HAW HAMBURG

BACHELOR THESIS
Hani Alshikh

Evaluation and Use of Event-Sourcing for Audit Logging

Faculty of Engineering and Computer Science
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE
WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Hani Alshikh

Evaluation and Use of Event-Sourcing for Audit Logging

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

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

Submitted on: 28. February 2023

Hani Alshikh

Title of Thesis

Evaluation and Use of Event-Sourcing for Audit Logging

Keywords

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

Abstract

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

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

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

Contents

List of Figures

List of Tables

Shell Instructions

Abbreviations

AI Artificial Intelligence.

AOP Aspect-Oriented Programming.

API Application Programming Interface.

CA Certificate Authority.

CD Continuous Delivery.

CI Continuous Integration.

CQRS Command Query Responsibility Segregation.

CRD Custom Resource Definition.

CRUD Create, Read, Update and Delete.

CSV Comma Separated Values.

DDD Domain-Driven Design.

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.

OIDC Open-ID Connect.

PKI Public Key Infrastructure.

POC Proof Of Concept.

REST REpresentational State Transfer.

TS Typescript.

URI Universal Resource Identifier.

URL Universal Resource Locater.

UX User Experince.

XML Extensible Markup Language.

1 Acknowledgments

Special thanks to Prof. Dr. Stefan Sarstedt for supervising this work, the guidance and support throughout the process, Prof. Dr. Olaf Zukunft for reviewing and evaluating this work and Christian Hüning for the help finding, choosing and polishing this work and being there and supportive.

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 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.

Beside evaluating Event-Sourcing in regards to auditing and audit logging this work make use of a real world use-case of audit logging needs by implementing the Audit Component to serialize the event log into an audit log and offer an audit Application Programming Interface (API) for a Kubernetes (k8s) multi-cloud multi-cluster authentication and authorization system used by a FinTech organization as well as an Audit Browser to ease the auditors interactions with the system and help answers various compliance questions.

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, 2014a]

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, 2014a]

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.

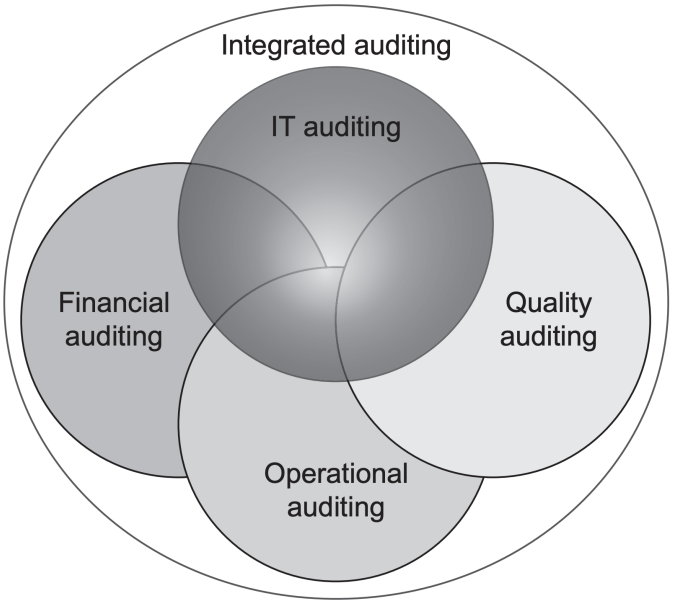

Figure 3.1: IT auditing commonality with other types of audit [Gantz, 2014a]

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, 2014a].

