

BACHELOR THESIS  
Hani Alshikh

# Evaluation and use of Event-Sourcing for audit logging

---

Faculty of Engineering and Computer Science  
Department Computer Science

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-Technologies (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

<b>List of Figures</b>	<b>vii</b>
<b>List of Tables</b>	<b>viii</b>
<b>Abbreviations</b>	<b>x</b>
<b>Shell</b>	<b>xii</b>
<b>1 Acknowledgments</b>	<b>1</b>
<b>2 Introduction</b>	<b>2</b>
<b>3 Audit</b>	<b>3</b>
3.1 IT Audit . . . . .	4
3.2 IT Auditing in Context . . . . .	5
3.3 Audit log . . . . .	6
3.4 Auditing 2.0 . . . . .	7
3.4.1 Business Provenance . . . . .	8
3.4.2 Process Mining . . . . .	8
3.4.3 Challenges . . . . .	9
<b>4 Event Sourcing</b>	<b>10</b>
4.1 Terminology . . . . .	11
4.1.1 The Event in Event-Sourcing . . . . .	11
4.1.2 Event Store . . . . .	11
4.1.3 Streams . . . . .	12
4.1.4 Projections . . . . .	12
4.1.5 Snapshots . . . . .	12
4.2 The Core Pattern . . . . .	13
4.3 Event-Sourced Architecture . . . . .	13

4.4	Challenges . . . . .	15
4.4.1	Event Storage . . . . .	15
4.4.2	Event Schema Evolution . . . . .	15
4.4.3	Deleting Data is Tricky . . . . .	16
4.4.4	Querying the Event Store is Challenging. . . . .	16
4.4.5	Eventual Consistency . . . . .	17
4.5	Benefits . . . . .	17
<b>5</b>	<b>Software Architecture and Auditing</b>	<b>19</b>
5.1	Implementing audit logging . . . . .	19
5.1.1	Audit logging code in business logic . . . . .	20
5.1.2	Aspect-Oriented programming . . . . .	21
5.1.3	Event Sourcing . . . . .	21
5.2	traditional persistence . . . . .	21
5.2.1	Problems . . . . .	22
5.3	Event Sourcing . . . . .	23
5.3.1	Database Driven . . . . .	24
5.4	Blockchain . . . . .	24
5.5	Auditing 2.0 . . . . .	25
5.5.1	Auditing 2.0 Framework . . . . .	25
<b>6</b>	<b>Audit Component</b>	<b>28</b>
6.1	Monoskope . . . . .	28
6.2	Requirements . . . . .	29
6.2.1	Use-Cases . . . . .	29
6.2.2	Architectural Constraints . . . . .	30
6.3	System Design . . . . .	32
6.3.1	Scope and Context . . . . .	32
6.3.2	Solution Strategy . . . . .	33
6.3.3	Building Block View . . . . .	34
6.3.4	Runtime View . . . . .	37
6.3.5	Deployment View . . . . .	37
6.4	Design Decisions . . . . .	37
6.4.1	DD01: Registry pattern . . . . .	37
6.5	Technical Decisions . . . . .	37
6.5.1	TD01: Limit get user actions query to max 1 year . . . . .	37

<b>7</b>	<b>Audit Browser</b>	<b>38</b>
7.1	Requirements . . . . .	39
7.1.1	Use-Cases . . . . .	40
7.1.2	Architectural Constraints . . . . .	41
7.2	System Design . . . . .	42
7.2.1	Scope and Context . . . . .	42
7.2.2	Solution Strategy . . . . .	44
7.2.3	Building Block View . . . . .	45
7.2.4	Runtime View . . . . .	53
7.3	Design Decisions . . . . .	56
7.3.1	DD01: google Remote Procedure Call (gRPC) Client-Server Communication . . . . .	56
7.4	Technical Decisions . . . . .	60
7.4.1	TD01: Javascript (JS) and React . . . . .	60
7.4.2	TD02: Headless Use-Cases Implementation . . . . .	61
<b>8</b>	<b>Installation and Configuration</b>	<b>63</b>
8.0.1	Deployment View . . . . .	63
8.1	Test Run . . . . .	64
<b>9</b>	<b>Conclusion</b>	<b>69</b>
	<b>Bibliography</b>	<b>71</b>
<b>A</b>	<b>Anhang</b>	<b>74</b>
	<b>Glossary</b>	<b>75</b>
	Declaration of Autorship . . . . .	78

# List of Figures

3.1	IT auditing commonality with other types of audit [Gantz, 2014]	4
3.2	IT audit activities and scopes [Gantz, 2014]	5
6.1	Audit Component Business Context Diagram	32
6.2	Audit Component Technical Context Diagram	33
6.3	Overall System Component Diagram	34
6.4	Audit Component Diagram	35
7.1	Audit Browser business context diagram	42
7.2	Audit Browser technical context diagram	43
7.3	Audit Browser overall system component diagram	45
7.4	Audit Browser MonoGUI gRPC class diagram	47
7.5	Audit Browser scenes audit class diagram	48
7.6	Audit Browser scenes auth class diagram	49
7.7	Audit Browser usecases audit class diagram	51
7.8	Audit Browser usecases auth class diagram	52
7.9	Audit Browser UC01 sequence diagram	53
7.10	Audit Browser UC02 sequence diagram	54
7.11	Audit Browser UC02 sign in sequence diagram	54
7.12	Audit Browser UC02 auth state machine	55
7.13	Audit Browser UC03 sequence diagram	56
7.14	HyperText Transfer Protocol (HTTP)/1.1 compared to HTTP/2	57
7.15	gRPC-Web proxy to allow browser gRPC support [Brandhorst, 2019]	58
7.16	gRPC-Gateway architecture diagram [gRPC Gateway Authores, 2023]	59
7.17	Top programming languages - rankings in comparison [Tagliaferri, 2023]	60
8.1	Deployment view diagram	64

# List of Tables

3.1	Examples of Internal Controls Categorized by Type and Purpose [Gantz, 2014] . . . . .	6
6.1	Audit component derived use-cases . . . . .	30
6.2	Audit component technical constraints . . . . .	30
6.3	Audit component organisational constraints . . . . .	31
6.4	Audit component solution strategy . . . . .	33
6.5	Audit component overall system contained building blocks white box . . .	34
6.6	Audit component black box . . . . .	35
6.7	Audit component contained building blocks white box . . . . .	36
6.8	Event component black box . . . . .	36
7.1	Audit Browser derived use-cases . . . . .	40
7.2	Audit Browser technical constraints . . . . .	41
7.3	Audit Browser organisational constraints . . . . .	41
7.4	Audit Browser solution strategy . . . . .	44
7.5	Audit Browser MonoGUI contained building blocks black box . . . . .	46
7.6	Audit Browser scenes contained building blocks black box . . . . .	46
7.7	Audit Browser UseCases contained building blocks black box . . . . .	46
7.8	Audit Browser Application Programming Interface (API) contained building blocks black box . . . . .	47
7.9	Audit Browser MonoGUI gRPC class diagram . . . . .	47
7.10	Audit Browser scenes audit class diagram . . . . .	48
7.11	Audit Browser scenes auth class diagram . . . . .	50
7.12	Audit Browser usecases audit class diagram . . . . .	51
7.13	Audit Browser usecases auth class diagram . . . . .	52
8.1	Installation required tools . . . . .	65



8.2	Installation automated tools . . . . .	66
-----	--	----

# Abbreviations

**AI** Artificial Intelligence.

**API** Application Programming Interface.

**CD** Continuous Delivery.

**CI** Continuous Integration.

**CQRS** Command Query Responsibility Segregation.

**CRD** Custom Resource Definition.

**CSV** Comma Separated Values.

**DDD** Domain-Driven Design.

**ERP** Enterprise Resource Planning.

**GDPR** General Data Protection Regulation.

**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 Locator.

**UX** User Experince.

**XML** Extensible Markup Language.

# Shell

8.1	Deploy with custom command . . . . .	65
8.2	Deploy all resources to a local cluster . . . . .	67
8.3	Trust Monoskope (m8) domain certificate . . . . .	67
8.4	Update hosts file . . . . .	67
8.5	Create port-forwards to route local request . . . . .	67
8.6	Populate the Event Store with some mock . . . . .	68

# 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.

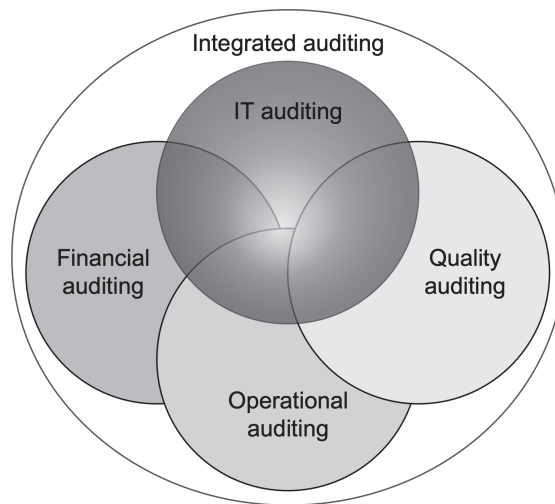


Figure 3.1: IT auditing commonality with other types of audit [Gantz, 2014]

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].

