HAW HAMBURG

BACHELOR THESIS
Hani Alshikh

Evaluation and use of Event-Sourcing for audit logging

Faculty of Engineering and Computer Science
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE
WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Hani Alshikh

Evaluation and use of Event-Sourcing for audit logging

Bachelor thesis submitted for examination in Bachelor´s degree
in the study course *Bachelor of Science Angewandte Informatik*
at the Department Computer Science
at the Faculty of Engineering and Computer Science
at University of Applied Science Hamburg

Supervisor: Prof. Dr. Stefan Sarstedt
Supervisor: Prof. Dr. Olaf Zukunft

Submitted on: 20. März 2023

Hani Alshikh

Title of Thesis

Evaluation and use of Event-Sourcing for audit logging

Keywords

Event Sourcing, Auditing, Auditing 2.0, Audit Trail, Audit log, Software Engineering, Software Architecture, gRPC-Web

Abstract

Keeping accurate audit records is a requirement for complaint Information-Technologie (IT) systems, especially when used in sensitive industries such as government, finance, infrastructure, etc.

Event-Sourced architectures are rapidly gaining in popularity as they provide reliability, flexibility, and scalability. One of the primary benefits of Event-Sourcing is that it provides complete and immutable records of all events and state changes within the system, allowing for efficient and thorough audit logging by design.

The benefits and challenges of Event-Sourcing compared to other approaches were examined and evaluated. A Proof Of Concept (POC) auditing component and an audit browser were also developed to showcase what to expect in terms of auditing capabilities as well as laying the groundwork for auditing 2.0 integration...

Contents

List of Figures

List of Tables

Abbreviations

AI Artificial Intelligence.

API Application Programming Interface.

CD Continuous Delivery.

CI Continuous Integration.

CRD Custom Resource Definition.

CSV Comma Separated Values.

ERP Enterprise Resource Planning.

gRPC google Remote Procedure Call.

GUI Graphical User Interface.

HTML HyperText Markup Language.

HTTP HyperText Transfer Protocol.

IAM Identity and Access Management.

IDP IDentity Provider.

ISO International Organization for Standardization.

IT Information-Technologie.

JS Javascript.

JSON JavaScript Object Notation.

k8s Kubernetes.

m8 Monoskope.

MVP Minimum Viable Product.

OIDC Open-ID Connect.

PKI Public Key Infrastructure.

POC Proof Of Concept.

POC Proof Of Concept.

REST REpresentational State Transfer.

TS Typescript.

URI Universal Resource Identifier.

URL Universal Resource Locater.

UX User Experince.

XML Extensible Markup Language.

Shell

1 Acknowledgments

2 Introduction

Audits are systematic and objective examinations of one or more aspects of an organization, that compares what the organization does to a defined set of criteria or requirements. IT auditing examines processes, IT assets, and controls at multiple levels within an organization to determine the extent to which the organization adheres to applicable standards or requirements.

Event-Sourcing is a software architecture pattern that insures a complete log of changes made to a system as a series of events. Instead of storing the current state in a traditional database, Event-Sourcing stores the history of changes made over time. This allows developers to rebuild the current state at any point in time and see exactly how it date has changed by replaying the stored events, which is very useful for debugging and performing rollbacks or reversals of changes.

Having a comprehensive and immutable audit trail makes Event-Sourcing particularly well-suited for systems with complex business processes, that need to track and audit changes to sensitive data and ensure they are in compliance with regulations and standards without relying on traditional logging mechanisms.

In addition to providing a detailed audit trail, Event-Sourcing also offers a number of other benefits. Since the events are stored in a chronological order, it is possible to implement time-based queries and manipulations, which lays the base for Auditing 2.0 discussed in section 3.4.

Beside evaluating Event-Sourcing in regards to auditing and audit controls this work provides a Proof Of Concept (POC) implementation of an Audit Browser for a multi-cloud multi-cluster authentication and authorization system for Kubernetes (k8s) as well as the Audit Component implementation of the reading modles as specified by Auditing 2.0

3 Audit

An audit is often defined as an independent examination, inspection, or review. While the term applies to evaluations of many different subjects, the most frequent usage is with respect to examining an organization's financial statements or accounts. Words like assessment, evaluation, and review are often used synonymously with the term audit and while it is certainly true that an audit is a type of evaluation, some specific characteristics of auditing distinguish it from concepts implied by the use of more general terms.

An audit always has a baseline or standard of reference against which the subject of the audit is compared. An audit is not intended to check on the use of best practices or to see if opportunities exist to improve or optimize processes or operational characteristics. Instead, there is a set of standards providing a basis for comparison established prior to initiating the audit. [Gantz, 2014]

Audit determinations tend to be more binary than results of other types of assessments or evaluations, in the sense that a given item either meets or fails to meet applicable requirements. Auditors often articulate audit findings in terms of controls' conformity or nonconformity to criteria.

The International Organization for Standardization (ISO) guidelines on auditing use the term audit to mean:

> A systematic, independent and documented process for obtaining objective evidence and evaluating it objectively to determine the extent to which the audit criteria are fulfilled [ISO 19011, 2018]

In contrast to conventional dictionary definitions and sources focused on the accounting connotation of audit, definitions used by broad-scope audit standards bodies and in IT auditing contexts neither constrain nor presume the subject to which an audit applies.

Such general interpretations are well suited to IT auditing, which comprises a wide range of standards, requirements, and other auditing criteria to audit IT subjects.

3.1 IT Audit

IT audit is the process of collecting and evaluating evidence of an organization's IT systems, practices, and operations to determine whether they are adequate, efficient, and effective in meeting the organization's objectives. [Gantz, 2014]

An IT audit typically includes a review of an organization's policies, procedures, and controls related to its IT systems, as well as an assessment of the security, reliability, and performance of its IT infrastructure. The goal of an IT audit is to identify any weaknesses or deficiencies in an organization's IT systems and recommend improvements that can help the organization achieve its goals.

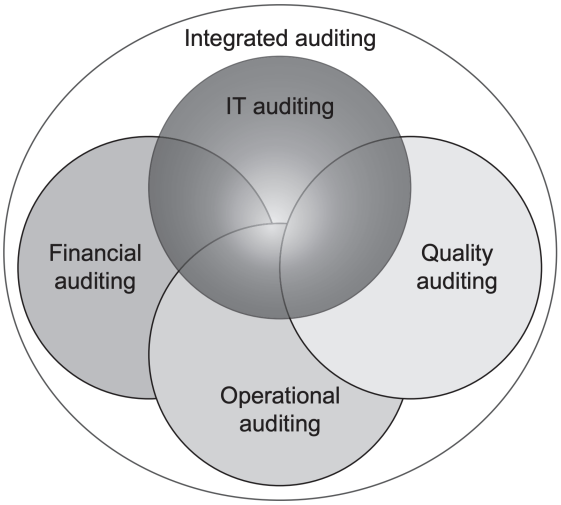

Figure 3.1: IT auditing commonality with other types of audit [Gantz, 2014]

IT auditing has much in common with other types of audit and overlaps in many respects with financial, operational, and quality audit practices. It is important to use "IT" to qualify IT audit and distinguish it from the more common financial connotation of the word audit used alone.

3.2 IT Auditing in Context

From the perspective of planning and performing IT audits, controls represent the substance of auditing activities, as the controls are the items that are examined, tested, analyzed, or otherwise evaluated [Gantz, 2014].

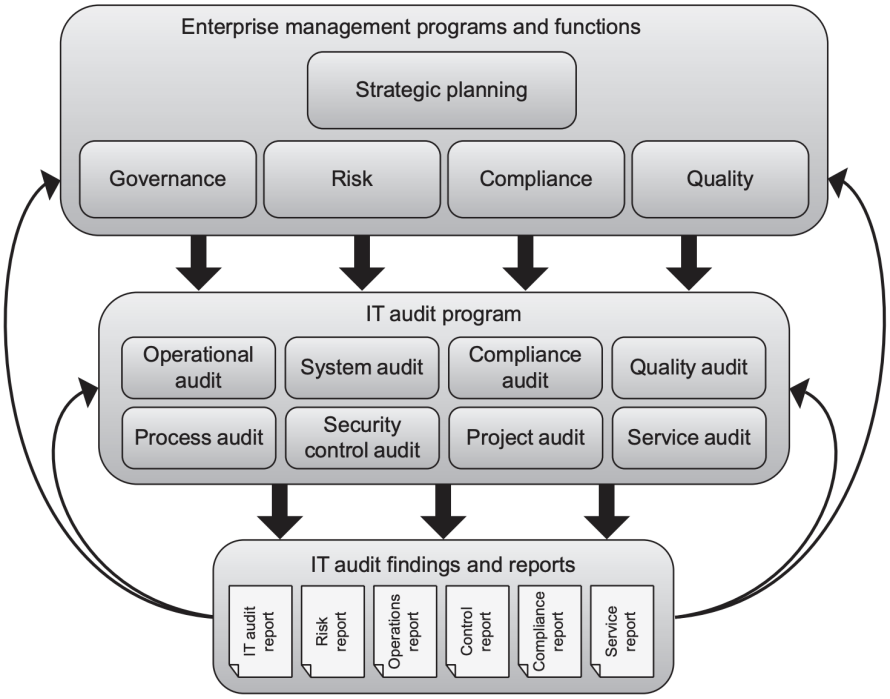

Figure 3.2: IT audit activities and scopes [Gantz, 2014]

IT audits are performed both by internal auditors working for the organization and external auditors hired by it. The processes and procedures followed in internal and external auditing are often quite similar

The following is a one of the wide spread control categorization schemes used in internal control frameworks. Controls are normally classified by purpose, functional type, or both.

	Preventive	Detective	Corrective
Administrative	Acceptable use policy; Security awareness training	Audit log review procedures; IT audit program	Disaster recovery plan; Plan of action and milestones
Technical	Application firewall; Logical access control	Network monitoring; Vulnerability scanning	Incident response center; Data and system backup
Physical	Locked doors and server cabinets; Biometric access control	Video surveillance; Burglar alarm	Alternate processing facility; Sprinkler system

Table 3.1: Examples of Internal Controls Categorized by Type and Purpose [Gantz, 2014]

Administrative controls specify what an organization intends to do to safeguard the integrity of its operations, information, and other assets.

IT audits and the approaches used to conduct them may consider internal controls from multiple perspectives by focusing on different IT elements. For now the focus will lay on audit log reviews and audit logging.

3.3 Audit log

The purpose of audit logging is to record each state change. An audit log is typically used to help customer support, ensure compliance, and detect suspicious behaviors. Each audit log entry records at minimum the identity of the entity, the action performed, and the business object(s) [Richardson, 2018]

Depending on the requirements maintaining and ensuring a comprehensive and immutable audit log is mandatory and might dictates the way a system is architected and developed. An example will be showcased in chapter 6.

The Federal Financial Supervisory Authority (short BaFin in german) requires appropriate precautions to be taken within the framework of application development, so that

the confidentiality, integrity, availability and authenticity of the data to be processed are transparently ensured even after each deployment of an application [BaFin, 2021].

One of the appropriate precautions suggested by BaFin is Audit logs. Audit logging is not only a suggestion but also an indirect requirement:

> In accordance with the target protection requirements the institution must set up processes for logging and monitoring, which make it possible to verify, that authorizations are only used as intended [BaFin, 2021].

An audit log can take many forms. The most common form is a file. A database table is also an option. However most problems comes mainly from the kind of logs written and the way they are processed. More on that is discussed in chapter 5

Audit Log is easy to write but harder to read, especially as it grows large. Occasional ad-hoc reads can be done by eye and simple text processing tools. More complicated or repetitive tasks can be automated [Fowler, 2004]. Audit log entries lack the context and describe an action, that may or may not be related to other entries in a specific period of time. Keeping track of changes without proper automation becomes harder and harder to the point it becomes imposable.

For some organizations logging system changes alone might not be enough. Suspicious activates might originate from read attempts Logging such activities increases the complexity of the audit log and poses the question of how to make sense of such data?

3.4 Auditing 2.0

Auditing 2.0, also known as continuous auditing, is a modern approach to auditing that uses technology and data analytics to continuously monitor and assess an organization's processes. Unlike traditional auditing, which is typically conducted on a periodic basis, continuous auditing is a continuous process that uses, but not limited to, real-time data to identify and address potential risks and issues as they arise. [van der Aalst u. a., 2010]

Auditing 2.0 makes use of technologies such as Artificial Intelligence (AI) to automate and streamline the audit process. For example, AI-powered systems can be used to analyze and interpret data in real-time, while patterns like Event-Sourcing provide a transparent and immutable record of events.

3.4.1 Business Provenance

The systematic, reliable, and trustworthy recording of events, known as business provenance, is essential to auditing in general and Auditing 2.0 in particular. This term acknowledges the importance of traceability by ensuring that history cannot be rewritten or obscured [van der Aalst u. a., 2010].

Traditionally, an audit can only provide reasonable assurance that business processes are executed within the given set of boundaries. Auditors assess the operating effectiveness of process controls, and when these controls are not in place or functioning as expected, they typically check samples of factual data. However, with detailed information about processes increasingly available in high-quality event logs, auditors no longer have to rely on a small set of samples offline. Instead, using process mining techniques, they can evaluate all events in a business process, and do so while it is still running.

3.4.2 Process Mining

The goal of process mining is to discover, monitor, and improve real (not assumed) processes by extracting knowledge from event logs.

Process mining starts with the event log: a sequentially recorded collection of events, each of which refers to an activity (well-defined step) and is related to a particular case (process instance). Some mining techniques use other information such as the person or resource initiating the activity, the event's time stamp, or data elements recorded with the event [van der Aalst u. a., 2010].

Auditors can use process mining techniques to evaluate all events in a business process, and do so while it is still running. With the help of AI potential compliance violation and suspicious behaviour can be be detected while in the making and prevented before even happening. Reliable information is needed to determine whether these processes are executed within certain boundaries set by managers, governments, and other stakeholders.

Process mining techniques as well as an auditing framework as suggested by [van der Aalst u. a., 2010] are addressed and discussed in chapter 5

3.4.3 Challenges

Move to Conclusion or evaluate if this is really relevent?

The main challenge of Auditing 2.0 is the introduced complexity and required skill. Process mining depends on the availability of relevant data, which is Traditionally stored in Enterprise Resource Planning (ERP) systems. While modern pattern like Event-Sourcing offer better integrations to couples detailed event logs with process mining mining ERP systems is challenging because they are not, despite having built-in workflow engines, process-oriented. Because data related to a particular process it usually scattered over dozens of tables, extracting it for auditing is nontrivial [van der Aalst u. a., 2010].

Further more adaption of Auditing 2.0 will lead to an increase in the so-called "auditing materiality". Considering only a small subset of data will not be an option anymore. Optimizing and improving the current audit practices will inevitably lead to more exceptions requiring follow-ups, which increases auditing time and cost.

4 Event Sourcing

Conformance checking If there is an a-priori model, then this model can be used to check if reality, as recorded in the log, conforms to the model and vice versa. For example, there may be a process model indicating that purchase orders of more than one million Euros require two checks. Another example is the checking of the four-eye principle. Conformance checking may be used to detect deviations, to locate and explain these deviations, and to measure the severity of these deviations. An example is the conformance checking algorithm described in (A. Rozinat and W.M.P. van der Aalst. Conformance Checking of Processes Based on Monitoring Real Behavior. Information Systems, 33(1):64-95, 2008).

Provenance data will make it possible to "replay" history reliably and accurately and to predict problems, thereby improving business processes.

Event-Sourcing is a software architecture pattern that insures a complete log of changes made to a system as a series of events. Instead of storing the current state in a traditional database, Event-Sourcing stores the history of changes made over time. This allows developers to rebuild the current state at any point in time by replaying the stored events.