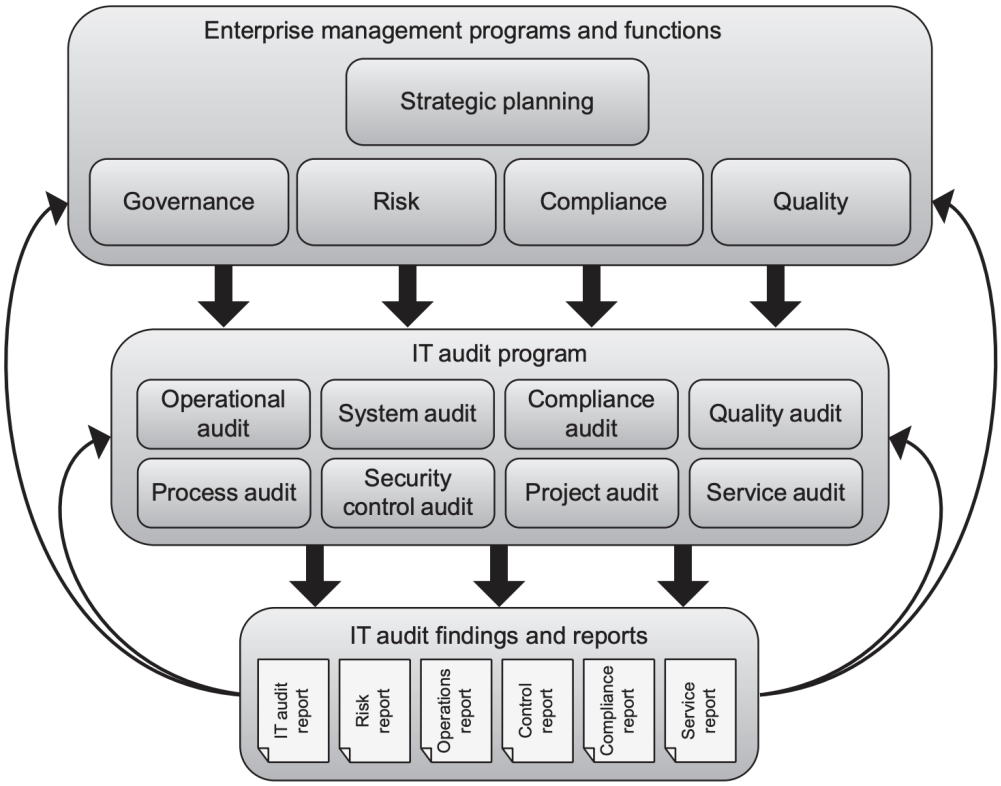

Figure 3.2: IT audit activities and scopes [Gantz, 2014a]

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 controls 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: Internal controls categorized by type and purpose [Gantz, 2014a]

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 dictate 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 this 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, 2004a]. 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 is not enough. Suspicious activates might originate from read attempts. Logging such activities increases the complexity of the Audit Log, violates the Audit Log definition 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 timestamp, or data elements recorded with the event [van der Aalst u. a., 2010].

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

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 system's 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 musst 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 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 this 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 and 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, business objects potentially need to fold events corresponding to multiple schema versions. There is a real risk that the objects 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 object by setting a deleted flag. The object will typically emit a deleted event, which notifies any interested consumers. Any code that accesses that object 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.

By implementing CQRS a separate query handler service is maintained. The query side keeps its data model synchronized with the command-side data model by subscribing to the events published by the command side and thus have direct access to the current transient state to efficiently handle such queries.

The following digram showcase how CQRS can be be implemented in a microservices system.