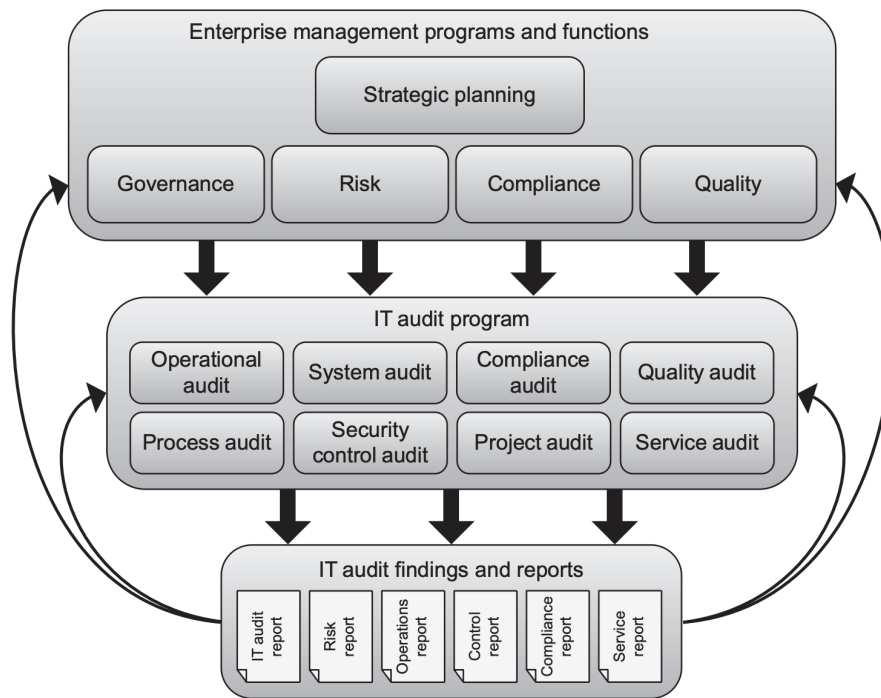


Figure 3.2: IT audit activities and scopes [Gantz, 2014]

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 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

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.

Move to  
Conclu-  
sion or  
evalu-  
ate if this  
is really  
relevant?

## 4 Event Sourcing

Event-Sourcing is a software architecture pattern that was originally established by [Fowler, 2005] and is gaining popularity as an alternative to traditional database systems. Event-Sourcing stores data in an append-only log. It is part of a wider ecosystem of design patterns that work together in various ways to allow developers to create the most effective architecture for their needs.

Traditionally applications use the current state to answer various queries, however in most cases such quires fails, when the path leading to the current state is required.

Event-Sourcing ensures that all changes to application state are stored as a sequence of events. Writing an event to the log is one single, therefore atomic, operation. These events can be aggregated in multiple ways not only can they be quired, reconstructing past states is also possible. Event-Sourcing can be thought of as the version-control way of working with systems state. Each change is committed and can be traced back and/or revisited.

Event-Sourcing is a superior pattern for auditing compared to other architectures. Just like accounting transactions events are never deleted nor modified. Event-Sourcing provides a complete and immutable record of all actions taken within a system, allowing for thorough and accurate auditing. Unlike traditional architectures that rely on snapshots of data at a specific point in time, Event-Sourcing captures every individual event and action, providing a more comprehensive and transparent record. This allows for more effective detection and investigation of potential issues or irregularities, ensuring the integrity and reliability of the auditing process. Additionally, the use of Event-Sourcing allows for greater flexibility and scalability in auditing, as it allows for easy replay and reconstruction of past states. Alternative histories can also be explored by injecting hypothetical events when reconstructing the state.

### 4.1 Terminology

#### 4.1.1 The Event in Event-Sourcing

While Event-Sourcing and Event-Driven might sound similar they differ in multiple aspects. The events in Event-Sourcing, as opposed to general event-driven architectures are stored as an append-only log of all state changes. The following are the two main key characteristics as derived by [Fowler, 2017] separating Event-Sourcing from other event-driven approaches.

1. Events in Event-Sourcing systems are stored as the state of the system. Other approaches use events to communicate and sometimes to passively send commands to the recipients. While the communication aspect in Event-Sourcing is present, it fail second to the usage as the state.
2. The second characteristic is, that events in Event-Sourcing are used as the source of truth. Unlike other patterns that uses events to carry state changes. Event-Sourcing uses the events to derive the state change.

An 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 [Richardson, 2018]. Events typically also have metadata, such as the event ID and a timestamp. The metadata can be part of the event object or, alternatively, in a separate envelope object. It might also have the identity of the entity who made the change. Such details will come in handy when utilising process mining as described in section 3.4.2. Examples of such events are showcased later in chapter 6.

#### 4.1.2 Event Store

The event store is the database storing the events. The event store is the principal source of truth, and the system state is purely derived from it [Fowler, 2017]. Thus the database must satisfy the following criteria:

- Events are immutable.
- New events are appended to the previous event.
- Events are stored in chronological order.

### 4.1.3 Streams

The set of events comprising a particular domain object are called a stream. Event streams are the the source of truth of all domain objects in a system and contain the full history of changes. Retrieving the state of a domain object consist of reading all events in a stream and applying them one by one in the order of appearance.

A stream have a unique identifier present in all corresponding events. Detecting concurrency issues and insuring ordinality require uniq numerical value, that can be used as a form of versioning.

### 4.1.4 Projections

Projections provide a view of the underlying stream as a form of a transient state. They represent the logic of translating the source events into a representation of the object state.

In many applications it is more common to request recent application states, if so a faster alternative is to store the current application projections and upsert on new changes.

### 4.1.5 Snapshots

snapshots create a working copy of a state that can be updated without replaying all events from scratch every time.

A common mistake is the assumption, that accessing the Event Store is required to rebuild the state on each change. Accessing the Event Store should generally be reserved to determining useful information. Only elements that really need the information in the event log should have to access it [Fowler, 2017].

Snapshots can also be used to mark a state just like a tag in version-control systems. Depending on the use-case, this becomes helpful if the event log became too large in size.



### 4.2 The Core Pattern

The fundamental idea of Event-Sourcing is that of ensuring every change to the state of a system is captured in an event object, and that these event objects are themselves stored in the sequence they were applied for the same lifetime as the system state itself.

This leads to a number of facilities that can be built on top of the event log [Fowler, 2005]:

- **Complete Rebuild:** the system state can be discarded completely and rebuilt by re-running the events from the event log.
- **Temporal Query:** the application state at any point in time can be determined. Notionally this is done by starting with a blank state and rerunning the events up to a particular time or event.
- **Event Replay:** the consequences of a corrupting event can be computed by reversing it and later events and then replaying the new event and later events. The same technique can handle events received in the wrong sequence - a common problem with systems that communicate with asynchronous messaging.

A common example of an application that uses Event-Sourcing is version control. Such a system uses temporal queries quite often [Fowler, 2005]. Recently, the Event-Sourcing pattern has become a popular answer to the challenges of complex, mission-critical, scalable systems [Overeem u. a., 2021]. Examples of organizations that apply Event-Sourcing are Netflix [Avery und Reta, 2017], and Walmart's Jet.com [Gorodinski, 2017], with the goal of creating scalable and reliable critical systems.

The main introduction to the inner workings of Event-Sourcing by [Fowler, 2005] gives a clear impression on the general implementation of the different building blocks of the pattern. More details and advanced concepts are covered by [Richardson, 2018].

### 4.3 Event-Sourced Architecture

Using Event-Sourcing as a design pattern within a wider architecture allows for the inclusion of other design patterns in the system that are the most suitable for the needs

of the domain. For example, Domain-Driven Design (DDD) in combination with Event-Sourcing and Command Query Responsibility Segregation (CQRS) lay the basis for a scalable architecture, that can be used in a variety of systems or in conjunction with many other patterns depending on the specific needs.

### **Domain-Driven Design (DDD)**

Using DDD with Event-Sourcing is not mandatory. However, in Event-Sourcing events are modeled as first-class objects and closely resemble real world business processes. The better business processes are understood, the more precise the business information will be in the events and thus the audit log. Concepts like speaking the same language as the business and using events as a design tool when modelling a system are advocated by DDD [Evans, 2004] as well, which makes Event-Sourcing a natural fit for DDD.

### **Command Query Responsibility Segregation (CQRS)**

Event-Sourcing is discussed in the context of CQRS, a pattern strongly related to Event-Sourcing.

CQRS and Event-Sourcing have a symbiotic relationship. CQRS allows Event-Sourcing to be used as the data storage mechanism for the domain. One of the largest issues when using Event-Sourcing is that you cannot ask the system a query such as “Give me all users whose first names are ‘Greg’”. This is due to not having a representation of current state. With CQRS the only query that exists within the domain is GetById which is supported with Event-Sourcing [Young, 2010].

The loosely coupled nature of CQRS combined with 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.

The justification for CQRS is that in complex domains, a single model to handle both reads and writes gets too complicated, and can be simplified by separating the models. This is particularly appealing when difference in access patterns is observed, such as lots

of reads and very few writes. However the gain for using CQRS has to be balanced against the additional complexity of having separate models [Fowler, 2017].

While using CQRS is also not mandatory the strong relation and added benefits do justify the cost. Aggregating the Event Store to satisfy the Audit Component use-cases is an example, where CQRS shines. More on that in chapter 6

## 4.4 Challenges

Event-Sourcing does have its problems. Replaying events becomes problematic when results depend on interactions with outside systems. Dealing with changes in the schema of events over time is not an easy task. Event processing adds complexity to the system (mostly when improperly done) [Fowler, 2017]

### 4.4.1 Event Storage

Event-Sourcing enables the reconstruction of arbitrary past states. However, an entirely unbounded log size can conflict with other system requirements. As discussed in section 4.1.5, snapshots offer a mitigation when the history is not relevant anymore for further processing as for example described in section 3.4.2.

Other approaches like log pruning are discussed by [Erb u. a., 2018] including an assessment of the impact of such mechanisms on state reconstructibility.

### 4.4.2 Event Schema Evolution

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 is a real risk that aggregates may become bloated with code to deal with all the different versions [Richardson, 2018].

Upgrading events to the latest version when they are loaded from the Event Store insures, that the system only ever deals with the current event schema. A component commonly called an `upcaster` as described by [Richardson, 2018] updates individual events from an old version to a newer one.