One of the primary benefits of Event-Sourcing is that it provides a complete audit trail. Because the log of events is comprehensive and immutable, it is possible to trace the history of any given piece of data and see exactly how it has changed over time. This is extremely useful for auditing purposes, as it allows organizations to track and monitor changes to sensitive data and ensure that they are in compliance with regulations and standards.

This makes Event-Sourcing particularly well-suited for applications that need to support complex business processes that require the ability to track and audit changes to the data over time without relying on traditional logging mechanisms.

In addition to providing a detailed audit trail, Event-Sourcing also offers a number of other benefits. Since the events are stored in a chronological order, it is possible to implement time-based queries and to reconstruct the state of the system at any point in the past. Beside this being useful for debugging and for performing rollbacks or reversals of changes it lays the base for Auditing 2.0 discussed in section 3.4.

DATA PROTECTION

...[Fowler, 2022]

Wenn mehrere Teil-Kapitel zu strukturieren sind: Schreiben Sie zu jedem Teil-Kapitel eine Ein- leitung ("Hier wird die folgende Fragestellung untersucht...") und eine Ausleitung ("Hiermit ist erreicht: ... Die folgenden Probleme sind aber noch offen:...").

Event Sourcing is a pattern for storing data as events in an append-only log. This simple definition misses the fact that by storing the events, you also keep the context of the events; you know an invoice was sent and for what reason from the same piece of information. In other storage patterns, the business operation context is usually lost, or sometimes stored elsewhere.

Event driven, CQRS and Evnet Sourcing are complementary patterns so it makes sense to consider them both as a set and individually. This article is the latter - a focus on event sourcing.

Martin Fowler believes that the key to Event Sourcing is that it guarantees that all changes to the domain objects are initiated by the event objects. This leads to a number of facilities that can be built on top of the event log:

Complete Rebuild: we can discard the application state completely and rebuild it by re-running the events from the event log on an empty application. Temporal Query: we can determine the application state at any point in time. Notionally we do this by starting with a blank state and rerunning the events up to a particular time or event. Event Replay: if we find a past event was incorrect, we can compute the consequences by reversing it and later events and then replaying the new event and later events.

One obvious form of return is that it's easy to serialize the events to make an Audit Log.Such an audit trail is useful for audit, no shocks there, but also has other usages. I chatted with someone who got their online accounts into an awkward state and phoned in for help. He was impressed that the helper was able to tell him exactly what he did and thus was able to figure out how to fix it. To provide such a capability means exposing

the audit trail to the support group so they can walk through a user's interaction. While Event Sourcing is a good way of doing this, you could also do this with more regular logging mechanisms, and that way not have to deal with the odd interface.

however, logging is mostly associated with debugging and has no direct relation to the system state. Logging style, verbosity....

Event Sourcing is a design approach where distributed applications keep their state as a sequence of state-changing operations. Instead of storing mutable objects, applications keep an immutable sequence of changes to such objects. Changing state and writing an event to the log is one single, therefore atomic, operation. The log becomes the authoritative source of truth, offering eventual consistency, as events propagate to different parts of the distributed application. This design entails a strongly decoupled architecture, typical of publish–subscribe systems (Clayman et al., 2010), while providing reliable auditing and logging functionality. For example, developers may bring the application back to a previous execution state, by replaying the events in the log. This feature is particularly powerful, not only for the sake of failure recovery and state management (most services can be stateless), but also for debugging and to experiment alternative what-if scenarios. Furthermore, it makes the system's data schema more flexible, as it becomes possible to recover field values or even calculate additional ones from the log.

The foundational idea of event sourcing is the domain event as described by Evans (2015). His seminal book on Domain-Driven Design (DDD), however, does not mention the pattern. Vernon (2013) describes event sourcing only briefly in his book on the implementation of various DDD patterns. Young (2017), as one of the original proposers of event sourcing, discusses the challenge of versioning ESSs. Event sourcing is also discussed in the context of CQRS (Young, 2010), a pattern strongly related to event sourcing. Recent academic literature (Erb, 2019, Zhong et al., 2019) shows an interest in applying event sourcing for research projects.

Recently, the event sourcing pattern has become a popular answer to the challenges of complex, mission-critical, scalable systems. Examples of organizations that apply event sourcing are Netflix (Avery and Reta, 2017), and Walmart's Jet.com (Gorodinski, 2017), with the goal of creating scalable and reliable critical systems. Event sourcing is informally described by Fowler (2005) as a pattern that "ensures that all changes to application state are stored as a sequence of events". Flexibility, debug-ability, and reliability are given by Avery and Reta (2017) as rationale for using event sourcing. Debski et al. (2017) and Erb and Hauck (2016) show how event sourcing can be applied

to achieve scalable, reactive systems. Kabbedijk et al. (2012) describes event sourcing as a sub pattern of Command Query Responsibility Segregation (CQRS) in his work on the improved variability and scalability of systems applying CQRS.

The events in event sourcing, as opposed to general event-driven architectures (EDAs) (Fowler, 2017), are stored as an append-only log of all state changes. Two key characteristics separate event sourcing from event-driven approaches, such as stream processing, transactional processing, and blockchain. First, events in Event Sourced Systems (ESSs) are stored as the state of the application. Other approaches use the events to communicate, while the communication aspect comes second in ESSs. The second difference is that events are closely related to events occurring in real world business processes. This allows event sourcing to be also used as a design approach. Domain-Driven Design (DDD), as described by Evans (2003), advocates events as a design tool for the process flow of a software system. Brandolini (2018) proposes event storming (analogous to brainstorming), a group design process that focuses on the events that take place in a software system. Further details on these analogous approaches are found in Section 3.

Our study regards a new research area, therefore, we apply Grounded Theory (GT). Adolph et al. (2011) describe GT as a useful approach for research in areas that have not previously been studied. A GT explains how people resolve their main concern by employing a certain process. That process is called the 'core category' of the GT. The core category of the work presented in this article is the process of designing and implementing event sourced systems, as performed by software engineers. The theoretical definition of event sourcing helps both researchers and practitioners to understand, reason about, and teach the pattern and its consequences. Section 2 explains how we applied GT to form a basis for conceptualization of ESSs from 25 interviews, and how the three essential elements are covered. From the gathered data we distill the pattern description and its consequences. This work has the following contributions:

the studie contudcted by ..... on 25 engineers of different backgrounds and roles by applying appling Grounded Theory (GT). Adolph et al. (2011) which showcase how event sourcing is ganging on popularity especially when it comes to satisfying auditory needs. As put by one of the engineers when asked about event sourcing "" The reasons for applying event sourcing can be grouped into four categories. Remarkably, all systems under study benefit from event sourcing, and no system returned to a current state model. Still, most engineers state that they would not apply event sourcing in every system. The reason given for this opinion is the added complexity of introducing event

sourcing. Engineer E2 would apply event sourcing by default, because of the benefits it gives. [Overeem u. a., 2021]

Together with this description we identify four categories of rationale for the application of event sourcing, such as a decrease of complexity. In this "In Practice" submission, we also identify five engineering challenges around the pattern, with schema evolution being one of the most complex challenges. With the pattern description and its liabilities presented in this article, we enable engineers to make a considered choice.

Event sourcing is an approach to store data in an application and was originally established by Fowler (2005a). This differs from the traditional approach of storing data which, is to store the current state of an application in a database. Fowler (2002) describes this approach as active records. With the traditional approach, the data is accessed or modified by four operations, Create, Read, Update and Delete, these operations are generally referred to as CRUD operations (Betts et al. 2013). The idea behind event sourcing is to treat each change to the state as an event where each event is part of a sequence of events in the order they occurred. The sequence of events can be used to recreate the application state at any point in time (Fowler 2005a). Figure 1 illustrates how the state of two shapes can be represented with active records and with event sourcing. In Figure 1a the current X and Y values, together with the type of shape, are stored in a specific row associated with the shape identification number. In Figure 1b the same shapes are stored as a series of events. Each row consists of the type of event and the parameters for the event, together with an aggregate number which describes to which shape the event is associated. The series of events can be used to rebuild the current state which would result in the same state as in Figure 1a. One difference between the two approaches is that in Figure 1b it is possible to see that shape number 1 has moved from the original position which is information is lost when using active records.

4.1 Terminology

CQRS and event driven

4.2 Command Query Responsibility Segregation

...[Young, 2010]

One of the architectural patterns that in recent years emerged in the development of cloud systems is Command Query Responsibility Segregation (CQRS). The pattern was introduced by Young [5] and Dahan [6], and the goal of the pattern is to handle actions that change data (those are called commands) in different parts in the system than requests that ask for data (called queries). By separating the command-side (the part that validates and accepts changes) from the query-side (the part that answers queries), the system can optimize the two parts for their very different tasks.

Young [7] describes CQRS as a stepping stone for event sourcing. Event sourcing is a data storage model that does not store the current (or last) state, but all changes leading up to the current state. Fowler [8] explains event sourcing by comparing it to an audit trail: every data change is stored without removing or changing earlier events. The events stored in an event store are stored as schema-less data, because the different events often do not share properties. A store with an explicit schema would make it more difficult to append events in the store to a single stream. Data in schema-less stores is not without schema, but the schema is implicit: the application assumes a certain schema. This makes the problem of schema evolution and data conversion more difficult as observed by Scherzinger et al. [9]. Schema-less data is more difficult to evolve as the store is unaware of structure and thus cannot offer tools to transform the data into a new structure. Relational data stores that have explicit knowledge of the structure of the data can use the standardized data definition language (DDL) to upgrade the schema and convert the data. Another problem in the evolution of event sourced systems is the amount of data that is stored, not only the current state, but also every change leading up to that state. This huge amount of data makes the problem of performing a seamless upgrade even more important: upgrades may need more time, but they are required to be imperceptible.

The foundations of CQRS were laid by Meyer [11] in the Command-Query Separation (CQS) principle. He defined a command as “serving to modify objects” and a query is “to return information about objects”, or informally worded “asking a question should not change the answer”. Figure 1 shows the CQRS pattern: commands are accepted by the command-side and produce events which are processed by the query-side. The query-side projects these events into a form that is suitable for querying and presenting. The command-side and the query-side both have their own data store: the first store is used to maintain data that is used in validating requested changes, and the second store is used to retrieve data for displaying or reporting.

figure to showcase CQRS

The command-side communicates with the query-side through asynchronously sending events. These events are used by the query-side to build a view of the state that can be used to query and present data. By doing this asynchronously the query-side does not influence the performance of the command-side. However, this does lead to eventual consistency. This is a weaker form of consistency that Vogels [12] defines as "when no updates are made to the object, the object will eventually have the last updated value". The system guarantees that the query-side eventually will reflect the events produced in the command-side. However, there are no guarantees on how fast this will be done. A system with a large delay is unfeasible, because in that case queries will often return data that does not reflect the latest changes send to the command-side. There are difficulties introduced by eventual consistency, such as returning items to a client that in fact are already deleted through commands send to the command-side. The patterns to overcome this difficulty and others are out of scope for the current paper.

The asynchronous sending of events between the command-side and query-side results in a weak coupling. The resulting freedom and flexibility in designing the system leads to availability, scalability, and performance among other advantages. The store used in the command-side is often an event store, because it is natural to store the events that are produced by the command-side. This proposed data storage model has a number of benefits that make it specifically useful as a store for the command-side of a CQRS system. First of all, the command-side is only used for accepting changes and never for queries, and the performance of the store is not thus not hampered by concurring reads and writes. Second, the store contains every change ever accepted into the system, making it easy to inspect when and by whom a change was done. A third benefit is the possibility to rebuild the current state (for instance the query-store) in the system by replaying the events. The replaying of events also enables easy debugging. The fourth benefit is the possibility to analyze the events for patterns in usage. This information is impossible to extract from a store that only persists the last state of the data. In the query-side a diverse range of stores can be used, such as relational, graph, or NoSql databases. The main goal of this store is to support the easy and fast retrieval of data, in whatever form the application requires.

The loosely coupled nature of CQRS combined the benefits of the event sourcing approach makes it a fitting architectural pattern for cloud systems. Event sourcing itself is not tied exclusively to CQRS, the coupling based on events is similar to that in more general

event-driven architectures, as described by Michelson [13]. The events in the event store are processed by the system to build the query-side or execute complex processes. The CQRS pattern and its sub-patterns are described in more detail by Kabbedijk et al. [14]. CQRS from a practitioners viewpoint is studied by Korkmaz [15] in order to gain better understanding of the benefits and challenges. Maddodi et al. [16] studies a CQRS system in the context of continuous performance testing.

An audit log is the simplest, yet also one of the most effective forms of tracking temporal information. The idea is that any time something significant happens you write some record indicating what happened and when it happened.

4.3 Event-Sourced Architecture

Event Sourcing is the foundation for Parallel Models or Retroactive Events. If you want to use either of those patterns you will need to use Event Sourcing first. Indeed this goes to the extent that it's very hard to retrofit these patterns onto a system that wasn't built with Event Sourcing. Thus if you think there's a reasonable chance that the system will need these patterns later it's wise to build Event Sourcing now. This does seem to be one of those cases where it isn't wise to leave this decision to later refactoring.

Event Sourcing also raises some possibilities for your overall architecture, particularly if you are looking for something that is very scalable. There is a fair amount of interest in 'event-driven architecture' these days. This term covers a fair range of ideas, but most of centers around systems communicating through event messages. Such systems can operate in a very loosely coupled parallel style which provides excellent horizontal scalability and resilience to systems failure.

An example of this would be a system with lots of readers and a few writers. Using Event Sourcing this could be delivered as a cluster of systems with in-memory databases, kept up to date with each other through a stream of events. If updates are needed, they can be routed to a single master system (or a tighter cluster of servers around a single database or message queue) which applies the updates to the system of record and then broadcasts the resulting events to the wider cluster of readers. Even when the system of record is the application state in a database this could be a very appealing structure. If the system of record is the event log, there is are plenty of options for very high performance since the event log is a purely additive structure that requires minimal locking.

Such an architecture isn't flawless, of course. The reader systems are liable to be out of sync with the master (and each other) due to differences in timing with event propagation. However this broad style of architecture is used and I've heard almost entirely favorable comment about it.

Using event streams like this also allows new applications to be added easily by tapping into the event streams and populating their own models, which don't need to be the same for all systems. It's an approach that fits in very well with a messaging approach to integration.

comparison table why event-sourced/driven [Richards, 2015]

4.3.1 Domain Event

A domain event is a class with a name formed using a past-participle verb. It has properties that meaningfully convey the event. Each property is either a primitive value or a value object. For example, an OrderCreated event class has an orderId property. A domain event typically also has metadata, such as the event ID, and a timestamp. It might also have the identity of the user who made the change, because that's useful for auditing. The metadata can be part of the event object, perhaps defined in a superclass. Alternatively, the event metadata can be in an envelope object that wraps the event object. The ID of the aggregate that emitted the event might also be part of the envelope rather than an explicit event property. The OrderCreated event is an example of a domain event. It doesn't have any fields, because the Order's ID is part of the event envelope. The following listing shows the OrderCreated event class and the DomainEventEnvelope class. [Richardson, 2018]

4.4 Challenges

However, an ESS introduces not only events and state transfers. Eventual consistency forces developers to let go of guarantees that they would have in a system using current state and synchronous processing. In a CQRS system, an update sent through a command will not immediately be reflected in the result of a query. The system first needs to process the event into one or more projections. Engineer E12 states that "a lot of developers had to get used to information not being in place", and E2 adds that "getting people

to understand eventual consistency is the biggest hurdle". Eventual consistency forces developers to rethink the basic interactions of the user with the system.

We give two examples of interactions that force developers to rethink system design. The first example is that of the expectation of users to retrieve data that they previously submitted into the system. However, in a CQRS system, the query system might not directly return the data that was submitted through a command. The user interface of the system should make it clear to the user what is going on, or even try to hide the fact that the system is eventual consistent. The second example is that of developers that more or less have the same expectation. Often developers try to use the result of the query to make decisions in an aggregate. However, the query system might not have processed all events and misses recent updates. If developers overlook this principle, the decisions lead to bugs in the system.