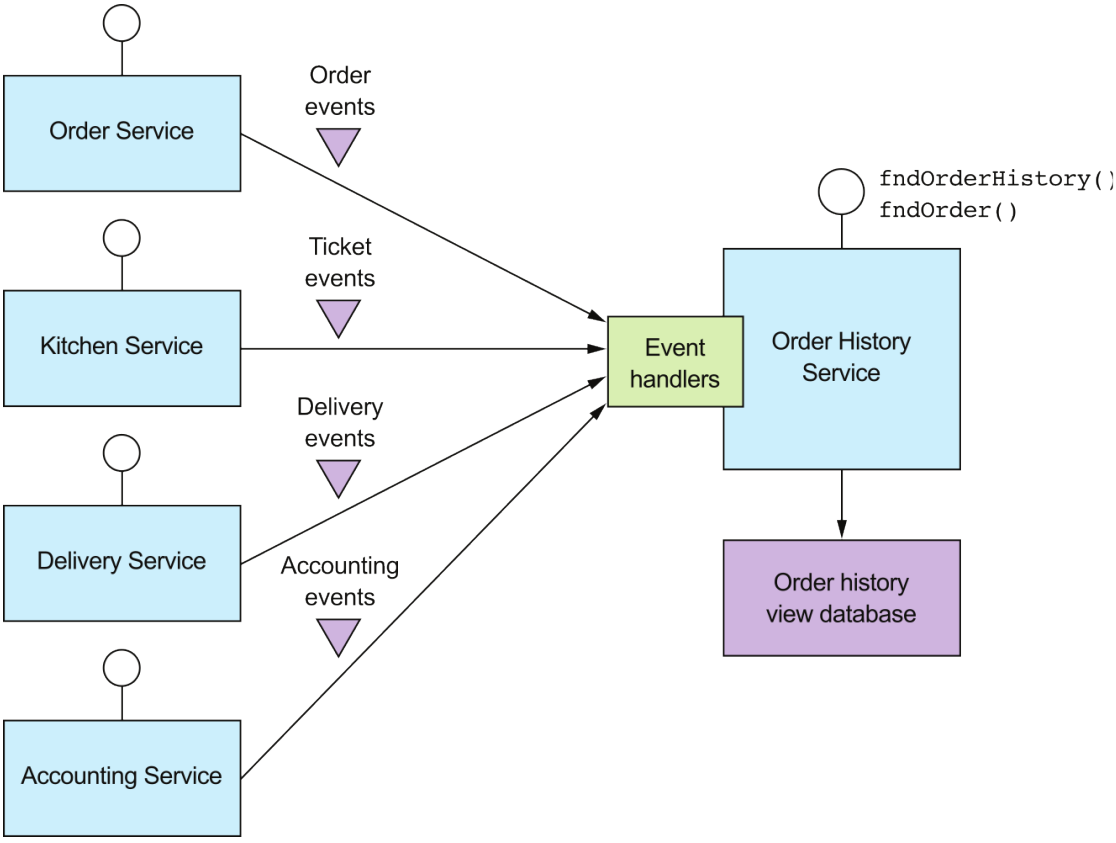

Figure 4.1: CQRS implementation in a microservices system [Richards, 2015]

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 utilise when implementing Auditing 2.0 as discussed in section 3.4

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 the 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

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.

The software architecture is defined based on the software requirements. Among the different types of requirements, the quality requirements are the most important for the specification of an architecture since it exerts considerable influence over it structure [Bass u. a., 2003].

Having a system that is designed and built from the ground up to offer the highest inclusion of auditing standards and capabilities starts by choosing the right architectural components. Auditing functionalities is often added as an afterthought, resulting in an inherent risk of incompleteness.

Avoiding such risk and insuring auditability by design can be done in many different ways. Ultimately, the choice of architecture patterns for auditing will depend on the specific needs of the system and the requirements of the audit process.

As discussed in section 3.3 audit logging is one of the simplest way to have an auditable system, that satisfies regulators specifications.

5.1 Implementing audit logging

The glory of the Audit Log pattern is its simplicity. Comparing Audit Log to other patterns such as Temporal Property [Fowler, 2004d] and Temporal Object [Fowler, 2004c] shows, 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 [Fowler, 2004b] everywhere [Fowler, 2004a].

As described in section 3.3 simply writing log entries into a file is not enough. At least when ensuring business provenance [3.4.1] is a requirement. Provenance data will make it possible to replay history reliably and accurately and to predict problems, thereby improving business processes.

Using Audit Log in some parts of the system and other patterns elsewhere is common and sometimes necessary. For example Audit Log can be used in combination with Event-Sourcing for state changes and a different pattern for read operations.

[Richardson, 2018] describes three main ways to implement audit logging:

1. Add audit logging code to the business logic.
2. Use Aspect-Oriented Programming (AOP).
3. Use Event-Sourcing.

5.1.1 Audit Logging Code in Business Logic

The first and most straightforward option is to sprinkle audit logging code throughout the 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 audi logging code and business logic, which reduces maintainability. The other drawback is that it is potentially error prone, because it relies on the developer writing audit logging code and insuring compliance.

Logging in general is mostly associated with debugging and has no direct relation to the system state. Logging style and verbosity might negatively effect the integrity and completeness of the Audit Log.

5.1.2 Aspect-Oriented Programming

The second option is to use AOP. Frameworks like spring offer AOP support, which 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, which insures a complete Audit Log of both reading and writing operations.

The main drawback of using AOP is that it only has access to the method name and its arguments, thus it might be challenging to determine the business object being acted upon and generate a business-oriented audit log entry.

5.1.3 Event Sourcing

The third option is to implement the business logic with auditability as a first class property by utilising Event-Sourcing and its complementary patterns as discussed in chapter 4.

Event-Sourcing offer an Audit Log by design for all state changing operations. The main limitation when using Event-Sourcing is the missing records for read operations.

If logging read operations is a requirement, using one of the other options is a must. Patterns like CQRS help encapsulate the different implementations and unify the auditing API

5.2 What to Consider

5.2.1 Traditional Persistence

When it comes to auditing a limitation of the traditional persistence methods is that, they only store the current state of a business object. Once the object has been updated, its previous state is lost.

Such historical records are the base for services like Asana or Jira. Keeping track of such changes, while insuring association to the correct object and actor is a challenge, that comes with its added complexity.

Simply logging different kind of operations is not enough. If tracking objects history and insuring association between each log entry, the object involved and the actor initiating the operation is a requirement choosing the right pattern makes all the difference.

The challenge of implementing auditing as an added feature like the case with the first two options discussed in section 5.1 is that, besides being a time-consuming chore, the audit logging code and the business logic can diverge, resulting in all different kind of bugs.