### 4.4.3 Deleting Data is Tricky

One of the key characteristics of Event-Sourcing is the immutable event log. The traditional way to delete data is to do a soft delete [Richardson, 2018]. The system 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.

However complying with General Data Protection Regulation (GDPR) grants individuals the right to erasure [EU, 2016]. An application must have the ability to forget a user's personal information. One way of doing so is to ensure, that user data are encapsulated in an independent data object, that can either be encrypted by per user encryption key, which is discarded on request or iteratively overwritten, which is against the immutability aspect of the event.

Some form of anonymization and removal of information are two techniques mentioned by the engineers of the study conducted by [Overeem u. a., 2021]. 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.

### 4.4.4 Querying the Event Store is Challenging.

As discussed in section 4.3 implementing the core Event-Sourcing pattern alone comes with the challenge of query complications. As put by [Richardson, 2018]:

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.

Which highlight the benefits of implementing a pattern like CQRS, but also a big limitation of Event-Sourcing, especially when the produced event log need to be utilized for auditing.

### 4.4.5 Eventual Consistency

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 [Overeem u. a., 2021], which leads to difficulties such as returning items to a client that in fact are already deleted. The reader systems are liable to be out of sync with the master (and each other) due to differences in timing with event propagation.

Getting people to understand eventual consistency is not easy. Eventual consistency forces developers to rethink the basic interactions of the user with the system.

However, eventual consistency is a weaker form of consistency. 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 happen. A system with a large delay is unfeasible, because in that case queries will often return data that does not reflect the latest changes

## 4.5 Benefits

Event-Sourcing introduces a lot of challenges and complexity. As with any other pattern Event-Sourcing is not always the solution. However a study conducted on 25 engineers with different roles and experiences reveled, that all systems under study benefit from Event-Sourcing. Flexibility, debug-ability, reliability and auditability are common rational given for using Event-Sourcing [Overeem u. a., 2021].

It is obvious, that the most repeated and almost always mentioned benefit of Event-Sourcing is the events serialization into an audit log, that satisfies all characteristics as defined in chapter 3. Beside all other benefits, this makes Event-Sourcing an optimized pattern to implement Auditing 2.0 as discussed in the next chapter section 5.5.1

Event-Sourcing also comes with debug-ability and customer support advantages. As put by [Fowler, 2005]:

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.

Providing such capability means exposing the audit trail to the support group so they can walk through a user's interaction. [Fowler, 2005] acknowledge, that using Event-Sourcing is not a requirement for such capabilities. Regular logging mechanisms are more than capable of achieving such results. However this assumes a logging infrastructure and utilisation that is capable of providing such information at ease, which is not always the case. More on this in chapter 5

Furthermore, Event-Sourcing allows developers and auditors to consider multiple timelines (analogous to branching in version control systems) and recreate historic states or explore alternative histories by injecting hypothetical events when replaying. This means that even if the current tarnsaite state of the data has been corrupted or lost, it is still possible to recreate it from the event log. In contrast, traditional systems rely on the current state of the data. If this state is lost or corrupted, there is no way to recover the data's history without extensive backups.

Having the entire history of the state comes with the advantage of preserving the context, which allows for evidence based explanations of when and why something happened. Events can also be analysed for patterns in usage. Such information is impossible to extract from a store that only persists the latest state of the data.

As long as the stores criteria are meet a diverse range of databases 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 system requires.

Another advantage of Event-Sourcing is that it allows for easy implementation of fine-grained access controls. Because each event is stored as an individual record, it is possible to apply different access controls to different events, allowing for more precise control over who can access the data. This is in contrast to traditional systems, which typically apply access controls at the level of the entire database or table.

Event-Sourcing can be a key element of a system, and that system can be as simple or as complex as the business domain requires it to be. It is useful to consider putting an event-sourced system in a part of the architecture that requires the preservation of context for all events, as this is where Event-Sourcing is most effective.

## 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 its structure [Bass u. a., 2003].

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

### 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 its 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 through 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.

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

### 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

Firstly, event sourcing provides a more detailed and comprehensive audit trail compared to traditional architectures. In traditional architectures, the current state of an application is typically stored in a database, and any changes to this state are recorded as updates to the database. This means that the only information available for auditing is the current state of the application and any changes made to it. In contrast, event sourcing records every change to the state of the application as an individual event, providing a complete history of the application's behavior and allowing for a much more detailed audit trail.

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]

Thirdly, event sourcing provides a more robust and flexible approach to auditing. In traditional architectures, the audit trail is typically limited to the information stored in the database, and any changes to the database schema or data model can affect the audit trail. In contrast, event sourcing uses an event log as the source of truth, which is separate from the application's data model and is not affected by changes to the data model. This means that the audit trail is more robust and flexible, and is not subject to the same limitations as traditional approaches.

### 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.

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.

### 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).

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.

## 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.

m8 is an event-sourced system...

rewrite  
m8  
overview

TODO



## 6.2 Requirements

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 must have a human-readable representation.

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

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

clarify  
how this  
should be  
written?  
meeting,  
discussion  
parties, etc...