4.4.1 Event storage

Event-sourcing enables the reconstruction of arbitrary past application states for event-based applications. However, an entirely unbounded log size can conflict with other application requirements. We provided a first exploration of log pruning approaches for event-sourced systems and evaluated their effects as the main contributions of this work. This included an overview of the design space of pruning approaches and an assessment of the impact of log pruning mechanisms on state reconstructability. Unbounded command and event sourcing approaches enable a full reconstruction of previous states, but come with high costs in terms of storage. In practice, pure command sourcing is often not feasible due to complex re-computations. Bounded approaches restrict the log sizes, but enable reconstructions primarily for more recent states. Additional periodic checkpoints can help to maintain some older states as reference. The probabilistic approach provides a configurable bias to determine what to prune. Eventually, the choice of a log pruning approach has to reflect the application-level requirements — which entity states should be kept, how far the application needs to go back in time, and in which resolution. Application-based and user-defined pruning mechanisms as well as a detailed analysis of pruning parameters constitute future work.

some math from https://dl.acm.org/doi/10.1145/3210284.3219767 (https://dl.acm.org/doi/pdf/10.1145/321

4.4.2 Event schema evolution

Table 6.1 shows the different types of changes that can occur at each level. [Richardson, 2018]

With event sourcing, the schema of events (and snapshots!) will evolve over time. Because events are stored forever, aggregates potentially need to fold events corresponding to multiple schema versions. There's a real risk that aggregates may become bloated with code to deal with all the different versions. As mentioned in section 6.1.7, a good solution to this problem is to upgrade events to the latest version when they're loaded from the event store. This approach separates the code that upgrades events from the aggregate, which simp

4.4.3 Deleting data is tricky.

Because one of the goals of event sourcing is to preserve the history of aggregates, it intentionally stores data forever. The traditional way to delete data when using event sourcing is to do a soft delete. An application deletes an aggregate by setting a deleted flag. The aggregate will typically emit a Deleted event, which notifies any interested consumers. Any code that accesses that aggregate can check the flag and act accordingly. Using a soft delete works well for many kinds of data. One challenge, however, is complying with the General Data Protection Regulation (GDPR), a European data protection and privacy regulation that grants individuals the right to erasure (https://gdpr-info.eu/art-17-gdpr/). An application must have the ability to forget a user's personal information, such as their email address. The issue with an event sourcing-based application is that the email address might either be stored in an AccountCreated event or used as the primary key of an aggregate. The application somehow must forget about the user without deleting the events. Encryption is one mechanism you can use to solve this problem. Each user has an encryption key, which is stored in a separate database table. The application uses that encryption key to encrypt any events containing the user's personal information before storing them in an event store. When a user requests to be erased, the application deletes the encryption key record from the database table. The user's personal information is effectively deleted, because the events can no longer be decrypted. Encrypting events solves most problems with erasing a user's personal information. But if some aspect of a user's personal information, such as email address, is used as an aggregate ID, throwing away the encryption key may not be sufficient.

For example, section 6.2 describes an event store that has an entities table whose primary key is the aggregate ID. One solution to this problem is to use the technique of pseudonymization, replacing the email address with a UUID token and using that as the aggregate ID. The application stores the association between the UUID token and the email address in a database table. When a user requests to be erased, the application deletes the row for their email address from that table. This prevents the application from mapping the UUID back to the email address. [Richardson, 2018]

4.4.4 Querying the event store is challenging.

Imagine you need to find customers who have exhausted their credit limit. Because there isn't a column containing the credit, you can't write SELECT * FROM CUSTOMER WHERE CREDIT_LIMIT = 0. Instead, you must use a more complex and potentially inefficient query that has a nested SELECT to compute the credit limit by folding events that set the initial credit and adjusting it. To make matters worse, a NoSQL-based event store will typically only support primary key-based lookup. Consequently, you must implement queries using the CQRS approach described in chapter 7. [Richardson, 2018]

4.4.5 Transactions

https://www.youtube.com/watch?v=RGqr1cXjS5o

4.5 Benefits

Event sourcing can be used to implement Auditing 2.0 in a number of ways. Here are a few examples:

Storing a log of events: As mentioned earlier, event sourcing involves storing a log of events that represent changes made to the data over time. This log can be used to reconstruct the state of the data at any point in the past, which can be useful for auditing purposes. By storing a comprehensive and immutable log of events, organizations can use event sourcing to track and monitor changes to sensitive data and ensure compliance with regulations and standards.

Analyzing event data: In addition to storing the log of events, organizations can also use data analytics and machine learning techniques to analyze the event data and identify patterns and trends that may indicate potential risks or issues. By continuously analyzing the event data, organizations can proactively identify and address potential problems as they arise, rather than waiting for them to be discovered during a periodic audit.

Automating the audit process: Event sourcing can also be used to automate the audit process by using technologies such as artificial intelligence and blockchain. For example, AI-powered systems can be used to analyze the event data and identify potential risks or issues, while blockchain technology can be used to provide a secure and transparent record of transactions. This can help to streamline the audit process and make it more efficient.

Overall, event sourcing can be a powerful tool for implementing Auditing 2.0. By storing a comprehensive log of events and using advanced technologies and data analytics to analyze the data, organizations can continuously monitor and assess their financial and operational processes, identify potential risks and issues, and take timely and appropriate action to address them.

Event sourcing is related to database systems techniques used for persistence guarantees and replication. Gray and Reuter (1992) describe how transaction logs can be used to replicate state between database systems. Every state change is recorded as a transaction, which is similar to event sourcing where every state change is recorded as an event. Kleppmann (2017) discusses event sourcing in the context of data-intensive applications, he relates the pattern to the change data capture approach, often used in Extract-Transform-Load (or ETL) processes (Vassiliadis, 2009). ETL solutions are often used for creating data warehouses. The primary difference between event sourcing and these techniques is that a transaction or a data change is a technical entity without relation to the real world, while an event in event sourcing resembles an event in the real world.

Event sourcing is a pattern that stores every state change, immutability is thus at the core of the pattern. Helland (2015) states that immutability of data is a crucial aspect for distributed systems. Although often seen as the defining characteristic of event sourcing, immutability is not enforced in any manner, as opposed to a blockchain. In a number of the systems under study, immutability is sacrificed for a simpler schema evolution technique (see Section 7). We observed different degrees of immutability. The first degree is strict, 8 out of the 19 ESSs never change an event. The second degree of

immutability is used by 3 out of 19 systems, which allow for cut-off moments. In such a cut-off moment, the event store is changed, but back-ups guarantee that no information is deleted. The goal of these back-ups is to satisfy regulations or service-level agreements, therefore, they are kept around forever. This degree of immutability still guarantees an audit trail, because the back-ups can be used to retrieve all the state changes. The last degree level of immutability is mutable, 8 out of 19 systems allow events to change. In these systems, the event store is changed on some occasions, and the back-ups are not kept forever. These systems do not satisfy the goal of a complete audit trail. However, the events can still be used to explain how the current state was reached. None of the ESSs lose information regarding the current state of a system. Events that are changed, or transformed, are in most cases changed because of technical reasons.

How can a System that Uses Event Sourcing Protect User Privacy? — Privacy regulations, such as the GDPR, are designed to protect users from being taken advantage of. Personal information should not be kept in a system for all eternity, but the system should delete it whenever someone requests that. However, such a requirement conflicts with the nature of event sourcing: retaining all the data. Engineers E20, E21, E23, E25 mention that they designed their systems to comply with these regulations. Systems HealthSys and P-PaySys use some form of anonymization and removal of information to comply. Obviously, this requires them to rewrite events. System IdentitySys takes a completely different approach. The system separates the events and the personal information in two different stores. When events are read, they are supplemented with the personal information. If that information is no longer present (because of removal requests), default values are supplied.

Easy temporal queries - Because event sourcing maintains the complete history of each business object, implementing temporal queries and reconstructing the historical state of an entity is straightforward.

5 Software Architecture and Auditing

Design patterns

Wenn mehrere Teil-Kapitel zu strukturieren sind: Schreiben Sie zu jedem Teil-Kapitel eine Ein- leitung ("Hier wird die folgende Fragestellung untersucht...") und eine Ausleitung ("Hiermit ist erreicht: ... Die folgenden Probleme sind aber noch offen:...").

Software architecture is the structure, or set of structures, which comprises software elements, the externally visible properties of those elements, and the relationships among them [Bass u. a., 2003]. This structure is an artifact from a software development process and is represented by a document composed by one or more models, which represent different perspectives about how the system will be structured, and information sets that facilitate the understanding of the proposed computational solution. It is defined based on the software requirements. Among the different types of requirements, the quality requirements are the most important for the specification of an architecture since it exerts considerable influence over it structure [Bass u. a., 2003].

5.1 Implementing audit logging

There are a few different ways to implement audit logging: [Richardson, 2018]

When you use Audit Log you should always consider writing out both the actual and record dates. They are easy to produce and even though they may be the same 99% of the time, the 1% can save your bacon. As you do this remember that the record date is always the current processing date. [Fowler, 2004]

Taking a more related example where users are saved in a database and one might get the impression that user X also created

Audit logging—Log user actions.

The glory of Audit Log is its simplicity. As you compare Audit Log to other patterns such as Temporal Property and Temporal Object you quickly realize that these alternatives add a lot of complexity to an object model, although these are both often better at hiding that complexity than using Effectivity everywhere.

But it's the difficulty of processing Audit Log that is it's limitation. If you are producing bills every week based on combinations of historic data, then all the code to churn through the logs will be slow and difficult to maintain. So it all depends how tightly the accessing of temporal information is integrated into your regular software process. The tighter the integration, the less useful is Audit Log.

Remember that you can use Audit Log in some parts of the model and other patterns elsewhere. You can also use Audit Log for one dimension of time and a different pattern for another dimension. So you might handle actual time history of a property with Temporal Property and use Audit Log to handle the record history.

Audit Log is easy to write but harder to read, especially as it grows large. Occasional ad hoc reads can be done by eye and simple text processing tools. More complicated or repetitive tasks can be automated with scripts. Many scripting languages are well suited to churning though text files. If you use a database table you can save SQL scripts to get at the information.

Provenance data will make it possible to "replay" history reliably and accurately and to predict problems, thereby improving business processes.

5.1.1 Audit logging code in business logic

The first and most straightforward option is to sprinkle audit logging code throughout your service's business logic. Each service method, for example, can create an audit log entry and save it in the database. The drawback with this approach is that it intertwines auditing logging code and business logic, which reduces maintainability. The other drawback is that it's potentially error prone, because it relies on the developer writing audit logging code.

5.1.2 Aspect-Oriented programming

The second option is to use AOP. You can use an AOP framework, such as Spring AOP, to define advice that automatically intercepts each service method call and persists an audit log entry. This is a much more reliable approach, because it automatically records every service method invocation. The main drawback of using AOP is that the advice only has access to the method name and its arguments, so it might be challenging to determine the business object being acted upon and generate a businessoriented audit log entry.

5.1.3 Event Sourcing

The third and final option is to implement your business logic using event sourcing. As mentioned in chapter 6, event sourcing automatically provides an audit log for create and update operations. You need to record the identity of the user in each event. One limitation with using event sourcing, though, is that it doesn't record queries. If your service must create log entries for queries, then you'll have to use one of the other options as well.

5.2 traditional persistence

The traditional approach to persistence maps classes to database tables, fields of those classes to table columns, and instances of those classes to rows in those tables. For example, figure 6.1 shows how the Order aggregate, described in chapter 5, is mapped to the ORDER table. Its OrderLineItems are mapped to the ORDER_LINE_ITEM table.

The application persists an order instance as rows in the ORDER and ORDER_LINE_-ITEM tables. It might do that using an ORM framework such as JPA or a lower-level framework such as MyBATIS. This approach clearly works well because most enterprise applications store data this way. But it has several drawbacks and limitations: - Object-Relational impedance mismatch. - Lack of aggregate history. - Implementing audit logging is tedious and error prone. - Event publishing is bolted on to the business logic. Let's look at each of these problems, starting with the Object-Relational impedance mismatch problem. [Richardson, 2018]

5.2.1 Problems

OBJECT-RELATIONAL IMPEDANCE MISMATCH

One age-old problem is the so-called Object-Relational impedance mismatch problem. There's a fundamental conceptual mismatch between the tabular relational schema and the graph structure of a rich domain model with its complex relationships. Some aspects of this problem are reflected in polarized debates over the suitability of Object/Relational mapping (ORM) frameworks. For example, Ted Neward has said that "Object-Relational mapping is the Vietnam of Computer Science" (http://blogs .tedneward.com/post/the-vietnam-of-computer-science/). To be fair, I've used Hibernate successfully to develop applications where the database schema has been derived from the object model. But the problems are deeper than the limitations of any particular ORM framework.

LACK OF AGGREGATE HISTORY

Another limitation of traditional persistence is that it only stores the current state of an aggregate. Once an aggregate has been updated, its previous state is lost. If an application must preserve the history of an aggregate, perhaps for regulatory purposes, then developers must implement this mechanism themselves. It is time consuming to implement an aggregate history mechanism and involves duplicating code that must be synchronized with the business logic.

implementing audit logging is tedious and error prone

Another issue is audit logging. Many applications must maintain an audit log that tracks which users have changed an aggregate. Some applications require auditing for security or regulatory purposes. In other applications, the history of user actions is an important feature. For example, issue trackers and task-management applications such as Asana and JIRA display the history of changes to tasks and issues. The challenge of implementing auditing is that besides being a time-consuming chore, the auditing logging code and the business logic can diverge, resulting in bugs.

event publishing is bolted on to the business logic

Another limitation of traditional persistence is that it usually doesn't support publishing domain events. Domain events, discussed in chapter 5, are events that are published by an aggregate when its state changes. They're a useful mechanism for synchronizing

data and sending notifications in microservice architecture. Some ORM frameworks, such as Hibernate, can invoke application-provided callbacks when data objects change. But there's no support for automatically publishing messages as part of the transaction that updates the data. Consequently, as with history and auditing, developers must bolt on event-generation logic, which risks not being synchronized with the business logic. Fortunately, there's a solution to these issues: event sourcing

5.3 Event Sourcing

Event sourcing is an architecture pattern that involves storing the history of events that have occurred in a system as a sequence of records. This allows the system to reconstruct past states and to track changes over time.

One way in which event sourcing can be used for auditing is by providing a complete record of all events that have occurred in the system, including information about when the events occurred and who was responsible for them. This can be useful for identifying and analyzing trends, identifying patterns of behavior, and reconstructing past states of the system.

There are several other architecture patterns that can also be used for auditing, including:

Command and Query Responsibility Segregation (CQRS): This pattern involves separating the responsibilities of reading and writing data, allowing for better scalability and security. CQRS can be used to maintain a separate audit log of all write operations, which can be used for auditing purposes.

Change Data Capture (CDC): This pattern involves capturing and storing changes to data as they occur, allowing for real-time analytics and data integration. CDC can be used to track changes to data over time, which can be useful for auditing purposes.

Two-Phase Commit (2PC): This pattern involves coordinating the execution of transactions across multiple systems, ensuring that either all or none of the changes are made. 2PC can be used to maintain a record of all transactions that have been committed, which can be useful for auditing purposes.

Ultimately, the choice of architecture pattern for auditing will depend on the specific needs of the system and the requirements of the audit process.

100% accurate audit logging - Auditing functionality is often added as an afterthought, resulting in an inherent risk of incompleteness. With event sourcing, each state change corresponds to one or more events, providing 100% accurate audit logging. [Richardson, 2018]

5.3.1 Database Driven

https://eventuate.io/whyeventsourcing.html

5.4 Blockchain

Anh et al. (2018) describes another append-only data structure: blockchain. While the data structure is similar to event sourcing, the goals of the two techniques are different. A blockchain focuses on solving problems related to distribution, consensus, and trust, while event sourcing solves problems with history, temporal complexity, and audit trails. The blockchain approach enforces the immutability of the data to solve its problems, while in event sourcing this immutability is self imposed. Event sourced systems could be build using a blockchain solution. However, the distribution and consensus features offered by blockchain do not improve the goals targeted by event sourcing.