5.2.2 Auditing 2.0

Performing and supporting IT audits and managing IT audit programs are time-, effort-, and personnel-intensive activities [Gantz, 2014b]. In an age of cost-consciousness and competition for resources, it is reasonable to keep Auditing 2.0 in mind when implementing Audit Log. Having an Audit Log, that can be utilised by the different process mining techniques as described in section 3.4.2 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.

Auditors can utilise process mining techniques to address multiple use-cases/processes. By implementing patterns like Event-Sourcing organizations can 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.

6 Audit Component

As discussed in chapter 5 choosing the correct architecture for auditing is an important choice to make. Handling auditing and audit logging as first class citizens comes with its owen constraints and requirements.

In an industry where strict auditing requirements apply using patterns like Event-Sourcing not only ensures a compliant audit log but also lays the ground work for broader auditing technologies like Auditing 2.0

Combining DDD, Event-Sourcing, and CQRS produces an architecture that is praised by many, when it comes to building audit first cloud systems.

To showcase the capabilities of such architecture and evaluate the development overhead compared to the gained benefits a real word example was chosen as a base to implement the Audit Component from scratch.

The Audit Component utilises the produced event log by the Event-Sourcing implementation to offer an auditing API, that can be utilised by all kind of clients like auditing programs as described in section 5.2.2 or in this case by the Audit Browser implemented in chapter 7

6.1 Monoskope

Monoskope (short m8 spelled "mate") implements the management and operation of authenticating and authorizing entities in a Kubernetes multi-cloud multi-cluster environments. It fulfills the needs of operators of the clusters as well as the needs of developers using the cloud infrastructure provided by the operators [Monoskope-Authors, 2021].

6.1.1 Architecture

M8 was designed with auditing in mind. It uses the event-sourced architecture [4.3] with a purpose built implementation of the Event Store as well as the CQRS pattern.

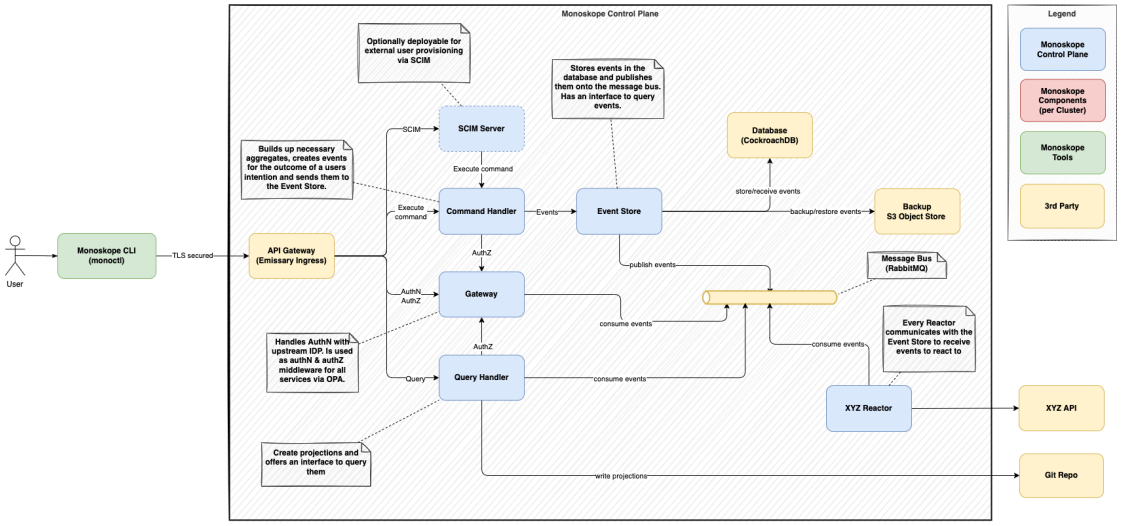

Figure 6.1: Monoskope architecture [Monoskope-Authors, 2021]

Since m8 is meant to be used as the single source of truth authority for authenticating and authorizing users into different clusters on different cloud providers ensuring auditability and complying with different regulators requirements was not an option.

While the base system was implemented and the main features were complete m8 still lacked a proper implementation to utilise the gained event log and offer an auditing API, that answers Auditors questions.