add chapter  
reference

### 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 must 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

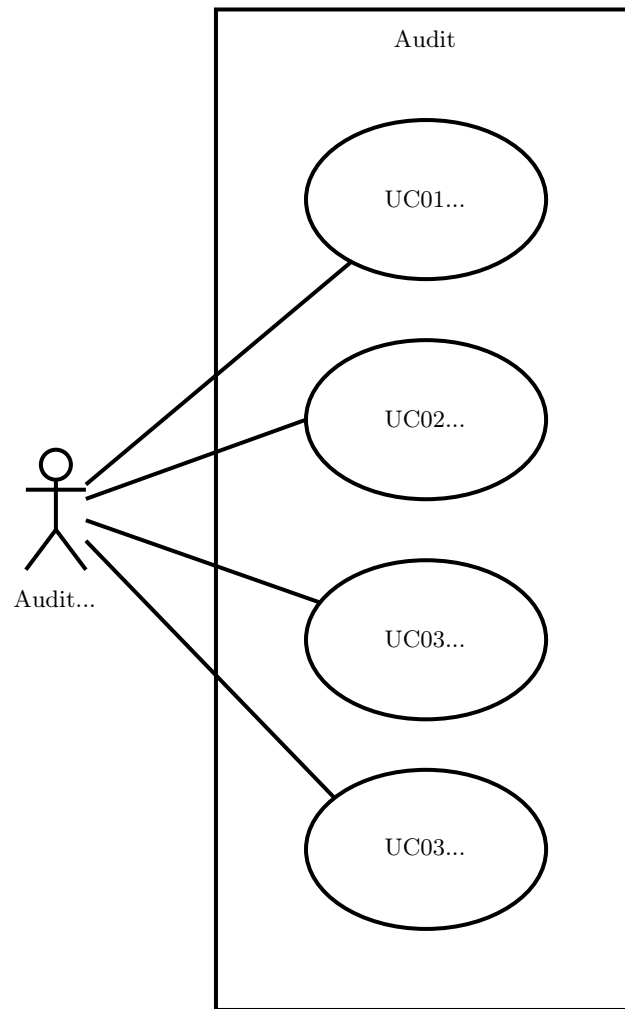


Figure 6.1: Audit Component Business Context Diagram

Technical Context

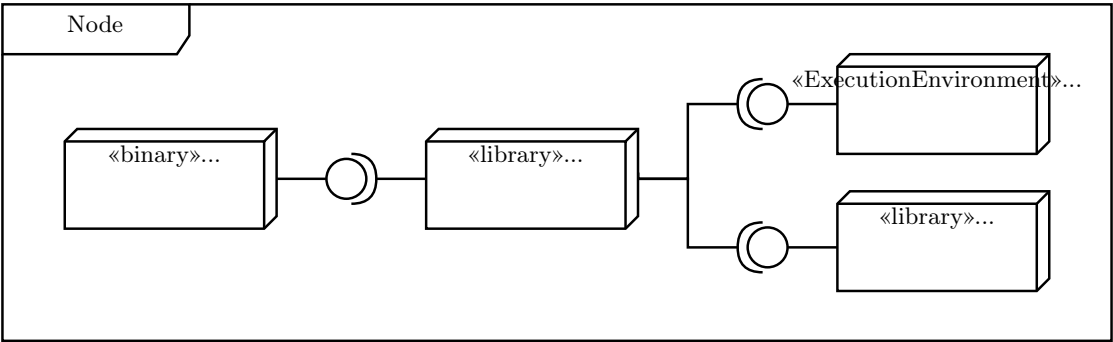


Figure 6.2: Audit Component Technical Context Diagram

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

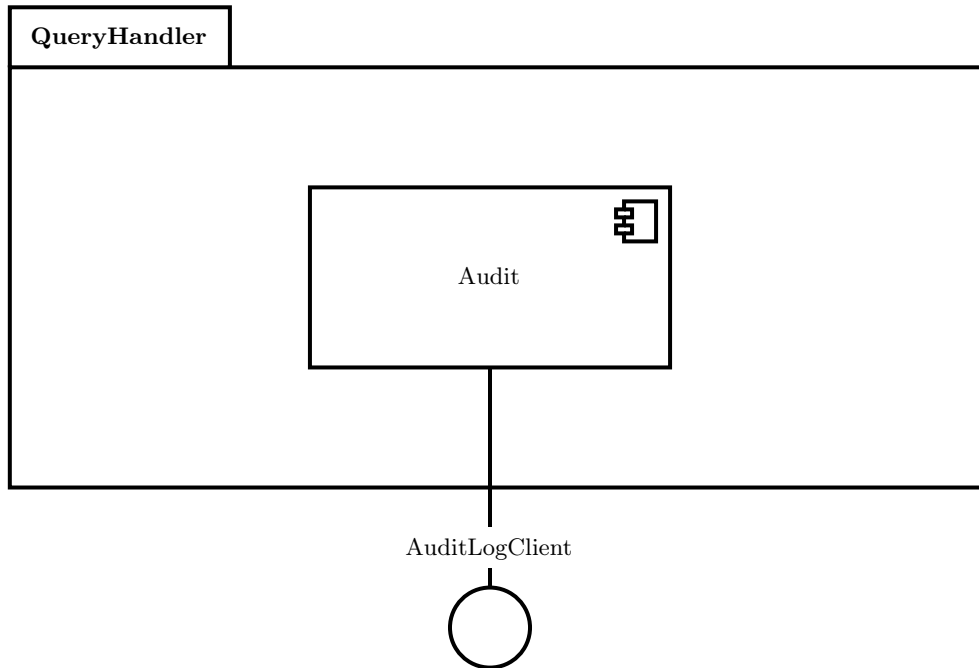


Figure 6.3: Overall System Component Diagram

#### 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

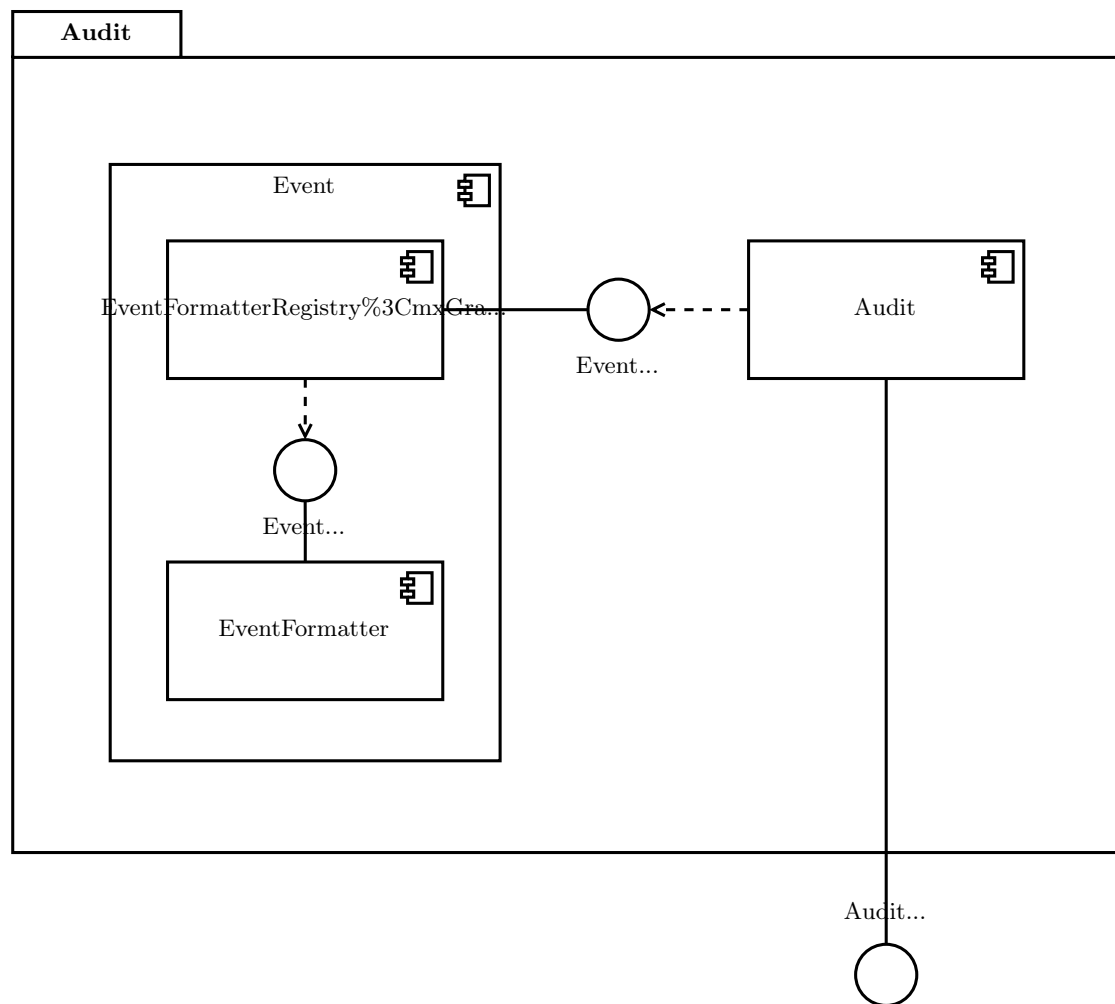


Figure 6.4: Audit Component Diagram

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 compliant 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 partitonend 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 must 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

