and select “cardinality(http.response.status_code.cardinality)”
url.path	cardinality	Type “url.path” into the “Add an aggregation” input field and select “cardinality(url.path)”
http.response.status_code	filter	Type “http.response.status_code” into the “Add an aggregation” input field, select “filter(http.response.status_code)”. An additional pop up field will open. First, define the Aggregation name as “error_count”. Second, in the “Filter query” dropdown, select “range”. Two more input fields are added to the bottom of the pop up. To get all errors we need the range from >399 and <600 (HTTP 4XX errors signify client errors, 5xx signify server errors). In “Greater than” put in 399 and in “Less than” put in 600 (or click on the “<” symbol to turn it into “≤”, then you can put in 599). Click on apply and you have created an aggregation field that counts all http errors in that session.
labels.anomaly	value_count	Type “labels.anomaly” into the “Add an aggregation” input field and select “value_count(labels.anomaly)”

Tab. 8: Aggregation Fields Setup

After defining the aggregate fields Kibana shows a preview how your dataset would look after applying the aggregation functions.

In next section, define the transform ID “user-session-labelled”, which is the name elastic will store the Transform object under. Do not run the Transform in continuous mode, since this is our training index, we want to keep this data limited and separate from the real data.

Do not change any of the advanced settings.

In the last step, click on “Create and start” to create your transform.

If you navigate back to the existing Transforms view you will see the new Transform under the name you have input in the “Transform ID” field.

6.1.3.2 Setup DFA Job

Configure a new Data Frame Analytics job, by navigating to “Stack Management” > “Data Frame Analytics Jobs”. Select the “user-session-labelled” index, on which the job should run its classification. Configure the job to be a classification job (as this is the goal, classify a session either as an “anomaly” or “no anomaly”).

In this state, the truth label of the labelled index is the number of anomalous actions in that session, this needs to be changed. Training with this field as is, results in infinite possible classes (since a session can potentially have infinite anomalous actions). The goal is to have two possible classes “anomaly” and “no anomaly”. Runtime fields are helpful in this case. With this option the index can be enriched with data from other fields (from the same index) available at runtime. Configure a new field called “anomaly_class” by clicking on the toggle next to “Edit runtime fields” and pasting this JSON config into the text field:

```
{
  "anomaly_class": {
    "type": "long",
    "script": {
      "source": "if (doc['labels.anomaly.value_count'].size() == 0) { emit(0L); } else
{ emit(doc['labels.anomaly.value_count'].value > 0 ? 1L : 0L); }",
      "lang": "painless"
    }
  }
}
```

Listing 16: Runtime Field Configuration

This field first checks whether the `anomaly_count` field exists at all. If it does, it further checks whether the `anomaly_count` is bigger than 0, if it is, the new `anomaly_class` field gets assigned the value 1 (anomaly), if the value of `anomaly_count` is 0, then `anomaly_class` is also assigned 0 (no anomaly). By doing this, the possibility of infinite different classes gets reduced to the two desired classes, 0 or 1 (anomaly or no anomaly).

After defining the runtime field, select it as the “Dependent variable”, this is the variable the model will try to predict.

In the next step the fields which the model includes in its training are selected. These are the configured selected fields:

- `@timestamp.value_count`
- `anomaly_class`
- `client.bytes.sum`
- `error_count`
- `http.response.status_code.cardinality`
- `url.path.cardinality`

The anomaly count is excluded because this is the field, on which the truth label is based. If this field were included, the model would just learn to look at the anomaly count, and assign class “1” (anomaly) if the count is bigger than 0. Excluding “labels.sessionId” is based on ML best-practices where it is not recommended to include names or other identifiers since the model would start learning them. This is problematic because if a new session non-anomalous identifier would be similar by text value to an anomalous identifier the model learned in training, the model could assign an anomaly, just because the identifiers are similar.

When looking at the list it might seem a bit confusing as to why the `anomaly_class` field, that was just created is also included, without the possibility to remove it from the list (the checkbox is disabled), but this is expected. The dependent variable in supervised machine learning is the one, which the model is trains to predict. Therefore it must be included so the model can learn the relationship between the dependent variable (the target) and the features.