M8 has no classic CRUD database with tables for storing state, however, the transient state built based on that has. The following is the data model for the projected transient state:

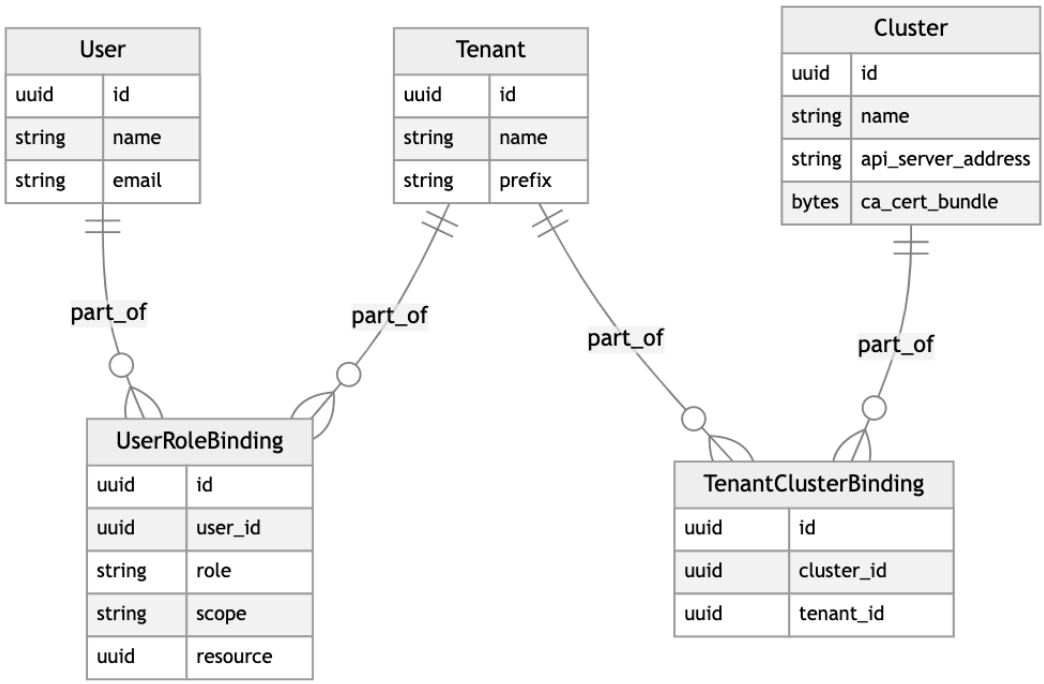

Figure 6.2: Monoskope data module

Generally users represent developers that are bound/a part of one or many teams/tenants that are bound/have access to one or many clusters.

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?
- etc. . .

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

It should be possible to utilise Event-Sourcing's temporal quires [4.2] to get a users overview at any date and time

6.2.1 Use-Cases

From the requirements overview the following use-cases were derived:

ID	Use-Case	Description
UC01	Audit-Log for date-range	As an Auditor, I want to get all actions taken within a specific date-range
UC02	Audit-Log about a user	As an Auditor, I want to get all actions taken on a user
UC03	Audit-Log of user-actions	As an Auditor, I want to get all actions taken by a user
UC04	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

Table 6.1: Audit Component derived use-cases

6.2.2 Architectural Constraints

Technical Constraints

ID	Constraint	Description
TC01	Human-Readable representation	Event objects musst contain a human-readable representation
TC02	Programming language	M8 is written in GO. No reason to use other languages
TC03	Middleware	M8 uses gRPC. No reason to use or support any other framework