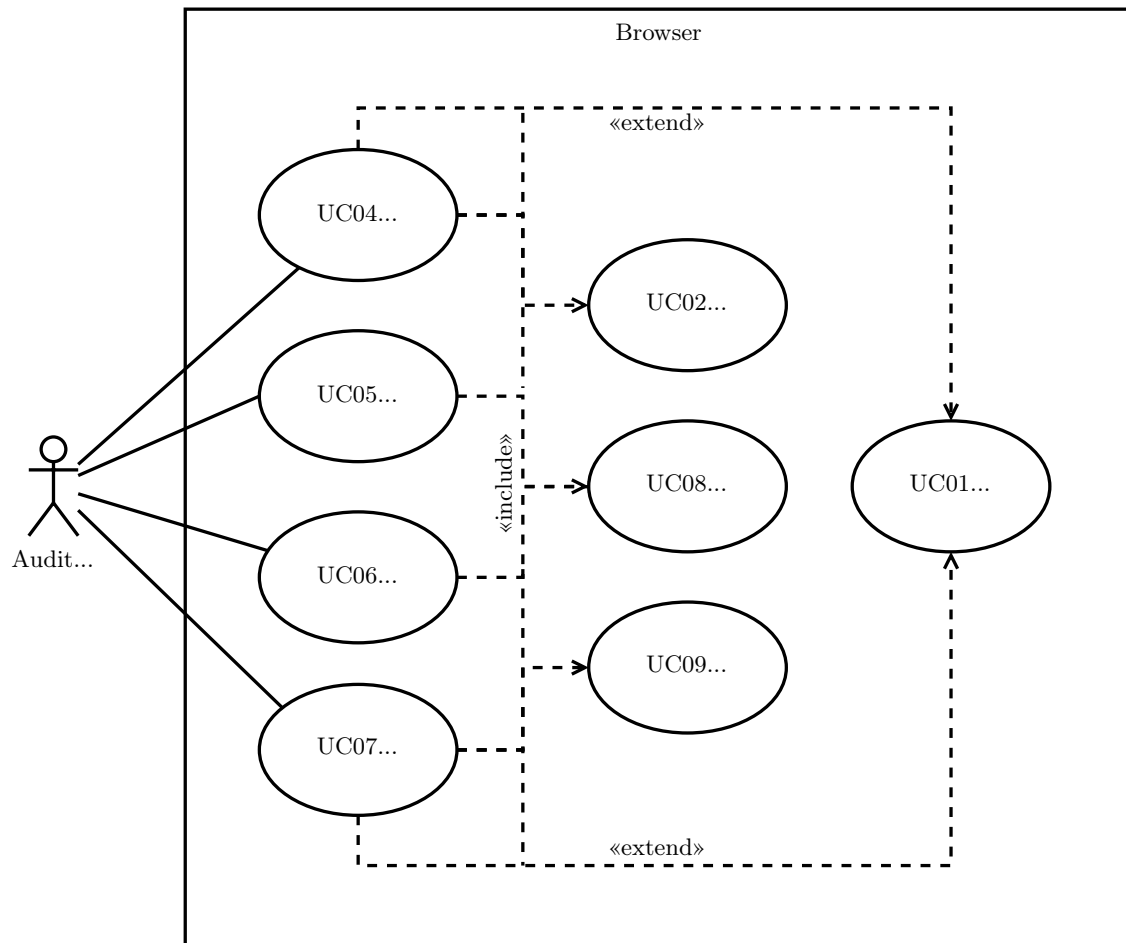


Figure 7.1: Audit Browser business context diagram

## Technical Context

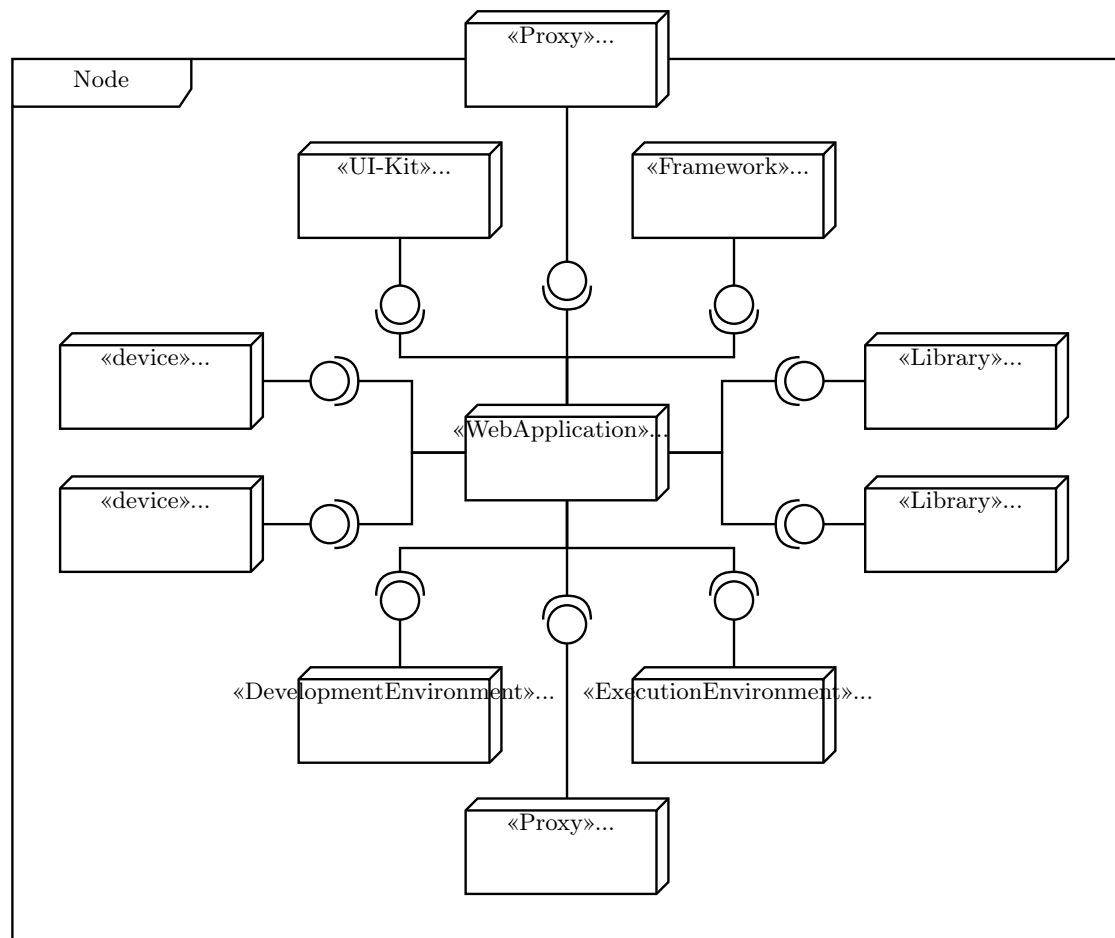


Figure 7.2: Audit Browser technical context diagram

## 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 authentication 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

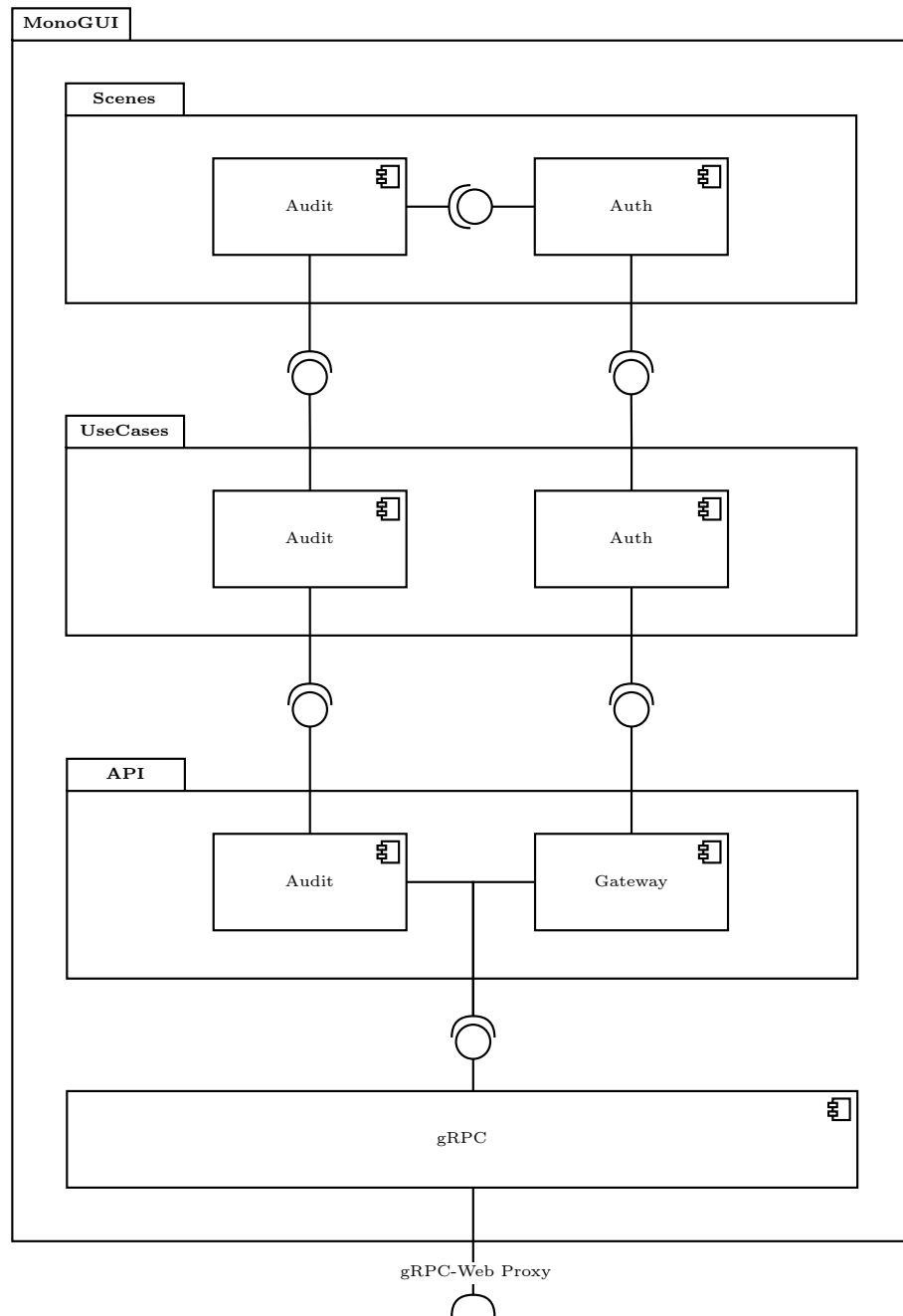


Figure 7.3: Audit Browser overall system component diagram

## Contained Building Blocks

### MonoGUI

Component	Description
gRPC	handles communication 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

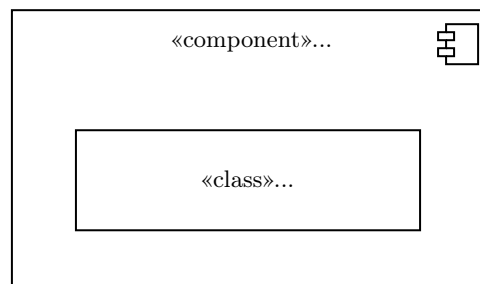
**Level 2****MonoGUI White Box****gRPC**