5.5 Auditing 2.0

Auditors can utilise process mining techniques to address multiple use-cases/process mining techniques like passive auditing process discovery, Conformance checking, model extension, etc. active auditing one can “replay” a running case on the process model at real-time and check whether the observed behavior fits. The moment the case deviates, an appropriate actor can be alerted. The process model based on historic data can also be used to make predictions for running cases, e.g., it is possible to estimate the remaining processing time and the probability of a particular outcome. . Similarly, this information can be used to provide recommendations, e.g., proposing the activity that will minimize the expected costs and completion time.

5.5.1 Auditing 2.0 Framework

This will be further discussed in 4

The presence of event logs and process mining techniques enables new forms of auditing. Rather than sampling a small set of cases, the whole process and all of its instances can be considered. Moreover, this can be done continuously.

WHY

Performing and supporting IT audits and managing an IT audit program are time-, effort-, and personnel-intensive activities, so in an age of cost-consciousness and competition for resources, it is reasonable to ask why organizations undertake IT auditing. The rationale for external audits is often clearer and easier to understand— publicly traded companies and organizations in many industries are subject to legal and regulatory requirements, compliance with which is often determined through an audit. Similarly, organizations seeking or having achieved various certifications for process or service quality, maturity, or control implementation and effectiveness typically must undergo certification audits by independent auditors. IT audits often provide information that helps organizations manage risk, confirm efficient allocation of IT-related resources, and achieve other IT and business objectives. Reasons used to justify internal IT audits may be more varied across organizations, but include:

- -

Further details on organizational motivation for conducting internal and external IT audits appear in Chapters 3 and 4, respectively. To generalize, internal IT auditing is often driven by organizational requirements for IT governance, risk management, or quality assurance, any of which may be used to determine what needs to be audited and how to prioritize IT audit activities. External IT auditing is more often driven by a need or desire to demonstrate compliance with externally imposed standards, regulations, or requirements applicable to the type of organization, industry, or operating environment.

IT auditing helps organizations understand, assess, and improve their use of controls to safeguard IT, measure and correct performance, and achieve objectives and intended outcomes. IT auditing consists of the use of formal audit methodologies to examine IT-specific processes, capabilities, and assets and their role in enabling an organization's business processes. IT auditing also addresses IT components or capabilities that support

other domains subject to auditing, such as financial management and accounting, operational performance, quality assurance, and governance, risk management, and compliance (GRC).

6 Audit Component

Wenn mehrere Teil-Kapitel zu strukturieren sind: Schreiben Sie zu jedem Teil-Kapitel eine Ein- leitung ("Hier wird die folgende Fragestellung untersucht...") und eine Ausleitung ("Hiermit ist erreicht: ... Die folgenden Probleme sind aber noch offen:...").

The event sourced system [Overeem u. a., 2021]

Auditing Framework [van der Aalst u. a., 2010] Event logs and process mining techniques enable new forms of auditing. Rather than sampling a small set of cases, auditors can consider the whole process and all of its instances. Moreover, they can do this continuously.

Figure 1 shows an Auditing 2.0 framework based on process mining. “Current data” events are cases that are still running, while “Historic data” events are completed cases. The figure also shows two types of process models: De jure models describe a desired or required way of working, while de facto models aim to describe reality with potential violations of the boundaries defined in de jure models (W. M. P. van der Aalst et al., “Conceptual Model for On Line Auditing,” tech. report BPM-09-19 BPMcenter.org, 2009).

6.1 Monoskope

m8 implements the management and operation of tenants, users and their roles in a Kubernetes multi-cluster environment. It fulfills the needs of operators of the clusters as well as the needs of developers using the cloud infrastructure provided by the operators.

rewrite m8 overview

m8 is an event-sourced system...

TODO

6.2 Requirements

clreaify how this should be written? meeting, disscussion parties, etc...

m8 has by nature a full audit log of every change to the system.This should be utilised to provide auditors and operators the ability to get detailed information of who is allowed to do what and why to answer questions like:

- How did a user get a specific role?
- How did a user become a tenant member?
- What actions were taken by a user?
- ...

Auditors have different backgrounds and technical knowledge thus all events musst have a human-readable representation.

add chapter refrence

It should be possible to utilise Event-Sourcing's temporal quires to get a system overview at a specific date and time

... [Bass u. a., 2003]

6.2.1 Use-Cases

From the requirements overview the following use-cases are derived

Table 6.1: Audit component derived use-cases

ID	Use-Case	Description
UC01	get audit-log for date-range	As an auditor, I want to get all actions taken within a specific date-range
UC02	get audit-log about a user	As an auditor, I want to get all actions taken on a user
UC03	get audit-log of user-actions	As an auditor, I want to get all actions taken by a user
UC04	get audit-log users overview	As an auditor, I want to get an overview of all users at a specific timestamp, tenants they belongs to, and their roles within the system or tenants/clusters

6.2.2 Architectural Constraints

Technical Constraints

Table 6.2: Audit component technical constraints

ID	Constraint	Description
TC01	human-readable representation	log structs musst contain a human-readable representation
TC02	Programming language	m8 is written in GO. No reason to use other language
TC03	Middleware	m8 uses gRPC. No reason to use or support any other framework

Organisational Constraints

Table 6.3: Audit component organisational constraints

ID	Constraint	Description
OC01	Deadline	implementation musst be finalised before 31.01.2023
OC02	The Twelve-Faktor App	adhere to the The Twelve-Faktor App methodology

6.3 System Design

6.3.1 Scope and Context

Business Context

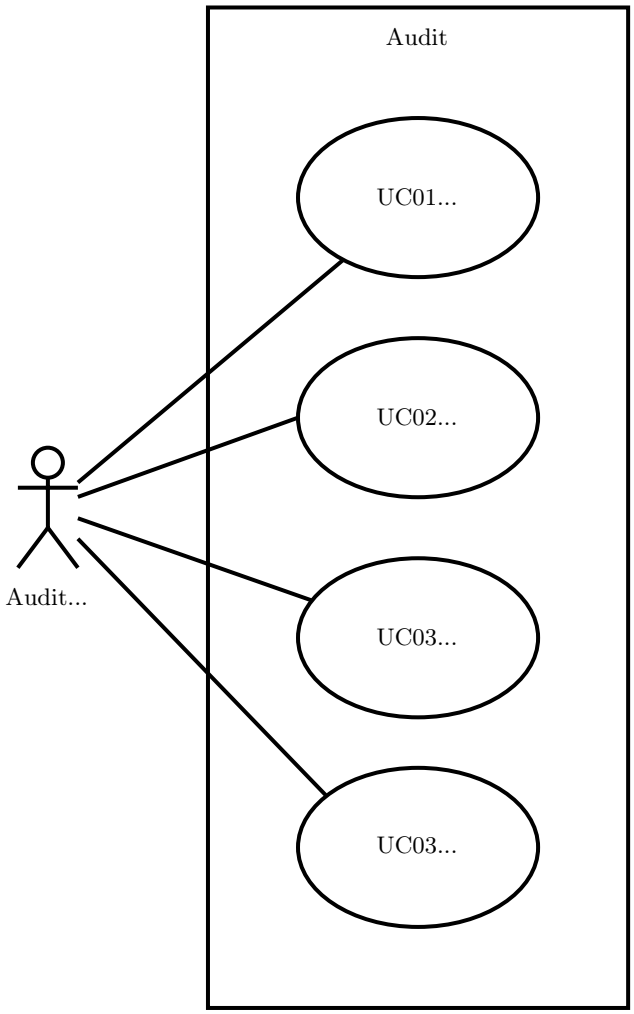

Figure 6.1: Audit Component Business Context Diagram

Technical Context

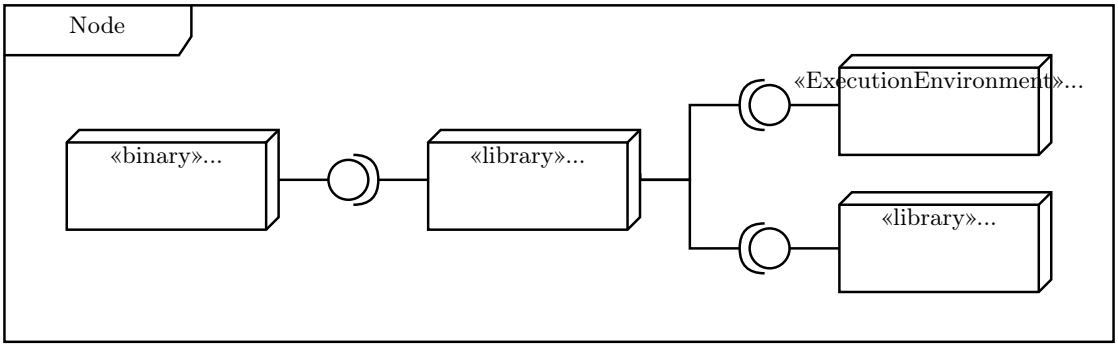

Figure 6.2: Audit Component Technical Context Diagram

6.3.2 Solution Strategy

Table 6.4: Audit component solution strategy

Actor	Funktion	UCID	Semantics	Pre-condition	Post-condition
QueryHandler	NewAudit-LogServer-(EventStore-Client, Event-Formatter-Registry)	All	Creates server instance of the audit log component to handle grpc client requests	EventFormatter server is running and Event-Formatters are registered	Audit-log server is ready to handle client requests
AuditLogClient	GetByDate-Range(Range-Request, Stream)	UC01	streams formatted events	AuditLogServer is running	human-readable events were streamed to the client
...	...	...	...	...	...

...

finish the table

6.3.3 Building Block View

Overall System White Box

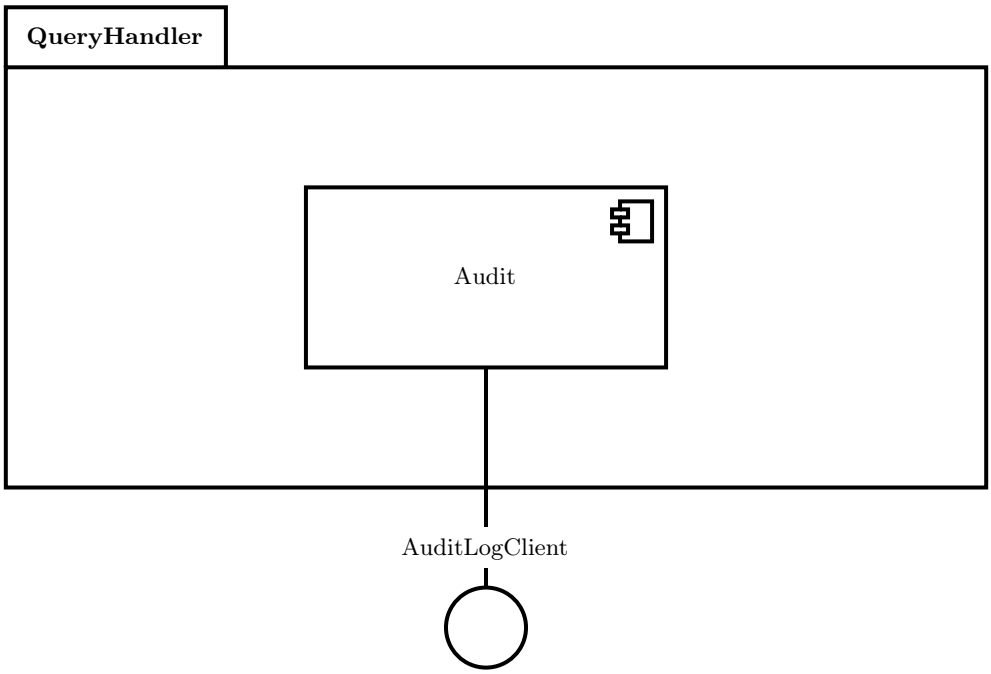

Figure 6.3: Overall System Component Diagram

Level 1

Contained Building Blocks

Table 6.5: Audit component overall system contained building blocks white box

Component	Description
Audit	handles aggregating and formatting events for audit related queries

Audit Component Black Box

Table 6.6: Audit component black box

Interface	Description
AuditLogClient	handles communication with the audit log server

Level 2

Audit Component White Box

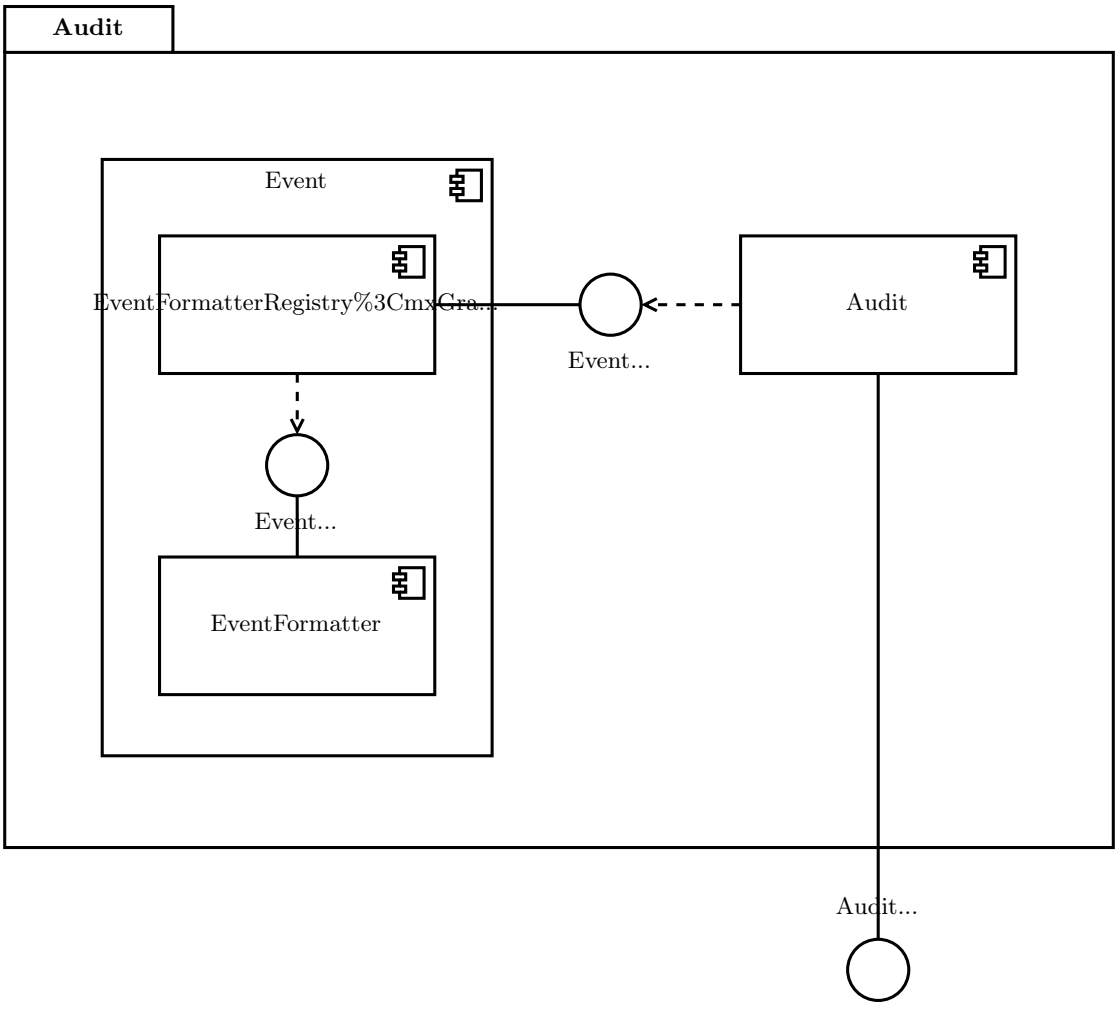

Figure 6.4: Audit Component Diagram

Table 6.7: Audit component contained building blocks white box

Component	Description
Audit	handles audit client queries
EventFormatterRegistry	register and initialise EventFormatters based on the event type
EventFormatter	formats an event in human-readable format

Event Component Black Box .