Table 6.2: Audit Component technical constraints

Organisational Constraints

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

Table 6.3: Audit Component organisational constraints

6.3 System Design

6.3.1 Scope and Context

Business Context

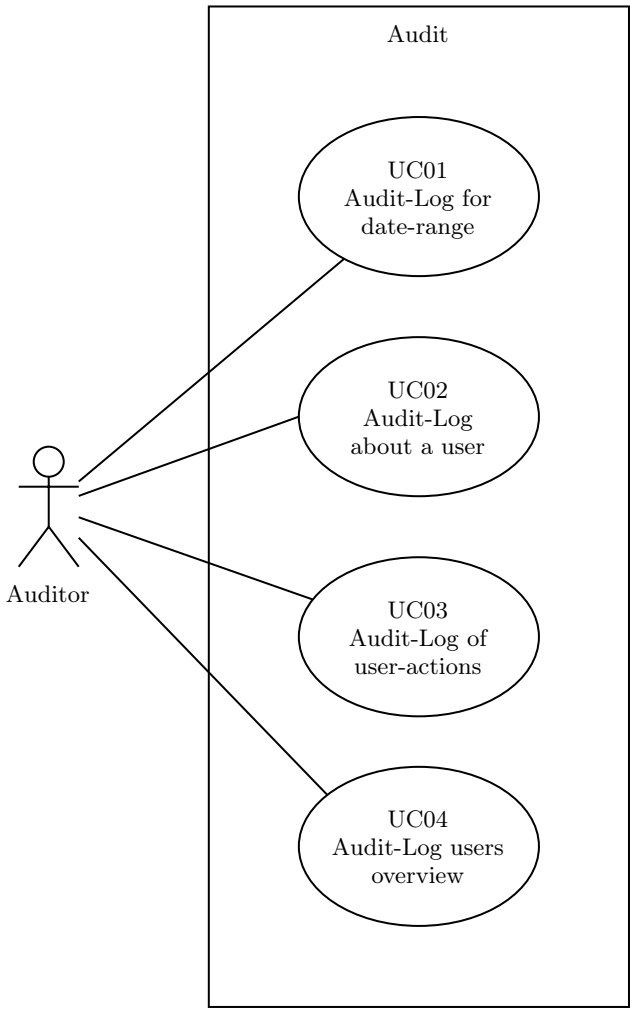

Figure 6.3: Audit Component business context diagram

Technical Context

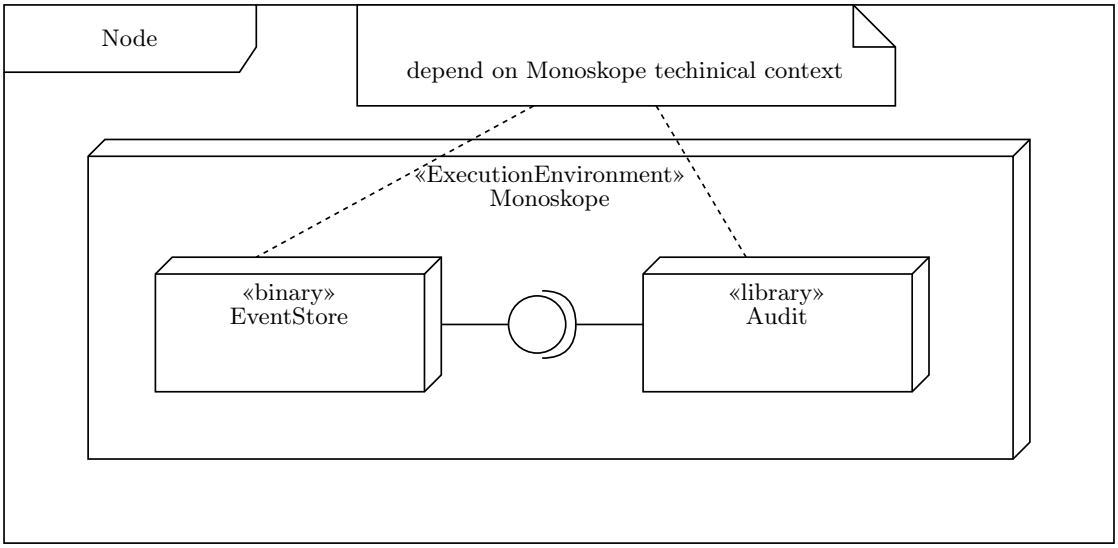

Figure 6.4: Audit Component technical context diagram

6.3.2 Solution Strategy

Funktion	UCID	Semantics	Pre-condition	Post-condition
newEvent-Formatter-Registry()	All	Creates a new EventFormatter-Registry	QueryHandler is initializing	EventFormatters can be registered
registerEvent-Formatter(-eventFormatter,-eventType)	All	Register Event-Formatter for an event type	EventFormatter registry is initiated	EventFormatter can be used to format events of eventType
newAuditLog-Server(event-StoreClient, eventFormatter-Registry)	All	Creates server instance of the Audit Component to handle client requests	Event StoreClient and EventFormatterRegistry are initialized	Audit-log server is ready to handle client requests

getByDate-Range(date-RangeRequest)	UC01	Streams human-readable events within a date-range	AuditLogServer is running	Human-readable events were streamed to the client
newHuman-ReadableEvent(-event)	All	creates a human-readable event of an event	EventFormatter for the corresponding event type was registered	A new event with human-readable details is created
getByUser(get-ByUserRequest)	UC02	Streams formatted events caused by others actions on the given user	AuditLogServer is running	Human-readable events were streamed to the client
getUsers-Actions(getUser-ActionsRequest)	UC03	Streams formatted events caused by the given user actions	AuditLogServer is running	Human-readable events were streamed to the client
getUsers-Overview(get-UsersOverview-Request)	UC04	Streams formatted events at the specified timestamp of users, tenants/clusters they belong to, and their roles	AuditLogServer is running	Human-readable events were streamed to the client