At last define the training-split. Setting it to 80 means that 80% of all available data during the run will be used for training and the other 20 for testing.

Not much time was invested into the tuning of hyperparameters because the focus of this work is to provide a testbed, not extract the best performance out of the models. For this project Elastics default values produce good enough results.

In the next step define a name, for the other settings use their defaults.

6.1.3.3 Import custom model

Rename `example.env` to `.env` and add the two API keys, which you have copied from Figure 30 to their respective variables in the .env file.

The import of the custom model is straight forward. The script, that trains the ML model, described in Section 5.2.1 , also includes a snippet to upload the model to Elastic.

```
user_input = input("Store model in Elastic? y/other: ")
if user_input.lower() == "y":
  # Import model into Elastic
  model_id = "session-anomaly-v1"
  es_model = MLModel.import_model(
      es_client=es,
      model_id=model_id,
      model=clf,
      feature_names=feature_cols,
      classification_labels=["no_anomaly", "anomaly"],
  )
  print(f"Model imported with ID: '{model_id}'")
else:
  print("Skipping storing model in Elastic.")
```

Listing 17: Import the model into Elastic

The import of the model is skippable by the user. This is helpful in case the models test performance is unsatisfactory and the user does not want to import it in order to not pollute the Elastic workspace with unusable models.

To import the model run the script `model.py`. On the first interrupt type ‘y’ to import the model into Elastic. On the second interrupt type ‘y’ to import the models test performance data into its own index for later evaluation.

For further insight you can read the README.md in the root directory of the code repository.

To check whether the model was correctly imported, navigate to “Stack Management” > “Trained Models” and you will see the name of your model in the list.

6.2 Application Setup

To create the initial database start the docker-compose stack.

1. Setup the `.env` file in the `services/` directory: Use the provided `env.example` file as a template. Setup the remaining empty environment variables.
2. Run `docker-compose up -d` in the `services/` directory. This will start PostgreSQL, filebeat, logstash and the student-portal application. The student portal application will automatically create the database schema on startup if it does not exist yet and seed the admin.

Once the application is running, interact with it via the API or run the provided traffic generator to simulate user behavior and generate logs. See Section 6.3 for more information about the traffic generator.

With the environment variable `DEMO_MODE` set to `true`, the application will trust some additional headers to simulate things like ip, user agent etc. This is required when using the traffic generator. See that all headers defined in Tab. 5 are set for the logging to work, the bold printed headers are required.

6.3 Traffic Generator Setup

The traffic generator is located in the `services/traffic-generator/` directory, but is also available as a Docker container. Since the traffic generator is not mandatory to use the application, it is not included in the main `docker-compose.yml` file located in the `services/` directory. To run it via Docker, you can use the provided `docker-compose.yml` file located in the `services/traffic-generator/` directory.

When running the traffic generator within docker with the default configuration of workers, it would not start since too many threads try to start. Additionally, very low CPU usage was observed in contrast to a local execution with very high CPU load during start, which indicates that docker is not utilizing all available resources of the host machine.

There are two ways to overcome this issue:
1. Reduce the number of workers in the `.env` file.
2. Run the traffic generator outside of docker.

All the flows spawn their own thread and will then run independently of each other. Every thread initiates its own session by logging in and then start performing the defined actions. A random device will be selected for these sessions. Every device has corresponding IP addresses available to simulate mobile users that are accessing the application via cellular networks for example while keeping the session bound to one browser.

The traffic generator will keep running until you stop it with `CTRL+C` or the container is stopped.

After data has been generated, and you want to generate new data, reset the following:
- `app.log`
- Postgres DB
- Clear/ delete and recreate the index or point the new data to a new index
- Restart the application
- Seed new users

6.3.1 Running the Traffic Generator in Docker

1. Create a `.env` file in the `traffic-generator` directory. Use the provided `env.example` file as a template. Use the same admin password as defined in the `services/.env` file.
2. After configuring everything, start the traffic generator with the following command:

```
docker-compose up
```

This starts the seeding of the users. After the seeding is finished, open the `traffic-generator/.env`, set `SEED=false` and run `docker compose up` again.