Table 6.8: Event component black box

Interface	Description
EventFormatterRegistry	register and initialise EventFormatters based on the event type
EventFormatter	formats an event in human-readable format

Level 3 Class Diagram

6.3.4 Runtime View

6.3.5 Deployment View

6.4 Design Decisions

6.4.1 DD01: Registry pattern

6.5 Technical Decisions

6.5.1 TD01: Limit get user actions query to max 1 year

a general restructure for the event structure will happen sooner or later (which should allow for a much better implementation) for now, limiting the query to a max 1 year should be enough as a preventive measure until the restructuring to filter on the store level is implemented All queries are bound to a time-out which will stop the query if it takes too long In case there are too many events within the limiting period.

7 Audit Browser

When it comes to staying complaint ascertaining the validity and reliability of information in organizations and their associated processes is a very important job of the organization security officers. It is also customary to have yearly audit sessions done by external entities. However, waiting for the audit report to hit should not be an option. Especially when it comes to highly regulated industries.

In chapter 3 Table **??** an example overview of the different internal auditing activities were showcased and discussed. The detective controls done by administrators like security officers rely on audit logs and the corresponding aggregation programs. Assuming a proper audit logging infrastructure there is still a dependency on the developer to properly log the relevant activities, which poses the responsibility to evaluate and reliably provide a consistent and complaint audit logs. Not only is this prone to human errors it is by design bond to to not provide the full picture and might lead to over logging.

With time the audit log file just becomes larger and larger and going through the logs becomes very tedious and tend to be avoided. Even when it comes the external controls it is customary to work with pre-recorded parteitonend samples, which might give a good overview but never the full one.

Auditing 2.0 discussed in section 3.4 describe practices to mitigate the mentioned problems and provides the basis for an auditing frameworks in attempt of continues auditing approach. Using Event-Sourcing as a reliable audit logging mechanism and the Audit Component to implement the different process mining modules all what is left is to provide a user friendly interface to the auditors while keeping the fact in mind, that most auditors have different technical backgrounds and some have none.

This implementation attempts a basic Minimum Viable Product (MVP) as a POC of the reading modules, that can be extended easily to handle actionable ones.

7.1 Requirements

The Audit Component implemented in chapter 6 is to be utilised to offer an audit log reporting Graphical User Interface (GUI) to enable continues auditing and lay the ground work for Auditing 2.0.

M8 offer much more features, that can be utilised. Since m8 has no official GUI the base implementation for the Audit Browser should be expandable to accommodate other use-cases and implementations.

Upstream Identity and Access Management (IAM) providers like onelogin are to be supported by implementing m8 authentication flow as described in section .

add section ref

Since Auditors have different backgrounds and technical knowledge the User Experince (UX) should be intuitive and well thought out. The following is to be considered:

- The same event view musst be ensured to ease dealing with deferent events and allow for a sense of familiarity. For example if a table is used to showcase the event, the table structure is preserved for all types.
- lack of events is not an error and musst be clearly represented

To ease auditors workflows and external auditing needs audit reports should be exportable and sendable via e-mail in a spread-sheet compatible format like Comma Separated Values (CSV). Automating these reports should be possible as well.

see if this will be possible timewise

7.1.1 Use-Cases

ID	Use-Case	Description
UC01	Monoskope GUI	As a user, I want to use m8 features through an easy to navigate GUI
UC02	OIDC authentication	As a user, I want to authenticate using my OIDC provider account
UC03	Audit-log for date-range report	As an auditor, I want to see all actions taken within a specific date-range
UC04	Audit-log about a user report	As an auditor, I want see all actions taken on a user
UC05	Audit-log of user-actions report	As an auditor, I want see all actions taken by a user
UC06	Audit-log users overview	As an auditor, I want a report of all users at a specific timestamp, tenants they belongs to, and their roles within the system or tenants/clusters
UC07	CSV audit-log reports export	As an auditor, I want to export audit-log reports in a CSV formatted file GUI
UC08	Send audit-log reports via e-mail	As an auditor, I want to send audit-log reports via e-mail GUI

Table 7.1: Audit Browser derived use-cases

7.1.2 Architectural Constraints

Technical Constraints

ID	Constraint	Description
TC01	Expandable GUI	GUI implementation allow for other m8 features support
TC02	Unified event view	Audit-log reports have unified event views
TC03	Backend	Backend is written in GO

Table 7.2: Audit Browser technical constraints

Organisational Constraints

ID	Constraint	Description
OC01	Deadline	implementation musst be finalised before 28.02.2023
OC02	The Twelve-Faktor App	adhere to the The Twelve-Faktor App methodology
OC03	Userbase	Userbase is mainly desktop users

Table 7.3: Audit Browser organisational constraints

7.2 System Design

7.2.1 Scope and Context

Business Context

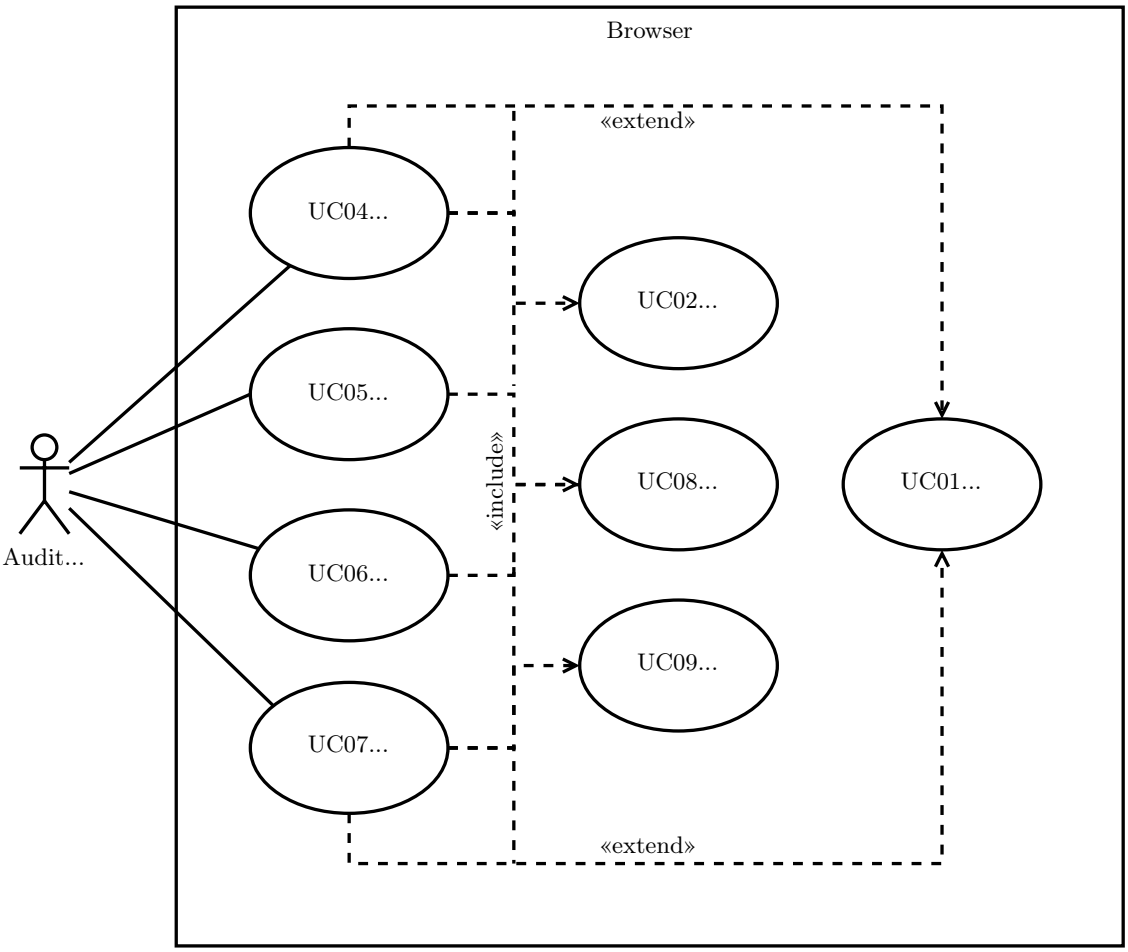

Figure 7.1: Audit Browser business context diagram

Technical Context

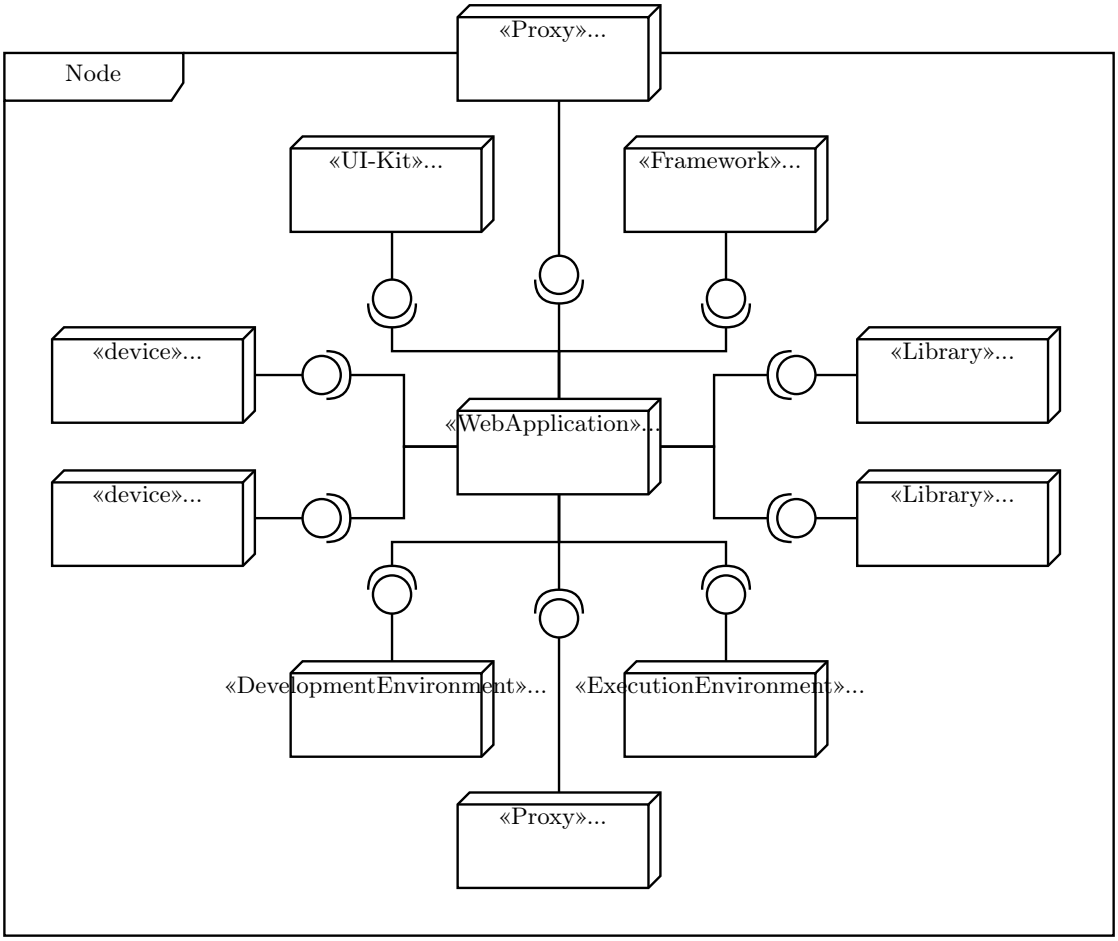

Figure 7.2: Audit Browser technical context diagram

7.2.2 Solution Strategy

Function	UCID	Semantics	Precondition	Postcondition
render(-components)	UC01	render components in the browser	user navigated to MonoGUI based URI	requested components are rendered or user is redirected to authenticate himself
signIn(-m8APIUrl)	UC02	Kick in m8 OIDC authentication flow	user clicked the sign in button	user is redirected to IDP authentication URL
requestAuth(-authCode)	UC02	request authenticaton token from m8	user signed in using his IDP account	user can access protected routes
getAuditLog(-from, to, kargs)	UC03 UC04 UC05 UC06	get audit log depending on the usecase as described in 7.1	user provided date range and other inputs based on the usecase	user can browse the requested log entries

Table 7.4: Audit Browser solution strategy

7.2.3 Building Block View

Overall System White Box

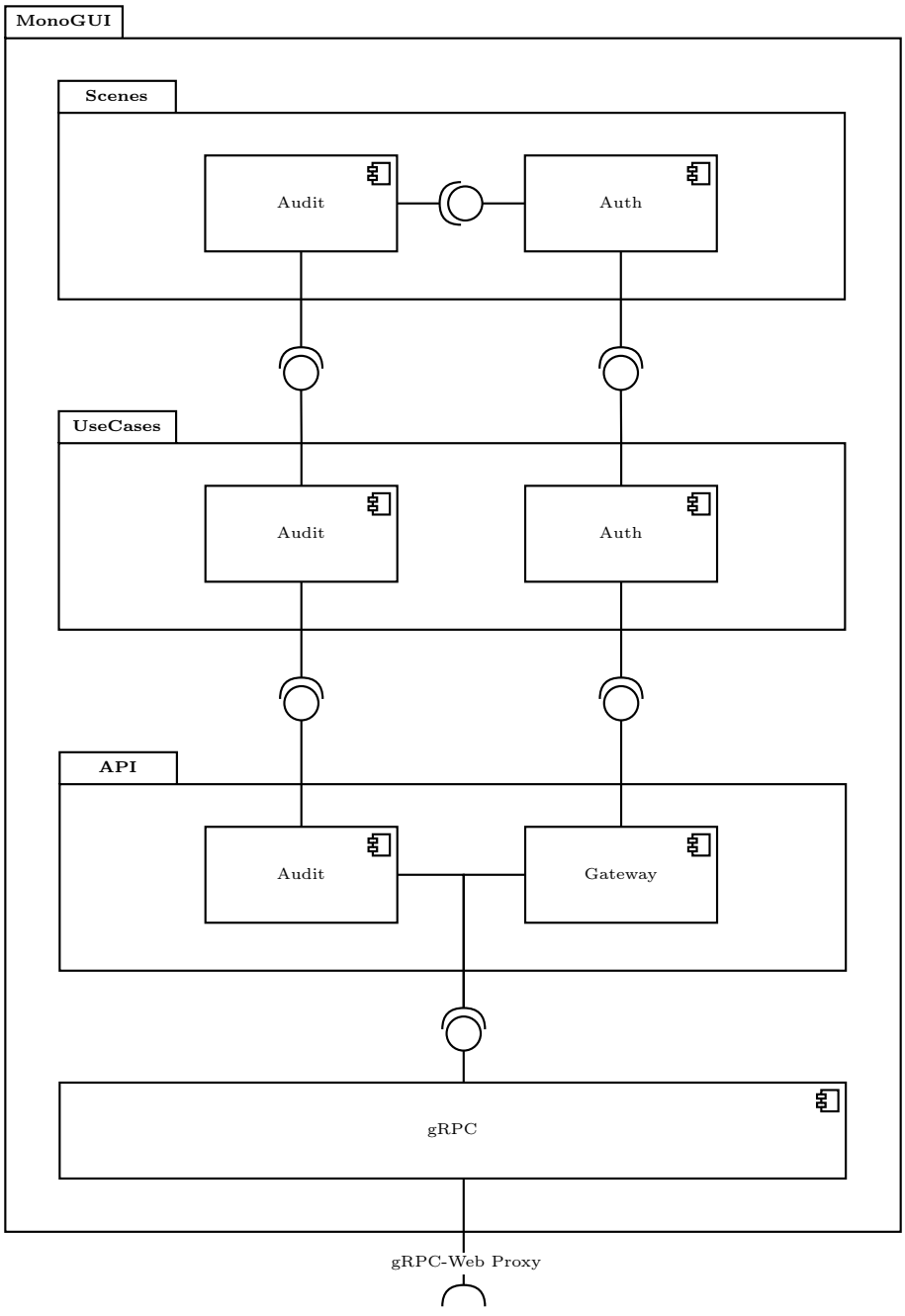

Figure 7.3: Audit Browser overall system component diagram

Contained Building Blocks

MonoGUI

Component	Description
gRPC	handles comunication with gRPC-Web Proxy

Table 7.5: Audit Browser MonoGUI contained building blocks black box

Scenes

Component	Description
Audit	handles rendering of the audit components and user inputs
Auth	handles rendering of the authentication components and user inputs