Table 6.4: Audit Component solution strategy

6.3.3 Building Block View

Overall System White Box

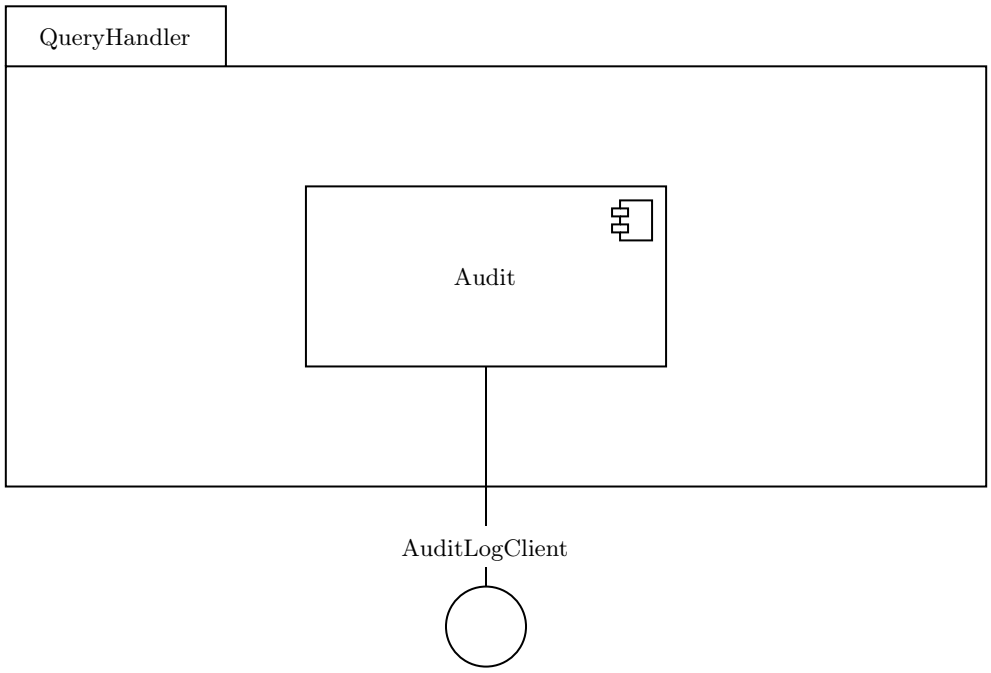

Figure 6.5: Audit Component overall system component diagram

Level 1

Contained Building Blocks

Audit Component Black Box

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

Table 6.5: Audit Component contained building blocks component black box

Interface	Description
AuditLogClient	handles communication with the audit log server