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

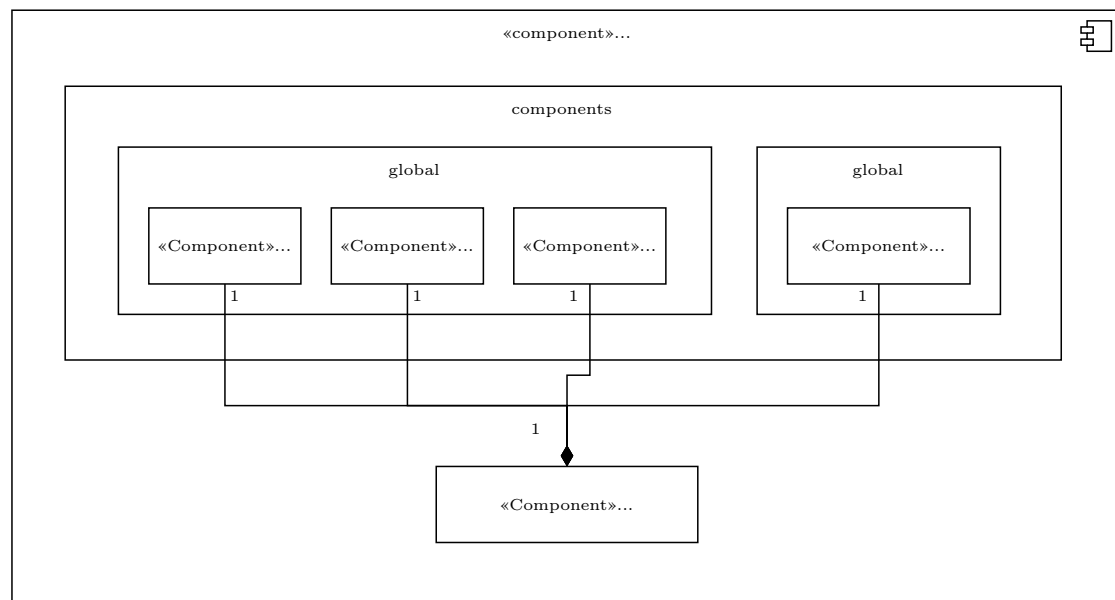


Figure 7.5: Audit Browser scenes audit class diagram

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

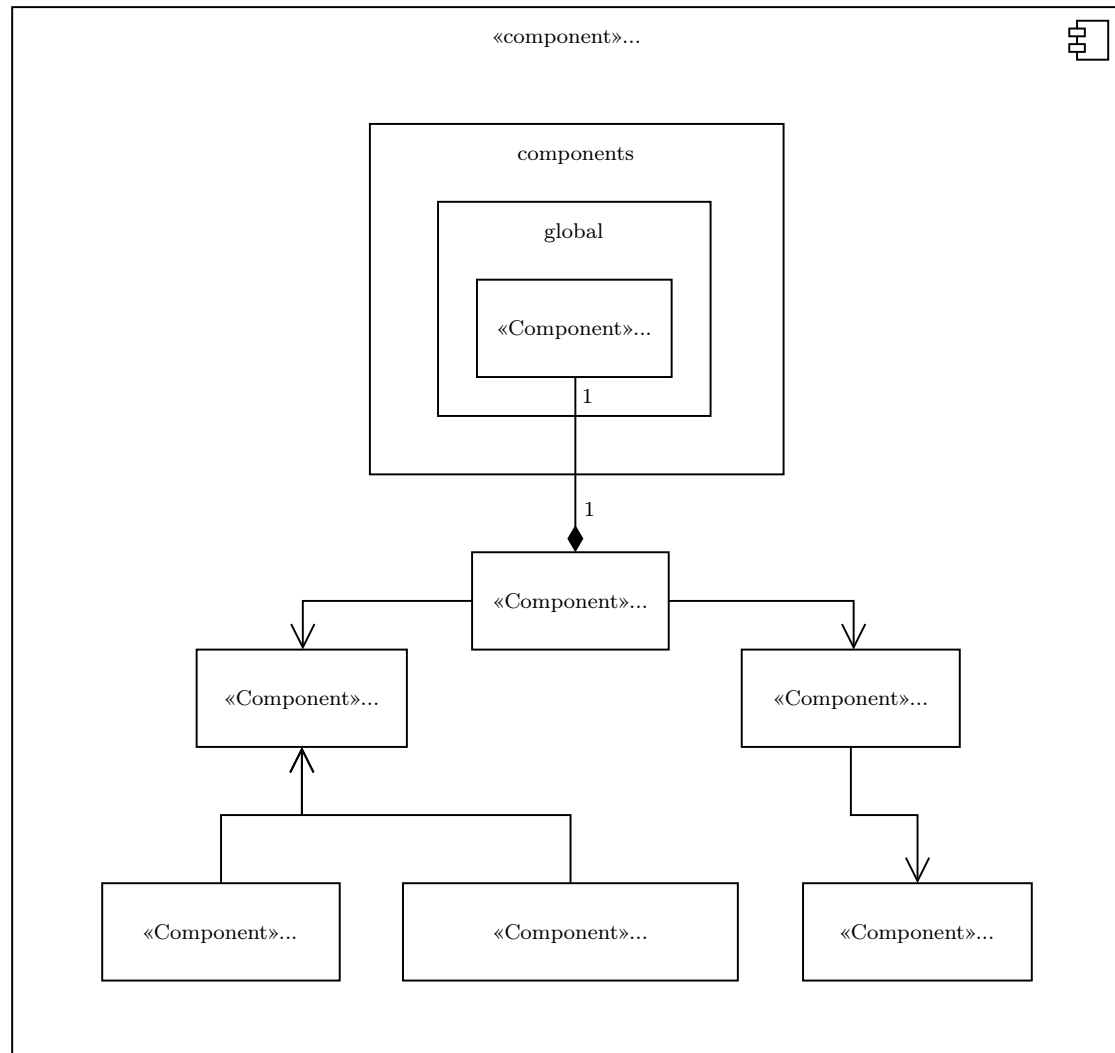
**Auth**

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

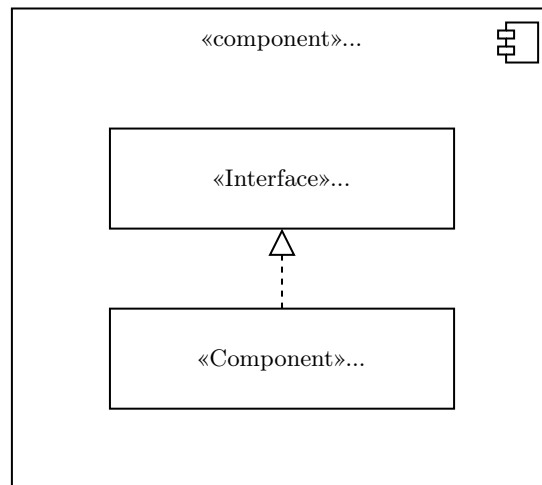


Figure 7.7: Audit Browser usecases audit class diagram

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

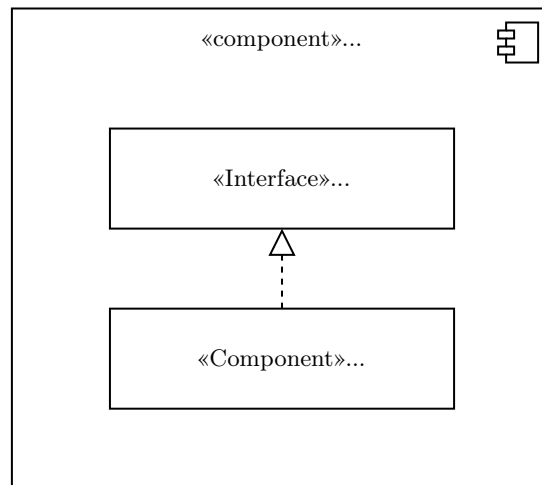


Figure 7.8: Audit Browser usecases auth class diagram

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

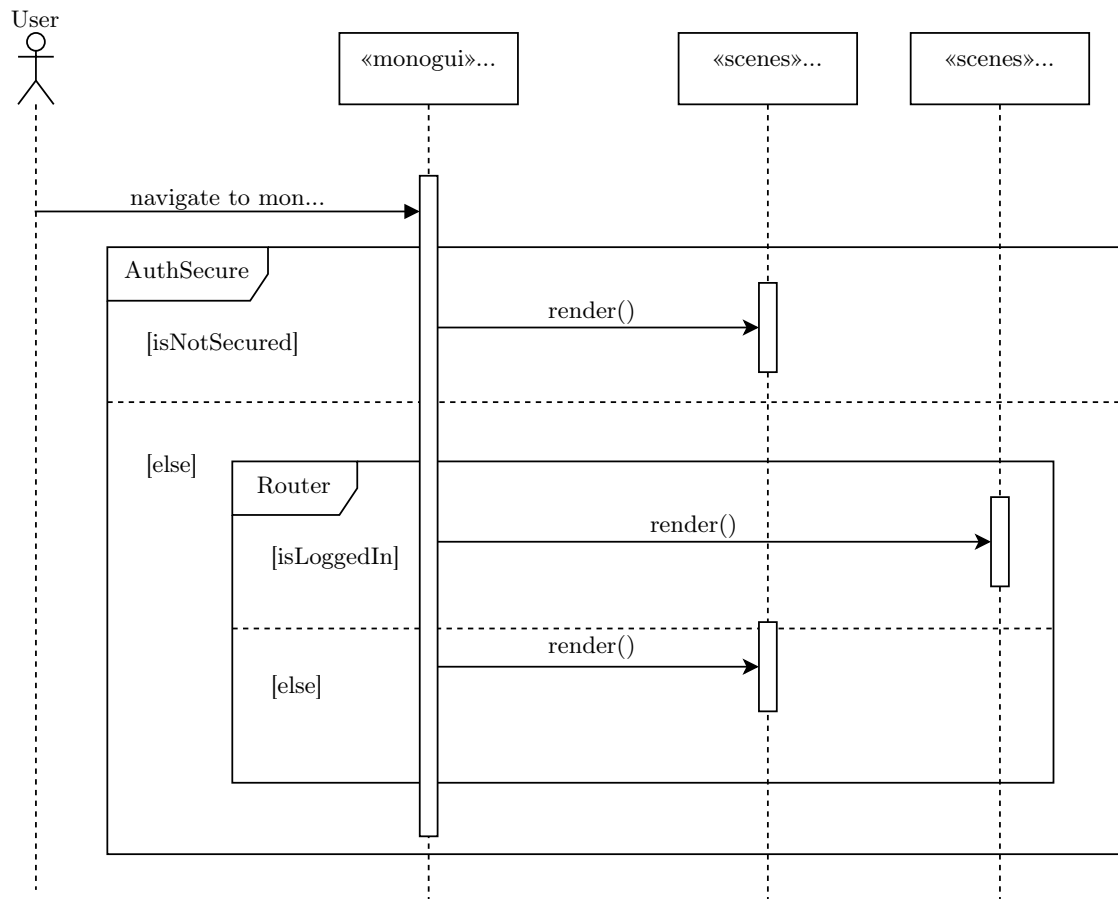


Figure 7.9: Audit Browser UC01 sequence diagram

## UC02: OIDC authentication

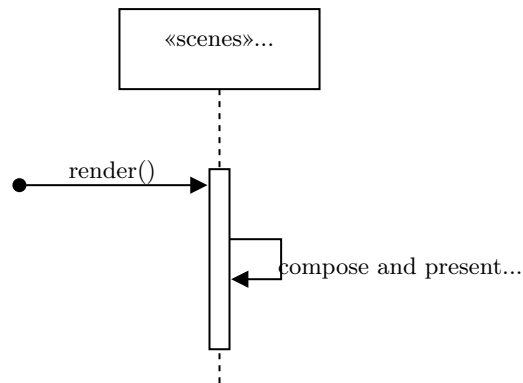