6.3.2 Running the Traffic Generator on the host

Alternatively, you can run the traffic generator locally without docker by running:

```
npm ci
node main.js
```

in the `services/traffic-generator/` directory. This starts the seeding of the users. After the seeding is finished, open the `traffic-generator/.env`, set `SEED=false` and run `node main.js` again.

7 Results

This chapter outlines the results that were achieved while writing this paper. It is important to remember that the results scrutinize the workings of the testbed and how easy it makes to compare different models on the same dataset. The models themselves were not the focus of this project, and while there will be a brief section about their performance and possible issues, it will not be as extensive.

7.1 Summary

The main goal of the project, creating a functional testbed to test ML models performance in user behavior anomaly detection, was successful. All components work reliably and the chosen architecture described in Section 2.3 shows that our approach is feasible with the possibility for future work and research to be done on it.

Data generation is reliable and enables us to train our ML models. Ingestion pipelines work flawlessly once setup and did not cause us any problems after the initial setup phase. The Transforms execute correctly according to the setup and enable the aggregation of raw logs into session-level features, but additional feature engineering definitely could improve models performance. The out of the box Data Frame Analytics work as expected and provide an easy to use setup experience. But is missing some metrics in the Results Viewer, which would have enabled on par evaluation. It would have been optimal to setup a custom dashboard but Elastic does not make their proprietary models test performance data available to work with. The custom machine learning model is the biggest weak point. While it works and would be able to run inference on an unlabelled dataset, the test and real world performance of the model definitely has room for improvement. There are multiple ways to solve this issue, which are explained in detail in Section 7.2 . Also the import procedure of the custom model requires Elastic-specific knowledge. As soon as the knowledge transfer happened, the process is not overly complex but it would be nice to smooth down this overhead. The dashboard itself works as expected and displays the required data in a clear manner once running. The dashboards setup process has the same challenge as the custom machine learning models setup, it does require some specific knowledge of the inner workings of this project, but once this knowledge is acquired the setup is not overly complicated. It would still be nice to reduce this overhead as well.

The deployment has a couple of steps the users must follow, but once set up, the system operates very stable and no unexplainable crashes were discovered during development.

Overall, once deployed and running, the system provides a stable testbed to evaluate different ML models for anomaly detection. This project shows that an automatic anomaly detection pipeline is feasible and opens up to new ideas for future work in both operational monitoring and machine learning research.

7.2 Findings

This section provides concrete answers to the research gap formulated in Section 1.3 . It consolidates what was learned across the implementation chapters and explains how these learnings demonstrate that a standardized evaluation of anomaly detection models in Elastic is feasible.

7.2.1 Gap: No suitable dataset for user behavior anomalies in web applications

The documentation shows that this gap can be addressed by generating a tailored dataset that is repeatable and purpose built for session based anomaly detection.

By implementing a data generator with deterministic regular and anomaly flows (Section 3), we created a dataset that matches the target domain (layer 7 network traffic) and contains a controlled amount of anomalous behavior. This makes repeated experimentation possible. The same types of sessions and anomalies can be recreated for different models.

At the same time, the work also shows the main problem of this approach: Synthetic data is less diverse than real world traffic. Therefore, the dataset is sufficient to validate the testbed and compare models, but it should not be seen as a replacement for real world data.

7.2.2 Gap: No standardized way to evaluate custom models in Elastic

The documentation demonstrates that a standardized evaluation procedure for custom models can be implemented in Elastic by storing and visualizing test performance data.