Table 7.6: Audit Browser scenes contained building blocks black box

UseCases

Component	Description
Audit	handles communication with m8's Audit Component and data preparation
Auth	handles communication between m8's Gateway, the user, the upstream IDP

Table 7.7: Audit Browser UseCases contained building blocks black box

API

Component	Description
Audit	m8's Audit Component stubs to handel gRPC requests
Auth	m8's Gateway stubs to handel gRPC requests

Table 7.8: Audit Browser API contained building blocks black box

Level 2

MonoGUI White Box

gRPC

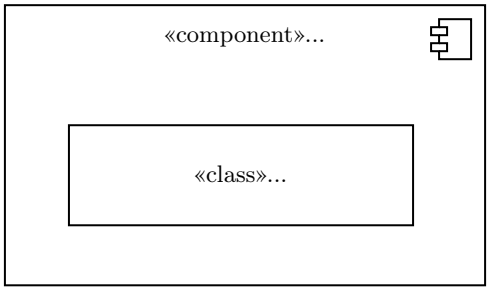

Figure 7.4: Audit Browser MonoGUI gRPC class diagram

Object	Description
GrpcConnectionFactory	creates preconfigured gRPC connection based on the use case for example authenticated with timeout and retries connection

Table 7.9: Audit Browser MonoGUI gRPC class diagram

Scenes White Box

Audit

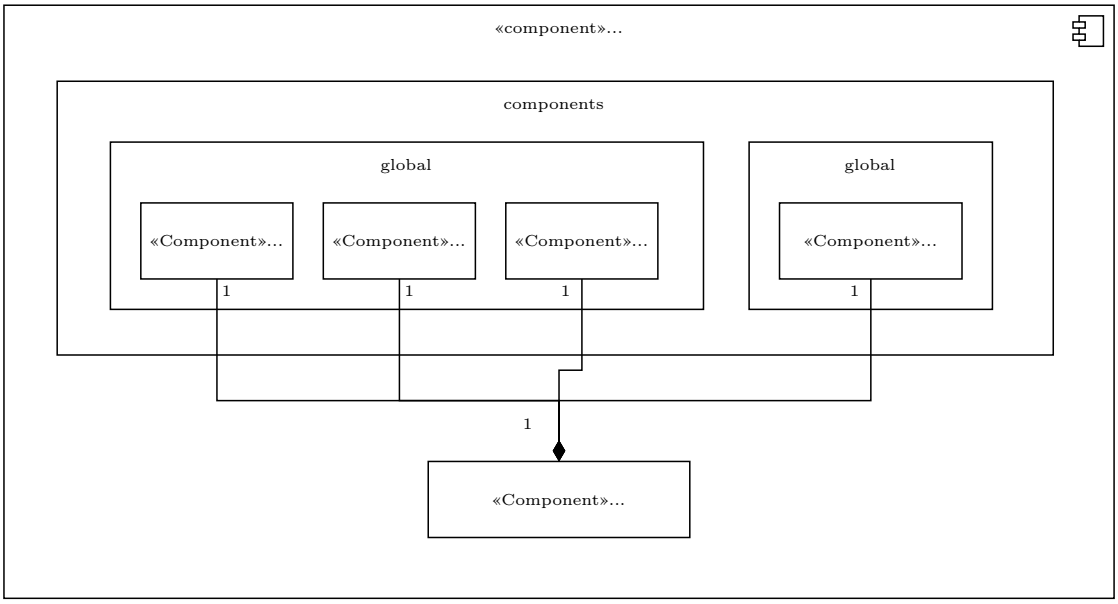

Figure 7.5: Audit Browser scenes audit class diagram

Object	Description
Sidebar	composition component for sidebar presentation and interaction
Header	composition component for content header presentation and interaction
Topbar	composition component for topbar presentation and interaction
AuditDatePicker	composition component for date range presentation and interaction
Audit	composed component for get by date range use case UC03 7.1 presentation and interaction

Table 7.10: Audit Browser scenes audit class diagram

Auth

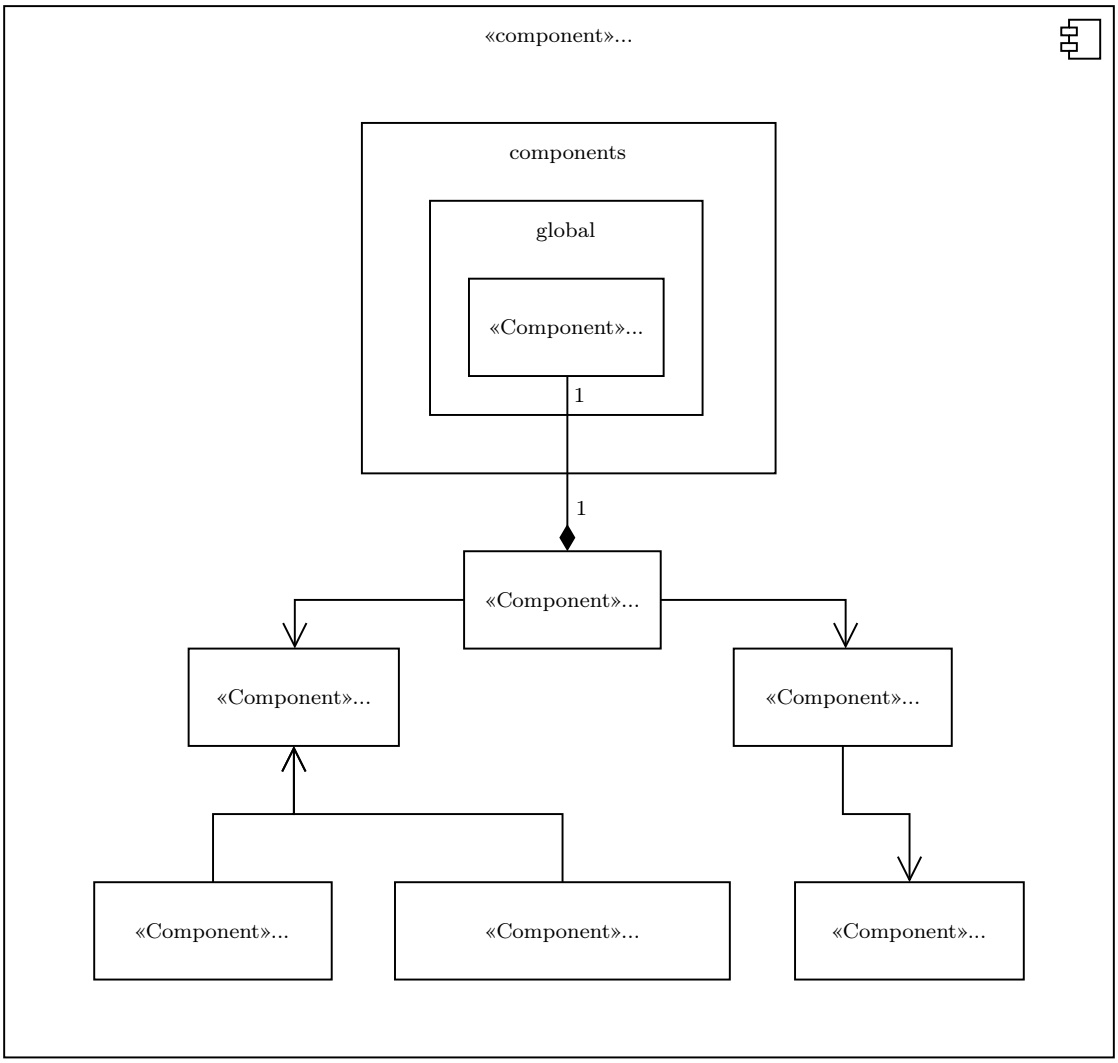

Figure 7.6: Audit Browser scenes auth class diagram

Object	Description
ThemeButton	composition component for the sign in button presentation and interaction
Auth	composed component for the authentication use case UC02 7.1 presentation and interaction
AuthContext	composition component to provide authentication context for composing components
useAuth	composed component to manage authentication flow components
AuthSecure	composition component to put composing components behind authentication wall
useAuthenticatedClient	composition component to provide preconfigured and authenticated gRPC client for composing components
AuthPopup	composition component for upstream IDP presentation and interaction

Table 7.11: Audit Browser scenes auth class diagram

UseCases White Box

Audit

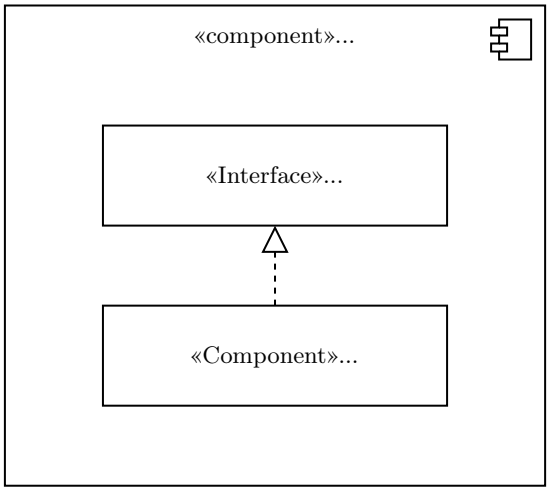

Figure 7.7: Audit Browser usecases audit class diagram

Object	Description
UseCase	use case base for API aggregation and data preparation for presentation components
GetAuditLogUseCase	request audit log events from the API and prepare them for presentation components

Table 7.12: Audit Browser usecases audit class diagram

Auth

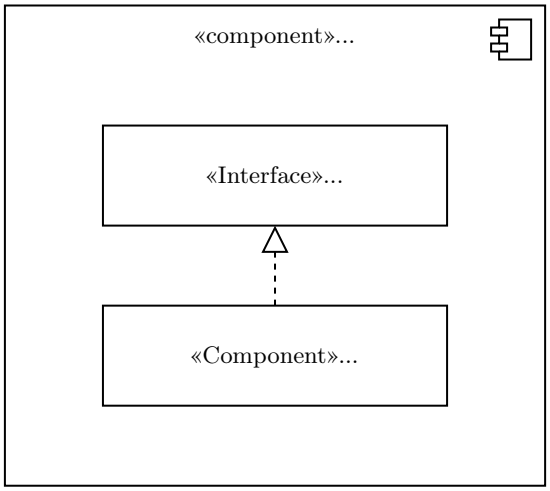

Figure 7.8: Audit Browser usecases auth class diagram

Object	Description
AuthUseCase	coordinate authentication flow initialisation and callbacks between the API and presentation components