Figure 7.10: Audit Browser UC02 sequence diagram

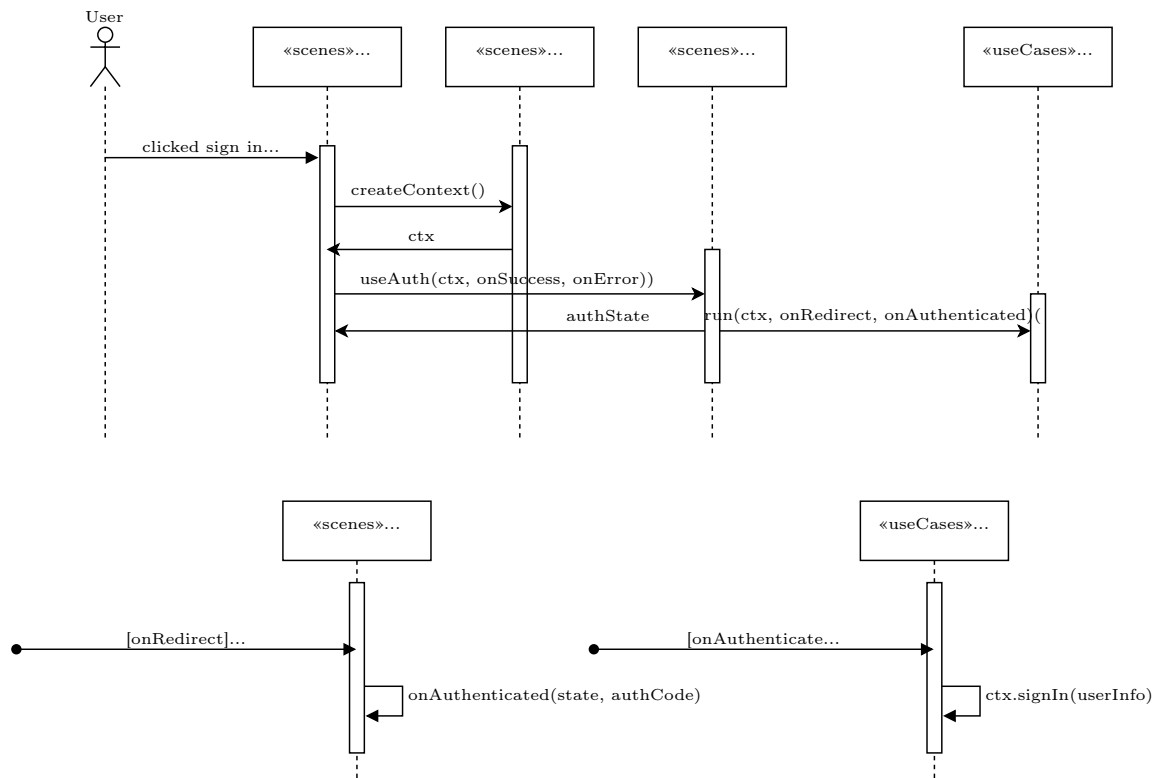


Figure 7.11: Audit Browser UC02 sign in sequence diagram

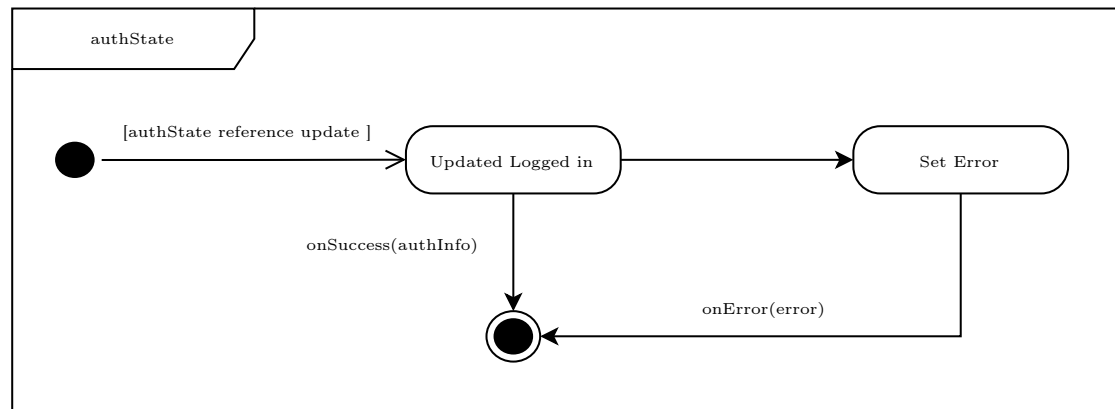


Figure 7.12: Audit Browser UC02 auth state machine

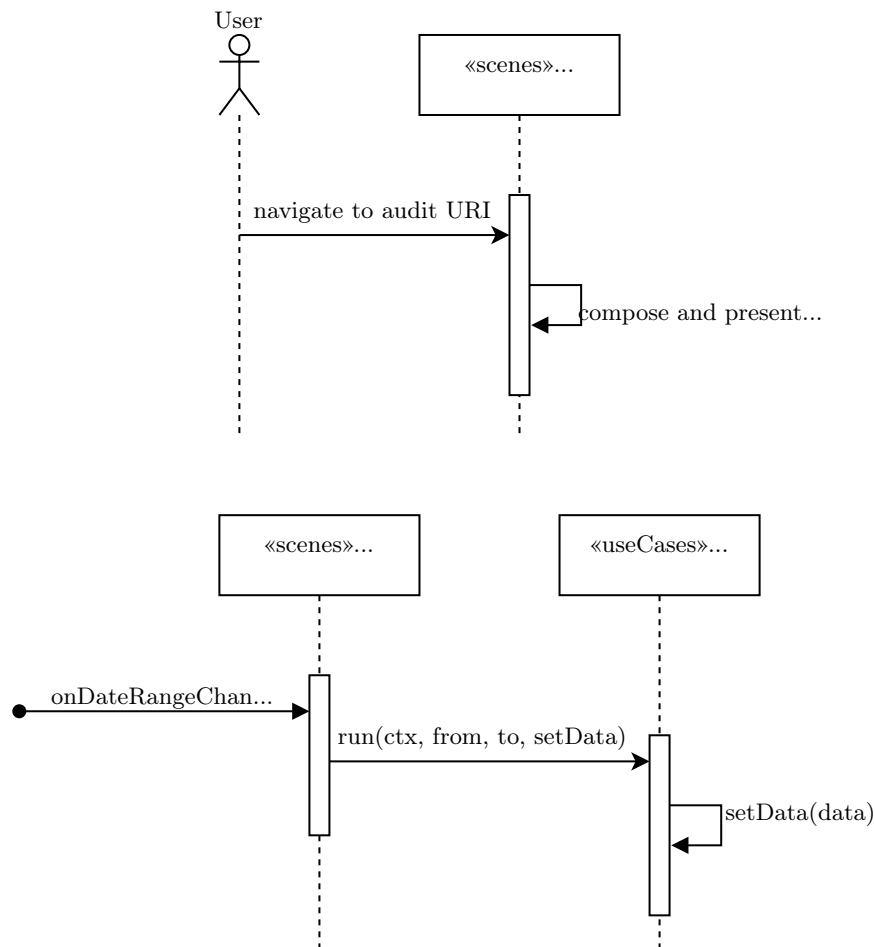
**UC03: Audit-log for date-range report**

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.

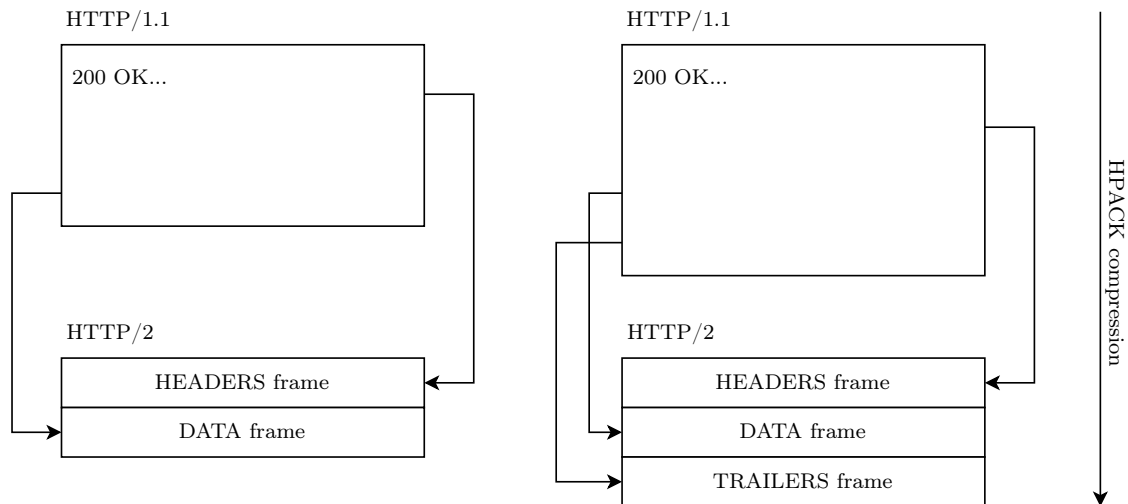


Figure 7.14: HTTP/1.1 compared to HTTP/2

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].

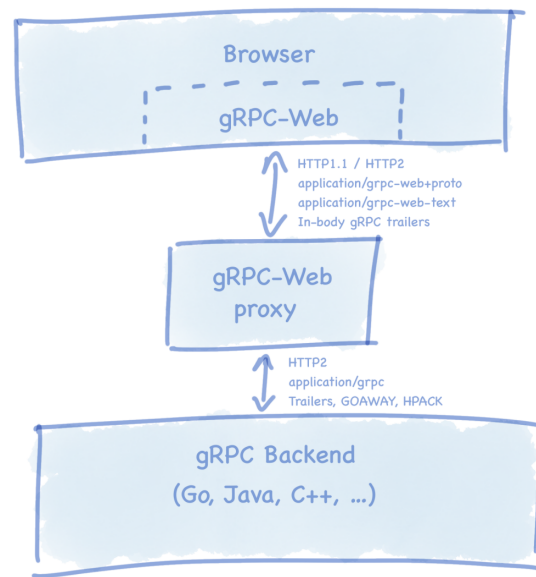


Figure 7.15: gRPC-Web proxy to allow browser gRPC support [Brandhorst, 2019]

### Why stick to gRPC?

Deciding whether 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 gains 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]