The key design choices to standardize evaluation:
- A consistent dataset entity (sessions rather than single requests) created via Transforms (Section 4.2)
- A consistent metric foundation (confusion matrix and derived metrics) described in Section 7
- Consistent visualization and comparison using Dashboards (Figure 29)

For Elastics built in approach, the DFA Results Explorer already provides evaluation views. For custom models, the Section 5.3.2 shows that similar evaluation becomes possible by storing per run test performance metrics and displaying them in a custom Kibana dashboard. This addresses the gap of missing standardized evaluation for custom models by making evaluation repeatable and comparable across multiple training runs.

However, the work also highlights a boundary: The DFA evaluation is integrated and easy to use, but not all metrics are directly accessible for reuse in our own dashboard. This means full parity between built in and custom evaluation cannot be achieved do to Elastic not exposing their test performance data.

7.2.3 Gap: Performance differences must be assignable only to a changed model

Across the documentation, everything except the model was kept stable by design:
- Same ingestion pipeline (Section 4)
- Same Transform logic and feature definitions (Section 4.2)
- Same evaluation methodology (Section 5.3)

Because these parts remain constant, the model (e.g. DFA vs custom import) becomes the only changing variable. This turns the setup from a simple pipeline into a testbed: Differences in evaluation results can be interpreted as differences between models, rather than differences caused by data handling or differing infrastructure.

7.2.4 Conclusion: A SIEM testbed for different ML models is feasible

The implementation described in Chapters 2 – 5 shows that the proposed setup is not a lab experiment but integrates into the Elastic Stack workflow. The logs are ingested and transformed into a representation suitable for ML. Built in and custom training are run, and the results consistently evaluated.

Therefore, the research gap is addressed: The work provides a reproducible environment for benchmarking anomaly detection models within a SIEM system, and it makes the evaluation of custom models possible through standardized visualization.

7.3 Future work

There are multiple possibilities for future projects that can be built on top of this testbed. This is a small selection of project we would find interesting/ would solve pain-points we have discovered during the writing of this project.

1. **Improve the ML models performance**. This is the biggest weak point of this project at this time and can be mitigated in various ways. One could invest time into proper feature engineering with mathematical and algorithmic procedures or create custom training loops, which better fit the current data.
2. **Run the system with live data**. Another issue (which also negatively impacts the problem of model performance) is the data itself. A large part of the time budget was spent on creating synthetic data to be able to run any type of machine learning algorithm on the dataset. While we did our best, the data is not as good as real data from a live system would be. It was tried to vary the user flows as much as possible but they are still repetitive.
3. **Improved Dashboard Automation**. Implement model run tags in the performance index to be able to filter the dashboard automatically without the need of creating a new one.

Section II
Glossary and List of Abbreviations

ML	Machine Learning
AI	Artificial Intelligence
SIEM	Security, Information and Event Management
PoC	Proof of concept
ERD	Entity Relationship Diagram, a type of diagram used to describe database structures
Beat	A Beat is a lightweight data shipper in the ELK stack that collects and forwards data from various sources (logfiles, events, metrics) to Elasticsearch or Logstash.
Filebeat	Filebeat is a tool from the Elastic Stack which centralizes access to log files and enables you to transfer them to destination across the whole Elastic Stack.
DFA	Data Frame Analytics. A built-in tool in Elastic to run ML algorithms on a dataset

Section III
Bibliography

[1] Aditya Raj and Sanskar Sharma, "A Comprehensive Study on Anomaly Detection Methods Using Traditional and Machine Learning Approaches." Accessed: Dec. 16, 2025. [Online]. Available: https://terra-docs.s3.us-east-2.amazonaws.com/IJHSR/Articles/volume6-issue11/IJHSR_2024_611_9.pdf

[2] Sidahmed Benabderrahmane, James Cheney, and Talal Rahwan, "Ranking-Enhanced Anomaly Detection Using Active Learning-Assisted Attention Adversarial Dual AutoEncoders." Accessed: Dec. 16, 2025. [Online]. Available: https://arxiv.org/pdf/2511.20480v1

[3] Jared Council, "Data Challenges Are Halting AI Projects, IBM Executive Says." Accessed: Nov. 25, 2025. [Online]. Available: https://www.wsj.com/articles/data-challenges-are-halting-ai-projects-ibm-executive-says-11559035800

[4] László Göcs and Zsolt Csaba Johanyák, "Identifying Relevant Features of CSE-CIC-IDS2018 Dataset for the Development of an Intrusion Detection." Accessed: Sept. 18, 2025. [Online]. Available: https://arxiv.org/abs/2307.11544

[5] Surasit Songma, Theera Sathuphan, and Thanakorn Pamutha, "Optimizing Intrusion Detection Systems in Three Phases on the CSE-CIC-IDS-2018 Dataset," 2023. doi: 10.3390/computers12120245 .