Table 7.13: Audit Browser usecases auth class diagram

7.2.4 Runtime View

UC01: Monoskope GUI

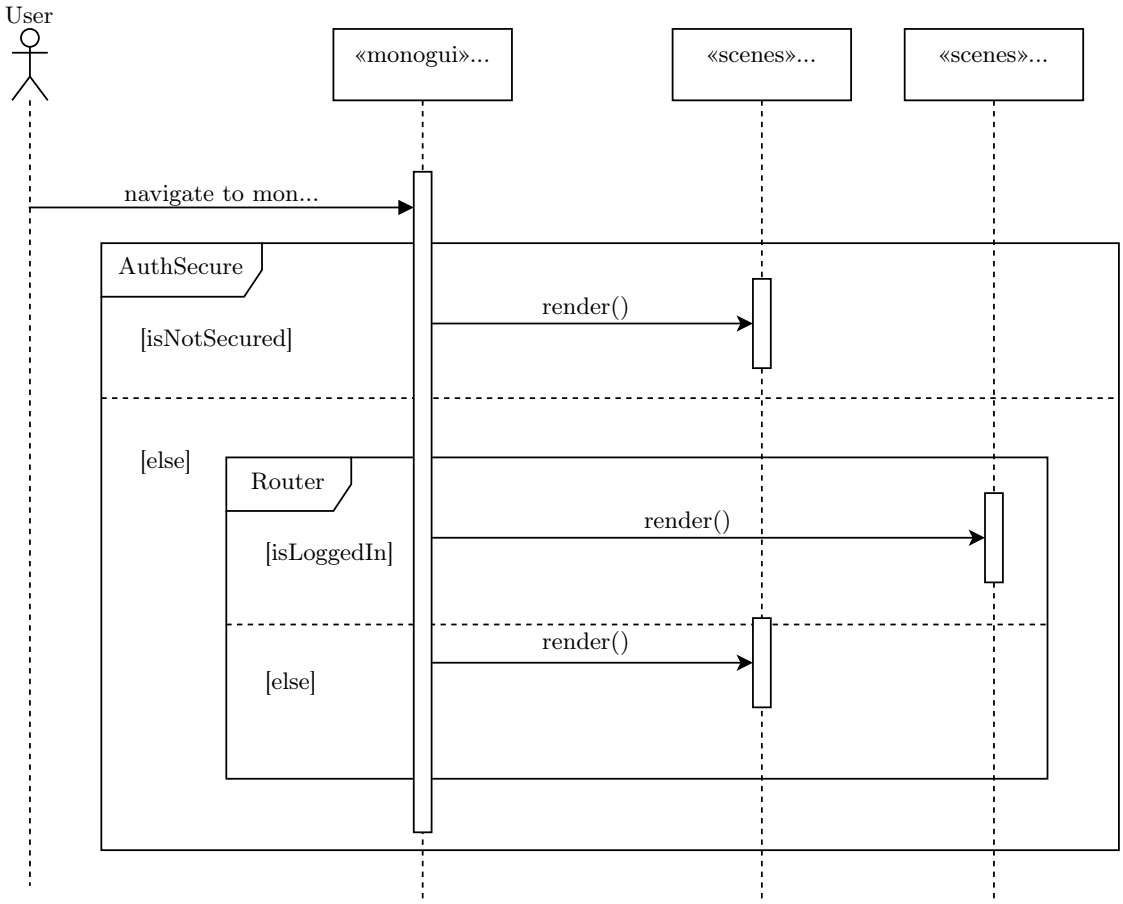

Figure 7.9: Audit Browser UC01 sequence diagram

UC02: OIDC authentication

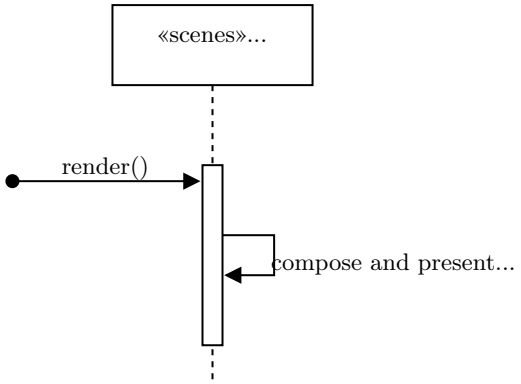

Figure 7.10: Audit Browser UC02 sequence diagram

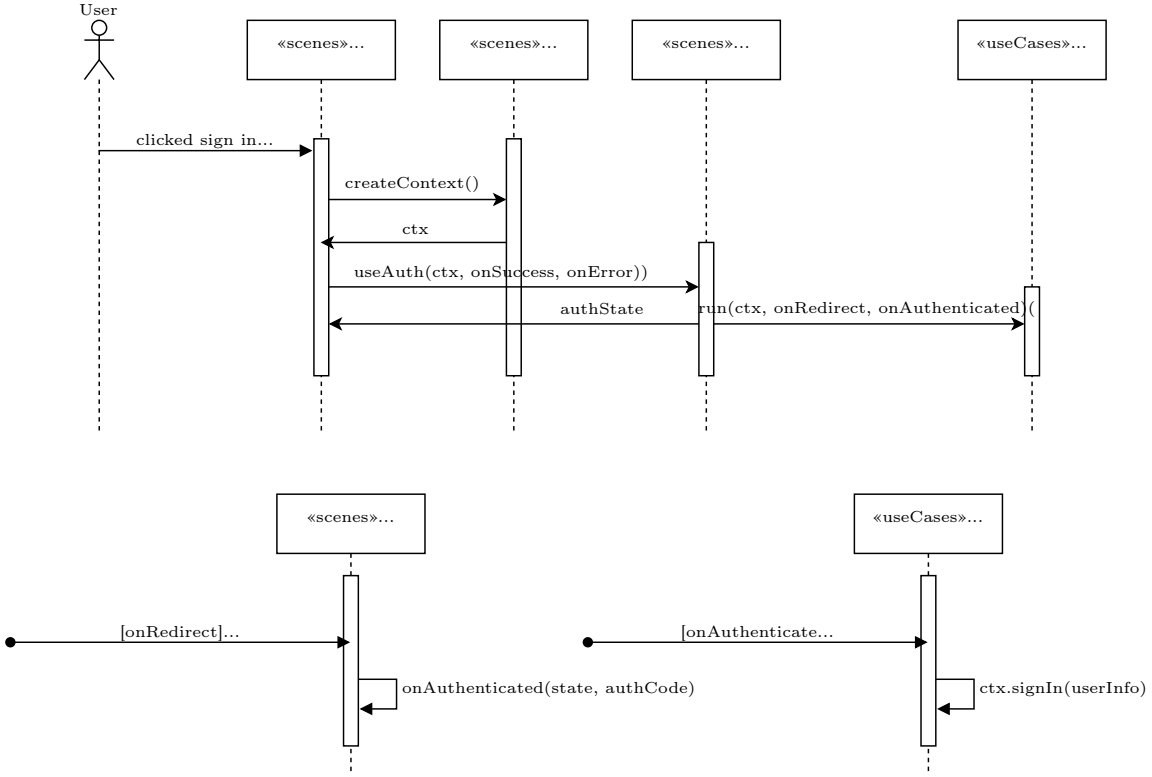

Figure 7.11: Audit Browser UC02 sign in sequence diagram

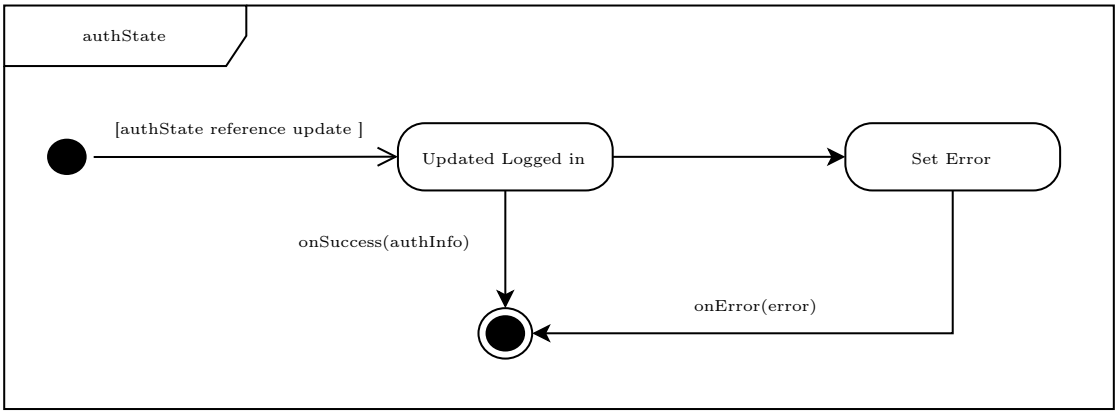

Figure 7.12: Audit Browser UC02 auth state machine

UC03: Audit-log for date-range report

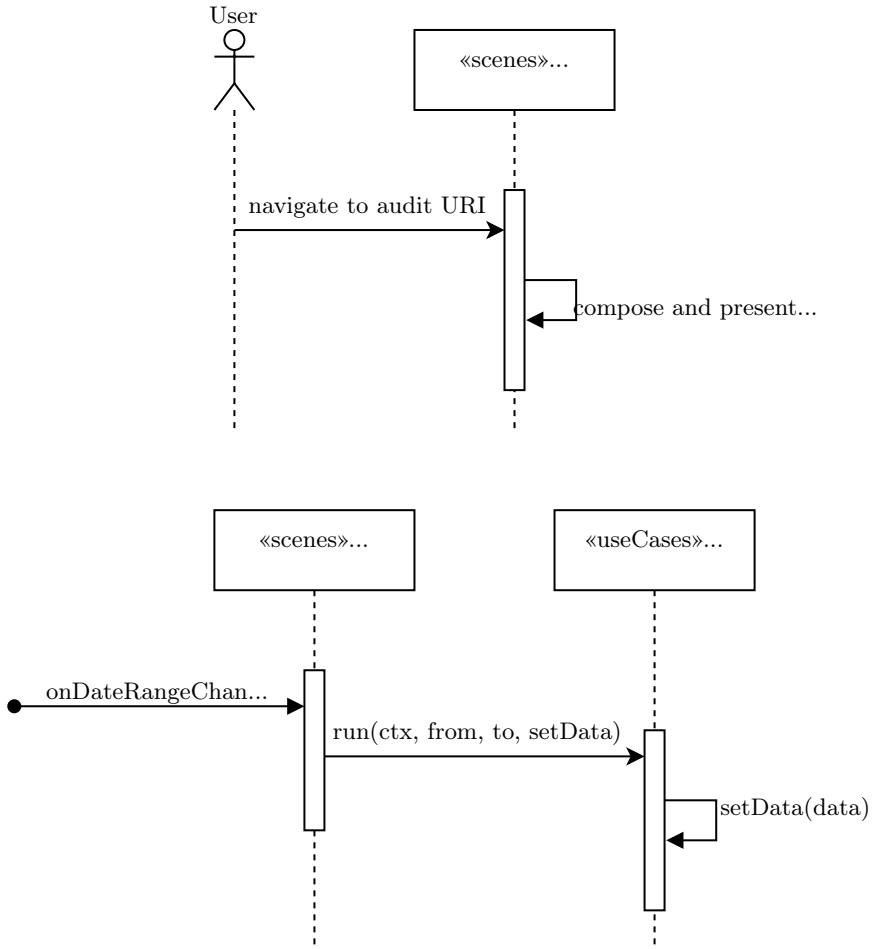

Figure 7.13: Audit Browser UC03 sequence diagram

7.3 Design Decisions

7.3.1 DD01: gRPC Client-Server Communication

HTTP/2 comes with multiple advantages to handle HTTP/1.1 limitations like multiplexing, HPACK compression and server push mechanism, which allows the server to push streams of messages without waiting for an explicit client request [Belshe u. a., 2015]

As of the time of writing Browsers do support HTTP/2 but only for static files like images, javascript, css etc. XMLHttpRequest/Ajax calls are still carried over HTTP/1.1. This is due to the fact, that browsers have no unified specification to handle HTTP/2 trailers.

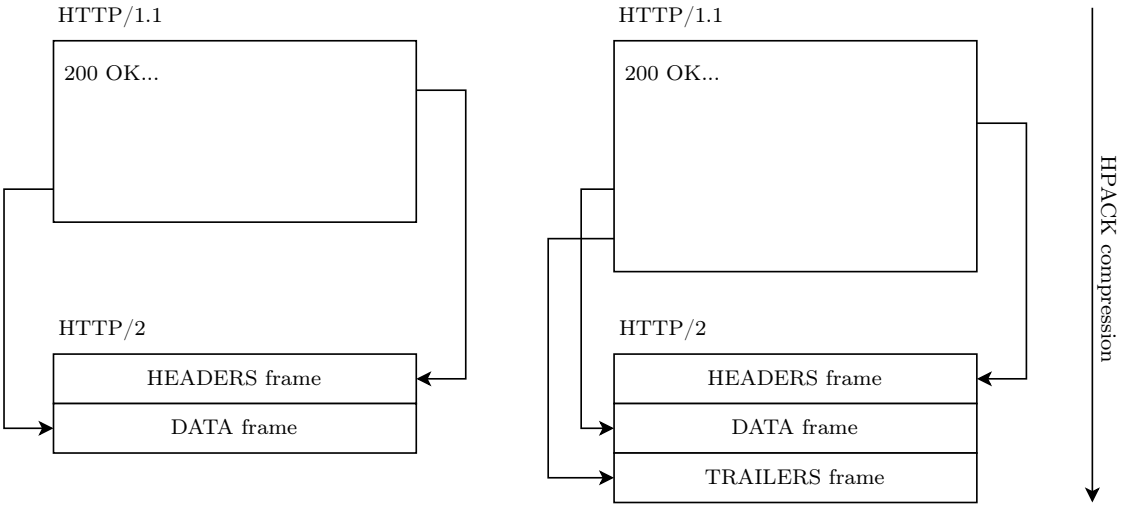

Figure 7.14: HTTP/1.1 compared to HTTP/2

gRPC makes heavy use of trailers to send status messages when streaming responses [Goolge, 2015]. Because of the browser HTTP/2 limitations Javascript (JS), the programming language of the web, can not directly talk to a gRPC APIs as normally done with REpresentational State Transfer (REST) APIs.

To mitigate this limitation a translation proxy has to be used to translate gRPC to REST requests/responses [Brandhorst, 2019].

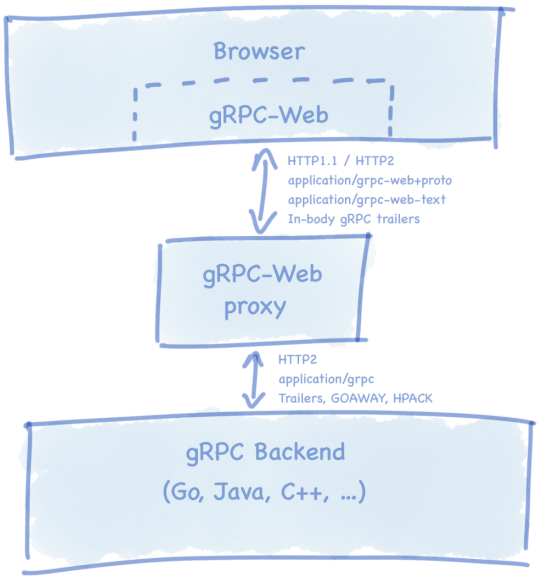

Figure 7.15: gRPC-Web proxy to allow browser gRPC support [Brandhorst, 2019]

Why stick to gRPC?

Deciding weather or not it is worth the effort to go through the hassle of proxying all requests to mitigate browser support and navigating through mainly an edge technology when it comes to web-development with no clear guides and mediocre documentation comes back to the magic phrase "it depends". the gained efficiency compared to REST and the maintainability efforts, especially for production usecases, requires thorough evaluation.

- Messages passing back and forth are encoded which makes them hard to read and debug
- Production use is minimal at best since the effort is still not worth the gaines for already established services
- there is still no Client-side and Bi-directional streaming support
- official Protobuf compiler extensions are buggy and have no ECMAScript support (usage of third party extensions like protobuf-es is needed)

Despite the mentioned above gRPC still shines when it comes to straightforward API definitions, efficient and compact mechanism to exchange big messages [Richardson, 2018], streaming and HTTP/2 advantages justify its usage in the right context.

Other Options

The only other option is to actively provide backend REST support by Transcoding HTTP/JavaScript Object Notation (JSON) to gRPC and utilising a reverse proxy to to serve a RESTfull API [gRPC Gateway Authores, 2023]

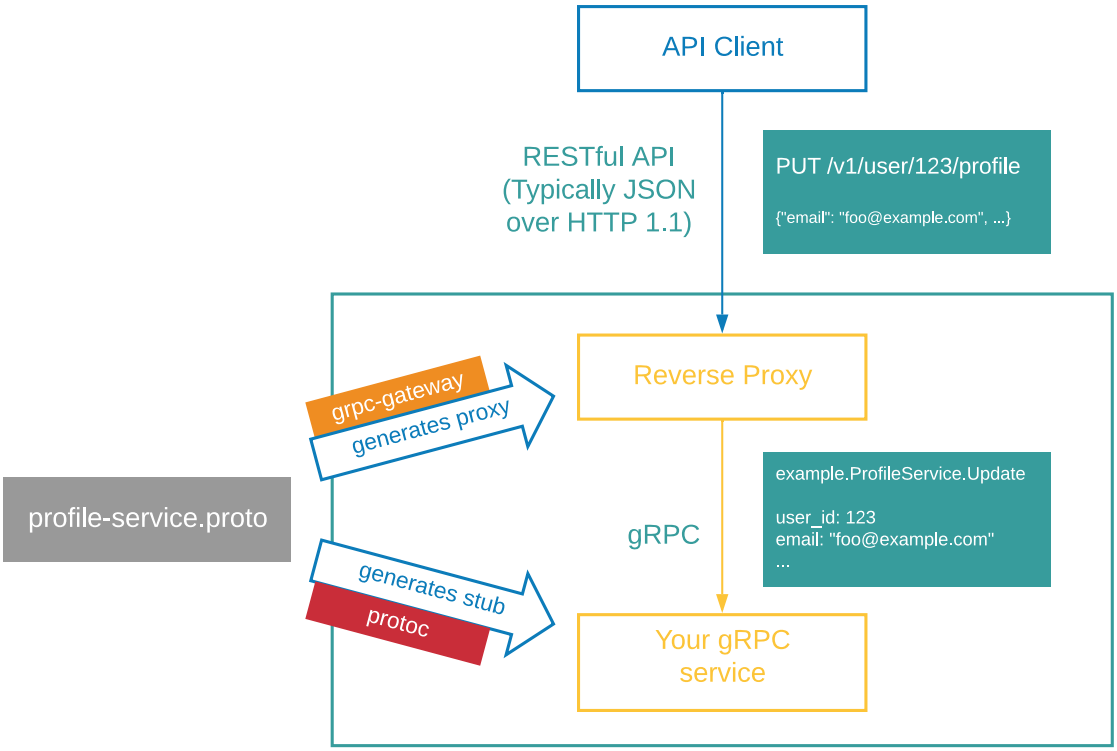

Figure 7.16: gRPC-Gateway architecture diagram [gRPC Gateway Authores, 2023]

While this comes with the RESTfull API advantages and full support of the web development community it dose not differ in it's essnase to gRPC-Web yet comes with the disadvantage of the server transcoding overhead.

Conclusion

From the perspective of using a technology that is on the bleeding edge of its peers just getting the basic workflows working with it vs. the well established patterns of REST is challenging.

However, since m8 only speaks gRPC and maintaining a REST API is not an options TC03, due to the fact that solutions do exists to to handle REST-gRPC translation, until full gRPC support by the browsers and the alternatives offers no real advantages in this case it was decided to go with gRPC for client-server communication.

7.4 Technical Decisions

7.4.1 TD01: Javascript (JS) and React

It is no secret that JS is the language of the web and is almost always ranked first by multiple indexes compared to other web development programing languages.