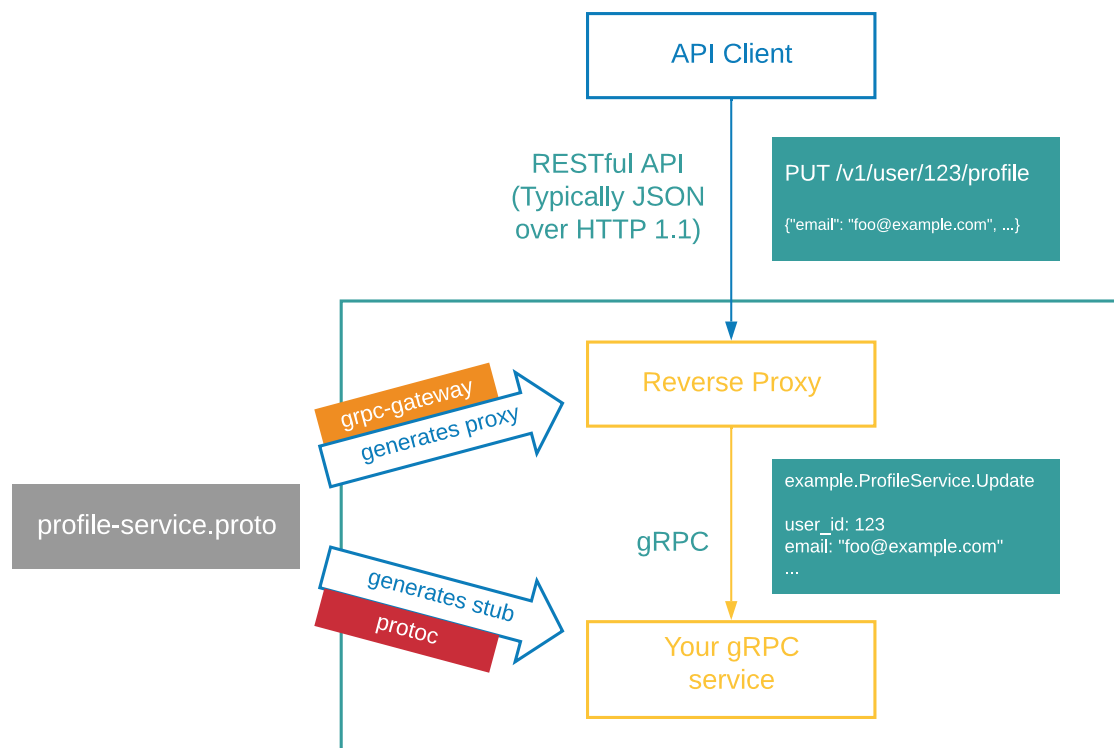


Figure 7.16: gRPC-Gateway architecture diagram [gRPC Gateway Authores, 2023]

While this comes with the RESTfull API advantages and full support of the web development community it does not differ in its essence 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-Factor 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

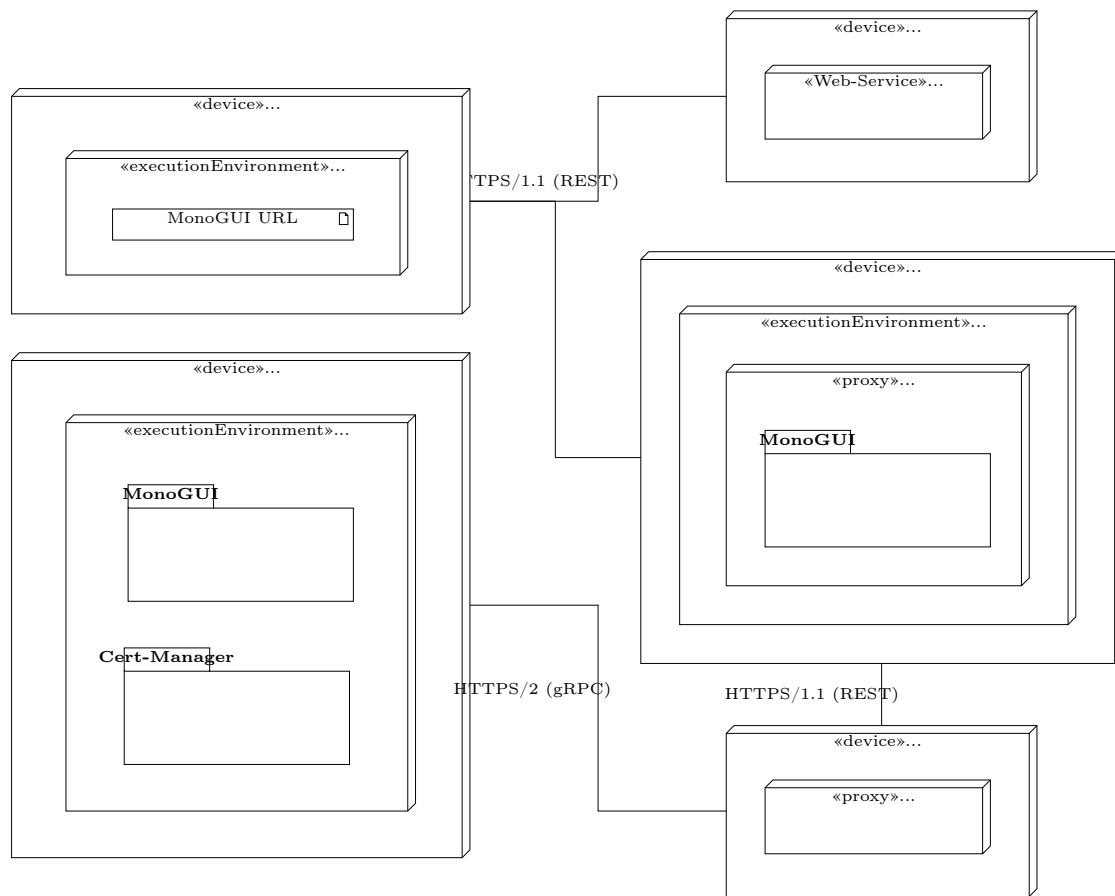


Figure 8.1: Deployment view diagram

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 enviornment 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  
win-  
dows 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 credentials:

- `username: admin@monoskope.dev`
- `password: password`

6. Populate the Event Store with some mock data for a better UX

```
$ make mock-data
```

Shell Instructions 8.6: Populate the Event Store 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 combined with Event-Sourcing allow for very advanced log analisys and bhaviour control .....

In summary, event sourcing offers several advantages over traditional database systems when it comes to auditing. It provides a more detailed and transparent audit trail, allows for the reconstruction of the data's history from the event log, and allows for the implementation of fine-grained access controls. These advantages make event sourcing a valuable tool for organizations that need to ensure the integrity and transparency of their data.

In conclusion, event sourcing offers several advantages in terms of auditing and accountability compared to other architectures. Its detailed and comprehensive audit trail, ability to reconstruct past states, and robust and flexible approach make it a powerful tool for ensuring transparency and accountability in applications.

the studie contudcted by ..... on 25 engineers of different backgrounds and roles by applying applying 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]

# 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>
- [Avery und Reta 2017] AVERY, Phillipa ; RETA, Robert: Scaling event sourcing for netflix downloads. In: *URL: https://www.infoq.com/presentations/netflix-scale-event-sourcing* (2017)
- [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)
- [Erb u. a. 2018] ERB, Benjamin ; MEISSNER, Dominik ; OGGER, Ferdinand ; KARGL, Frank: Log Pruning in Distributed Event-Sourced Systems. In: *Proceedings of the 12th ACM International Conference on Distributed and Event-Based Systems*. New York, NY, USA : Association for Computing Machinery, 2018 (DEBS '18), S. 230–233. – URL <https://doi.org/10.1145/3210284.3219767>. – ISBN 9781450357821

- [EU 2016] EU: *Art. 17 GDPR – Right to erasure ('right to be forgotten') - General Data Protection Regulation (GDPR)*. <https://gdpr-info.eu/art-17-gdpr/>. 05 2016. – URL <https://gdpr-info.eu/art-17-gdpr/>. – (Accessed on 02/21/2023)
- [Evans 2004] EVANS, Eric: *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional, 2004
- [Fowler 2004] FOWLER, Martin: *Audit Log*. <https://martinfowler.com/eaDev/AuditLog.html>. 03 2004. – URL <https://martinfowler.com/eaDev/AuditLog.html>. – (Accessed on 02/20/2023)
- [Fowler 2005] FOWLER, Martin: Event Sourcing. In: *martinfowler.com* (2005). – URL <https://martinfowler.com/eaDev/EventSourcing.html>. – (Accessed on 12/24/2022)
- [Fowler 2017] FOWLER, Martin: *What do you mean by "Event-Driven"?* <https://martinfowler.com/articles/201701-event-driven.html>. 02 2017. – URL <https://martinfowler.com/articles/201701-event-driven.html>. – (Accessed on 02/21/2023)
- [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)
- [Gorodinski 2017] GORODINSKI, Leo: *Scaling Event-Sourcing at Jet*. At Jet, we've been using event-sourcing... | by Leo Gorodinski | Medium. (2017), 10. – URL <https://medium.com/@eulerfx/scaling-event-sourcing-at-jet-9c873cac33b8>. – (Accessed on 02/22/2023)
- [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](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).

**Command Query Responsibility Segregation** .  **TODO**

**Continuous Delivery** .  **TODO**

**Continuous Integration** .  **TODO**

**Custom Resource Definition** .  **TODO**

**Docker** .  **TODO**

**Domain-Driven Design** .  **TODO**

**Emissary-Ingress** .  **TODO**

**Emissary-Ingress** .  **TODO**

**Event Store** ... [more under chapter 4.1.2.](#)  **TODO**

**Event-Sourcing** ... [more under chapter 4.](#)  **TODO**

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

**GO** .  

**gRPC-Web** .  

**Helm** .  

**HPACK** .  

**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** .  

**Kind** .  

**Kubectl** .  

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


[Kubernetes, 2014] .

**Make** .  

**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** .  

**Public Key Infrastructure** .  



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