[6] "CSE-CIC-IDS2018 on AWS." Accessed: Sept. 18, 2025. [Online]. Available: https://www.unb.ca/cic/datasets/ids-2018.html

[7] "KDD Cup 1999 Data." Accessed: Sept. 18, 2025. [Online]. Available: https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[8] [Online]. Available: https://www.kaggle.com/datasets/fcwebdev/synthetic-cybersecurity-logs-for-anomaly-detection

[9] "Web Server Access Logs." Accessed: Sept. 23, 2025. [Online]. Available: https://www.kaggle.com/datasets/eliasdabbas/web-server-access-logs

[10] "Awesome-Cybersecurity-Datasets." Accessed: Sept. 23, 2025. [Online]. Available: https://github.com/shramos/Awesome-Cybersecurity-Datasets

[11] "Network Traffic Dataset for Anomaly Detection." Accessed: Sept. 23, 2025. [Online]. Available: https://huggingface.co/datasets/onurkya7/NADW-network-attacks-dataset

[12] Venkata Gudelli, "Enhancing Application Security Using Web Application Firewalls and AI," 2023. doi: 10.37082/IJIRMPS.v11.i5.232166 .

[13] Australian Signals Directorate's Australian Cyber Security Centre (ACSC), U.S. Cybersecurity and Infrastructure Security Agency (CISA), and U.S. National Security Agency (NSA), "Preventing Web Application Access Control Abuse." Accessed: Oct. 04, 2025. [Online]. Available: https://www.cisa.gov/sites/default/files/2023-07/aa23-208a_joint_csa_preventing_web_application_access_control_abuse.pdf

[14] "Eland documentation - Importing Models." Accessed: Nov. 25, 2025. [Online]. Available: https://eland.readthedocs.io/en/latest/reference/api/eland.ml.MLModel.import_model.html

[15] "Eland documentation." Accessed: Nov. 25, 2025. [Online]. Available: https://eland.readthedocs.io/en/v9.2.0/

[16] Bamidele Matthew, John Jude, and Michelle James, "Applications of Anomaly Detection," 2025. Accessed: Nov. 25, 2025. [Online]. Available: https://www.researchgate.net/publication/391572833_Applications_of_Anomaly_Detection

[17] "Elastic Transforms." Accessed: Dec. 15, 2025. [Online]. Available: https://www.elastic.co/docs/explore-analyze/transforms

[18] "Elastic License." Accessed: Nov. 25, 2025. [Online]. Available: https://www.elastic.co/subscriptions

Section IV
List of Illustrations

Section V
List of Tables

Section VI
Code-Listings

Section VII
Appendix

8 List of Aids

Area of Responsibility	Tools
Literature research and management	Google, Google Scholar, IEEE Library, arxiv, kaggle, GitHub, huggingface, Elastic Docs, ChatGPT
Data Analytics and visualization	Excel, ChatGPT, Figma
Idea generation	ChatGPT, Google Gemini
Prototyping	Draw.io
Coding	VS Code, IntelliJ, PyCharm, GitHub Copilot, ChatGPT, Google Gemini
Text creation, text optimization, spelling and grammar checking	Typst, VS Code, DeepL, ChatGPT
Collaboration and project management	Outlook, Teams, GitLab, Jira, Notion
DevOps	Docker, GitLab, ELK Stack, Postgres, Node JS, UV
Image creation (Title Image)	ChatGPT SORA

Tab. 9: List of Aids