Index	1st	2nd	3rd	4th	5th
IEEE Spectrum	Python	Java	C	C++	JavaScript
GitHut 2.0 (Pull Requests)	JavaScript	Python	Java	Go	Ruby
GitHut 2.0 (Pushes)	JavaScript	Python	Java	C++	PHP
RedMonk	JavaScript	Python	Java	PHP	C++/C#
PYPL	Python	Java	Javascript	C#	C/C++
TIOBE	C	Python	Java	C++	C#

Figure 7.17: Top programming languages - rankings in comparison [Tagliaferri, 2023]

In a survey conducted on 39,472 person a clear trend towards Typescript (TS) can be seen as it improves the developer experience and provides various other features mainly based on the added typing support [Greif und Burel, 2022]

However writing vanilla JS or TS tend to be very cumbersome and follow a scripting approach (hence the name). Depending on the use-case this might be much preferable, sense it offer total control and a declarative approach when it comes to handling Extensible Markup Language (XML) and HyperText Markup Language (HTML) elements.

On the other hand building and maintaining web applications requires a programming approach and patterns to achieve production level results, which lead to the development of many frontend development frameworks lke React, Angular, Vue.js and many others.

Among the 3 stables mentioned above React has the higher usage across all years since 2016 [Greif und Burel, 2022] since it comes with many basic yet powerful features, that can be expanded depending on the use-case. Angular on the other hand tend to be tailored towards enterprise usage and comes with greater complexity and steeper learning curve, while Vue.js attempt to be the happy medium in between.

Conclusion

While TS seems to be the smart choose to make, especially for production environments the added complexity requires more development time. Since MonoGUI is meant to be a POC and is for now mainly developed as a base for the Audit Browser and due to the deadline set by OC01 it was decided to go with JS in combination with React since they offer the perfect platform for the planned solution.

7.4.2 TD02: Headless Use-Cases Implementation

While TD01 specifies usage of JS and React MonoGUI expansion and further development should be kept in mind as specified by TC01.

To stay flexible and framework independent it was decided to split the business logic from the presentation logic or as is commonly known follow a headless implementation approach. The latter specifies writing business logic in vanilla JS and the presentation

logic in the used framework notation. This allows for easier migration to Typescript while also staying framework independent.

8 Installation and Configuration

Installing and configuring the Audit Component and Audit Browser spans multiple resources and requires intricate yet clear and logical orchestration.

Such processes tend to be automated in production environments to insure auditable and reversible actions through Version Control, speed up development, reduce human errors, and benefits from Continuous Integration (CI)/Continuous Delivery (CD) practices.

Utilizing Docker, k8s, Helm and various backing technologies while keeping The Twelve-Faktor App [Wiggins, 2017] rules in check (OC02) the entire installation and configuration process also known as a deployment can be trimmed down to just a few steps.

8.0.1 Deployment View

The following is a general overview of the main resources and services to successfully deploy and use the 6 and 7

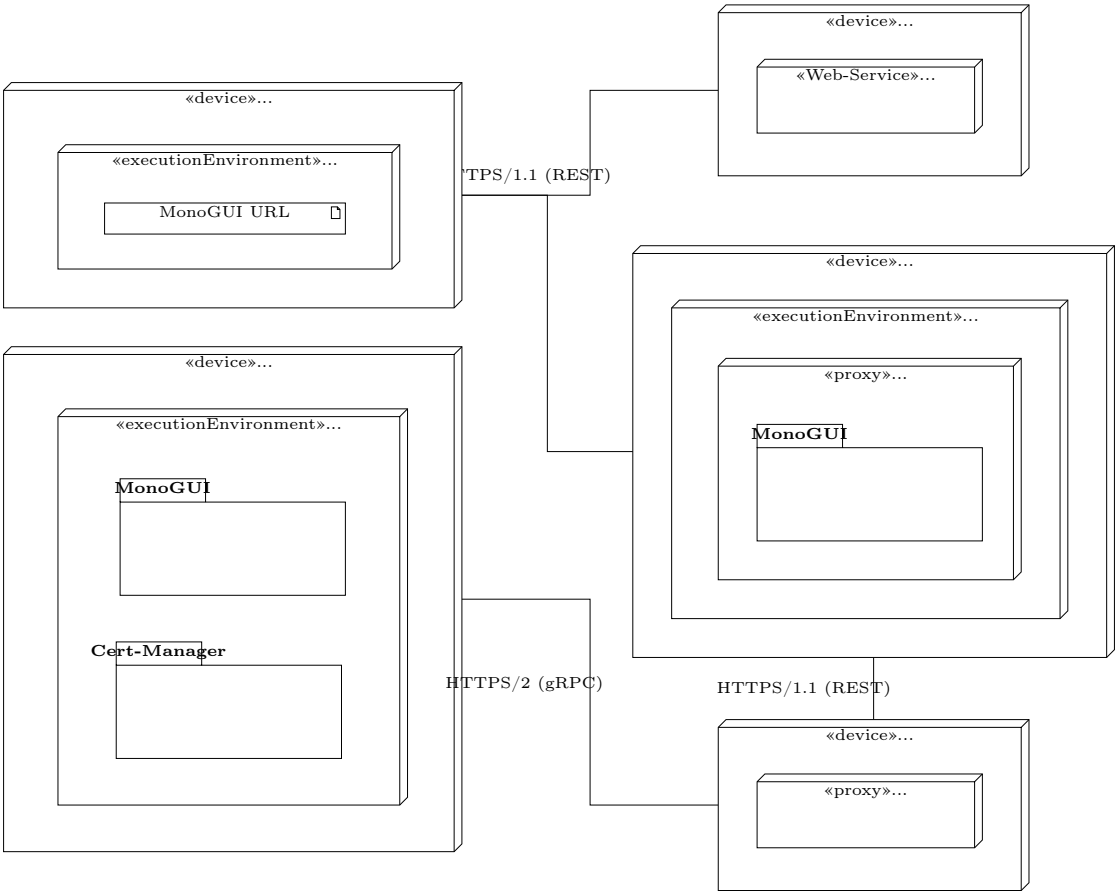

Figure 8.1: Deployment view diagram

Running MonoGUI and the gRPC-Web proxy on separate nodes is not necessary but supported to allow for flexible deployment usecases.

8.1 Test Run

For the local installation all resources and services will be deployed on the same node. Since m8 already uses Emissary-Ingress it will be used as a gRPC-Web proxy. Emissary-Ingress will be used as the IDP.

To deploy m8 and MonoGUI along all other resources to a local cluster and experiment with the Audit Component and Audit Browser a script is provided, that will:

1. take care of the preparation work and installing the automated (8.1) tools
2. create local k8s cluster using Kind
3. add m8 and MonoGUI dependencies to Helm
4. setup and deploy all needed resources

All while making sure to stay idempotent and localized.

Prerequisite

A supported system from the following:

- MacOS
- Linux (tested on ubuntu)
- Windows (WSL)

Standard userspace utilities like `bash`, `curl`, `tar`, etc.... You can always change to alternatives by setting the corresponding environment variable. For example replacing `curl` with `wget` will be as follows:

```
$ CURL=wget make deploy
```

Shell Instructions 8.1: Deploy with custom command

The following tools:

Tool	Reson
Docker	to create isolated environments
Make	to run deployment scripts

Table 8.1: Installation required tools

Automated

The following will be downloaded and configured locally. If any should fail please download and install manually then follow the same instruction as described in the shell example 8.1.

Tool	Reson
Kubectl	to manage k8s
Kind	to create a local k8s cluster
Helm	to generate components resources and install required CRDs
Step-cli	to generate m8 PKIs trust-anchor

Table 8.2: Installation automated tools

Configure

Depending on the version on hand open the provided CD-drive or navigate to https://thesis.alshikh.de and clone the repository.

As specified by The Twelve-Faktor App [Wiggins, 2017] methodology (OC02) all configuration are environment specifc and can be configured locally or using k8s resource definiations by setting the the corresponding enviorment variable.

All variables are documented under the directory `deploy/setup` and set with defaults for ease of use.

Install

Open the directory `deploy` and follow the following instruction. Just retry.

If for some reason a command should fail the system and commands are ensured to be idempotent.

note style

test on windows and ubuntu

1. Create a local cluster and deploy all resources:

```
$ make deploy
# you can also run the following to monitor the state
of the resources
$ make kind-watch
```

Shell Instructions 8.2: Deploy all resources to a local cluster

2. Trust m8 domain certificate: `tmp/domain-ca.crt` , otherwise the browser will block communication with m8 API

```
# MacOS
$ sudo security add-trusted-cert -d -r trustRoot -k "
/Library/Keychains/System.keychain" tmp/domain-ca.crt
```

Shell Instructions 8.3: Trust m8 domain certificate

3. Add the following to your hosts file:

```
# Linux & MacOS: /etc/hosts
# windows: c:\Windows\System32\Drivers\etc\hosts
127.0.0.1 api.monoskope.dev
127.0.0.1 dex
```

Shell Instructions 8.4: Update hosts file

4. Create port-forwards to route local request to backing services in the cluster: (Make sure `8443` , `5556` and `3000` are not in use)

```
$ make port-forward
```

Shell Instructions 8.5: Create port-forwards to route local request

5. navigate to http://localhost:3000 and sign in using the following credintials:
   - `username:` admin@monoskope.dev
   - `password:` password
6. Populate the EventStore with some mock data for a better UX

```
$ make mock-data
```

Shell Instructions 8.6: Populate the EventStore with some mock

9 Conclusion

Weiterentwicklungsmöglichkeiten

In this case the Audit Browser was implemented as a part of MonoGUI and used for m8 only. As discussed in chapter 3 section 3.4 Having a central and unified home for log monitoring and further processing especially on an organisation level further improve and complies with Auditing 2.0 vision. querring raw events from m8 or any event-sourced system is and should be possible. Depending on the use-case further processing upon ingestion might be needed to allow for some unification .....

Process mining poses a lot of possibilities and combained with Event-Sourcing allow for very advanced log analisys and bhaviour control .....

Bibliography

[van der Aalst u. a. 2010] Aalst, Wil M. van der ; Hee, Kees M. van ; Werf, Jan M. van der ; Verdonk, Marc: Auditing 2.0: Using Process Mining to Support Tomorrow's Auditor. In: *Computer* 43 (2010), Nr. 3, S. 90–93. – URL https://ieeexplore.ieee.org/document/5427384

[BaFin 2021] BaFin, Bundesanstalt für F.: *Rundschreiben 11/2021 (BA) - Zahlungsdiensteaufsichtliche Anforderungen an die IT (ZAIT).* 11 2021. – URL https://www.bafin.de/dok/16514544. – (Accessed on 12/15/2022)

[Bass u. a. 2003] Bass, Len ; Clements, Paul ; Kazman, Rick: *Software Architecture In Practice.* URL https://books.google.de/books?id=mdiIu8Kk1WMC, 01 2003. – ISBN 978-0321154958

[Belshe u. a. 2015] Belshe, M. ; Peon, R. ; Thomson, M.: Hypertext Transfer Protocol Version 2 (HTTP/2) / RFC Editor. RFC Editor, May 2015 (7540). – RFC. – URL http://www.rfc-editor.org/rfc/rfc7540.txt. http://www.rfc-editor.org/rfc/rfc7540.txt. – ISSN 2070-1721

[Brandhorst 2019] Brandhorst, Johan: The state of gRPC in the browser | gRPC. (2019), 01. – URL https://grpc.io/blog/state-of-grpc-web/. – (Accessed on 02/19/2023)

[Fowler 2004] Fowler, Martin: *Audit Log.* https://martinfowler.com/eaaDev/AuditLog.html. 03 2004. – URL https://martinfowler.com/eaaDev/AuditLog.html. – (Accessed on 02/20/2023)

[Fowler 2022] Fowler, Martin: Event Sourcing. In: *martinfowler.com* (2022). – URL https://martinfowler.com/eaaDev/EventSourcing.html. – (Accessed on 12/24/2022)

[Gantz 2014] Gantz, Stephen D. ; Gantz, Stephen D. (Hrsg.): *The Basics of IT Audit*. Boston : Syngress, 2014. – URL https://www.sciencedirect.com/book/9780124171596. – ISBN 978-0-12-417159-6

[gRPC Gateway Authores 2023] Gateway Authores gRPC: gRPC-Gateway. (2023), 02. – URL https://github.com/grpc-ecosystem/grpc-gateway#usage. – (Accessed on 02/19/2023)

[Goolge 2015] Goolge: gRPC over HTTP2. (2015), 08. – URL https://github.com/grpc/grpc/blob/master/doc/PROTOCOL-HTTP2.md. – (Accessed on 02/01/2023)

[Greif und Burel 2022] Greif, Sacha ; Burel, Eric: The State of JS 2022. (2022), 12. – URL https://2022.stateofjs.com/en-US/. – (Accessed on 02/19/2023)

[ISO 19011 2018] ISO 19011: *ISO - ISO 19011:2018 - Guidelines for auditing management systems*. 07 2018. – URL https://www.iso.org/obp/ui/#iso:std:iso:19011:ed-3:v1:en:term:3.1. – (Accessed on 12/13/2022)

[Kubernetes 2014] Kubernetes: Kubernetes. (2014), 09. – URL https://kubernetes.io/. – (Accessed on 02/16/2023)

[Overeem u. a. 2021] Overeem, Michiel ; Spoor, Marten ; Jansen, Slinger ; Brinkkemper, Sjaak: An empirical characterization of event sourced systems and their schema evolution — Lessons from industry. In: *Journal of Systems and Software* 178 (2021), S. 110970. – URL https://www.sciencedirect.com/science/article/pii/S0164121221000674. – ISSN 0164-1212

[Richards 2015] Richards, Mark: *Software architecture patterns*. Bd. 4. O'Reilly Media, Incorporated 1005 Gravenstein Highway North, Sebastopol, CA ..., 2015

[Richardson 2018] Richardson, Chris: *Microservices patterns: with examples in Java*. Simon and Schuster, 2018. – URL https://books.google.de/books?id=QTgzEAAAQBAJ

[Tagliaferri 2023] Tagliaferri, Costanza: Programming Languages Ranking: Top 9 in 2023 - DistantJob - Remote Recruitment Agency. (2023), 01. – URL https://distantjob.com/blog/programming-languages-rank. – (Accessed on 01/31/2023)

[Wiggins 2017] Wiggins, Adam: The Twelve-Factor App. (2017). – URL https://12factor.net/. – (Accessed on 02/15/2023)

[Young 2010] Young, Greg: CQRS documents by greg young. In: *Young* 56 (2010). – URL `https://cqrs.files.wordpress.com/2010/11/cqrs_documents.pdf`

A Anhang

Glossary

RESTfull . TODO

Angular . TODO

Audit Browser ... more under chapter 7. TODO

Audit Component ... more under chapter 6. TODO

Auditing 2.0 . TODO

BaFin the Federal Financial Supervisory Authority (short BaFin in german).

Continuous Delivery . TODO

Continuous Integration . TODO

Custom Resource Definition . TODO

Docker . TODO

Emissary-Ingress . TODO

Emissary-Ingress . TODO

Event-Sourcing ... more under chapter 4. TODO

EventFormatter An event formatter is responsible for creating a human-readable representation of a given event.

EventStore . TODO

GO . TODO

gRPC-Web . TODO

Helm . TODO

HPACK . TODO

idempotent Idempotency is a property of a system or operation where the result of performing that operation multiple times is the same as performing it once. In other words, if an idempotent operation is performed multiple times, the end result is the same as performing it only once..

Javascript . TODO

Kind . TODO

Kubectl . TODO

Kubernetes also known as K8s, is an open-source system for automating deployment, scaling, and management of containerized applications.

[Kubernetes, 2014] .

Make . TODO

MonoGUI is a Graphical User Interface for Monoskope (m8).

Monoskope (short m8 spelled "mate") implements the management and operation of tenants, users and their roles in a Kubernetes multi-cloud multi-cluster environment..

Open-ID Connect . TODO

Public Key Infrastructure . TODO

React . TODO

Step-cli . TODO

The Twelve-Faktor App . TODO

Typescript . TODO

Version Control . TODO

Vue.js . TODO

Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original