Table 6.6: Audit Component contained building blocks interface black box

Level 2

Audit Component White Box

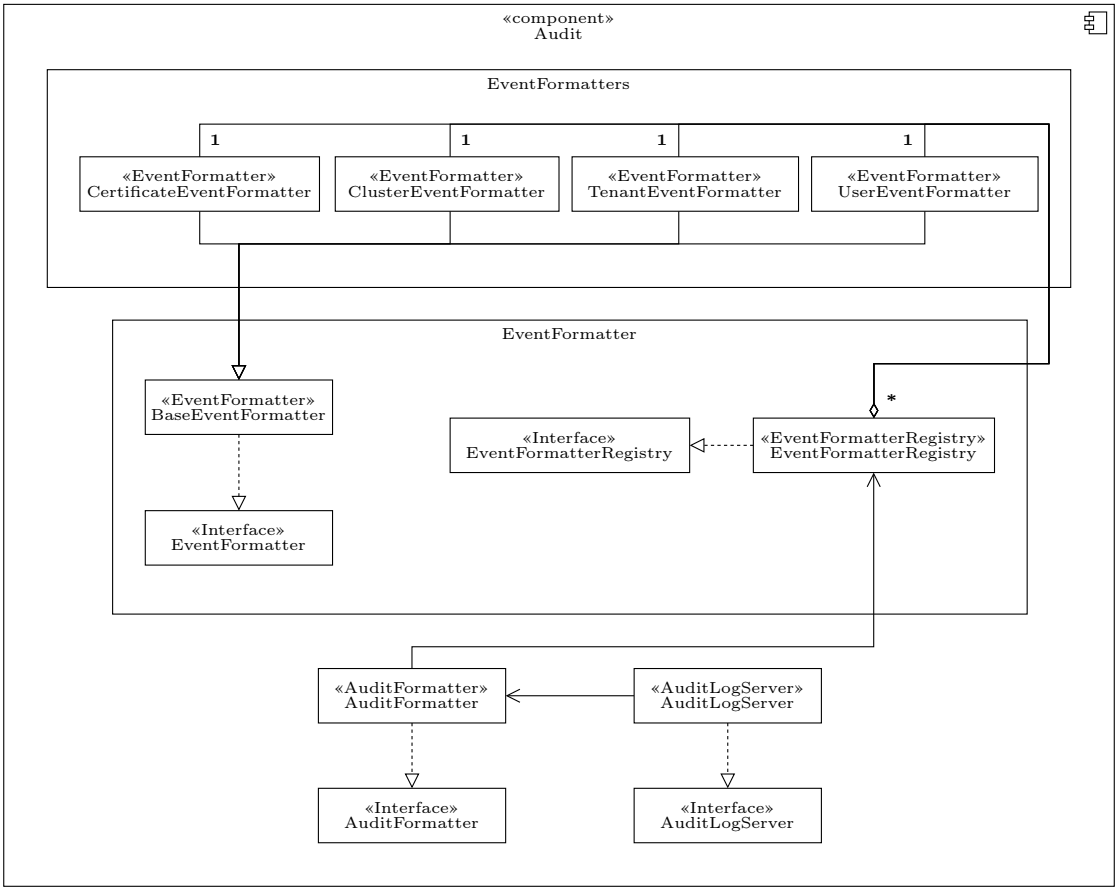

Figure 6.6: Audit Component class diagram

Object	Description
CertificateEventFormatter	Formates certificate events in human-readable format
ClusterEventFormatter	Formates cluster events in human-readable format
TenantEventFormatter	Formates tenant events in human-readable format
UserEventFormatter	Formates user events in human-readable format
BaseEventFormatter	Base implementation of the EventFormatter interface to generalize common methods
EventFormatterRegistry	Register EventFormatter for event type
AuditFormatter	Creates a human-readable event of a given event
AuditLogServer	Handles communication with the AuditLogClient

Table 6.7: Audit Component class diagram

6.3.4 Runtime View

UC01-04: Audit-Log *

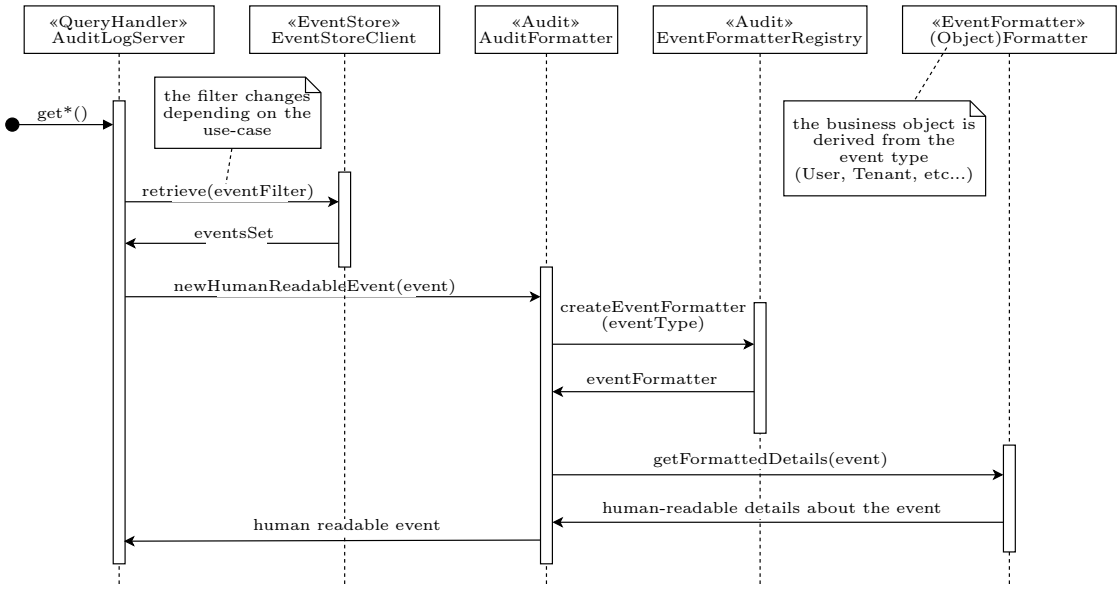

Figure 6.7: Audit Component UC01-04 sequence diagram

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 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 compliant 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 to external controls it is customary to work with pre-recorded parteitonend samples, which might give a good overview but never the full one.

Auditing 2.0 discussed in section 3.4 describe practices to mitigate the mentioned problems and provides the basis for an auditing frameworks in attempt of continues auditing approach. Using Event-Sourcing as a reliable audit logging mechanism and the Audit Component to implement the the underlying API 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.

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.

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 different 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 in a spread-sheet compatible format like Comma Separated Values (CSV).

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

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	M8 is written in GO

Table 7.2: Audit Browser technical constraints

Organisational Constraints

ID	Constraint	Description
OC01	Deadline	Implementation musst be finalised before 24.02.2023
OC02	The Twelve-Factor App	Implementation musst adhere to the The Twelve-Factor App methodology
OC03	Userbase	Userbase is mainly desktop users

Table 7.3: Audit Browser organisational constraints

7.2 System Design

7.2.1 Scope and Context

Business Context

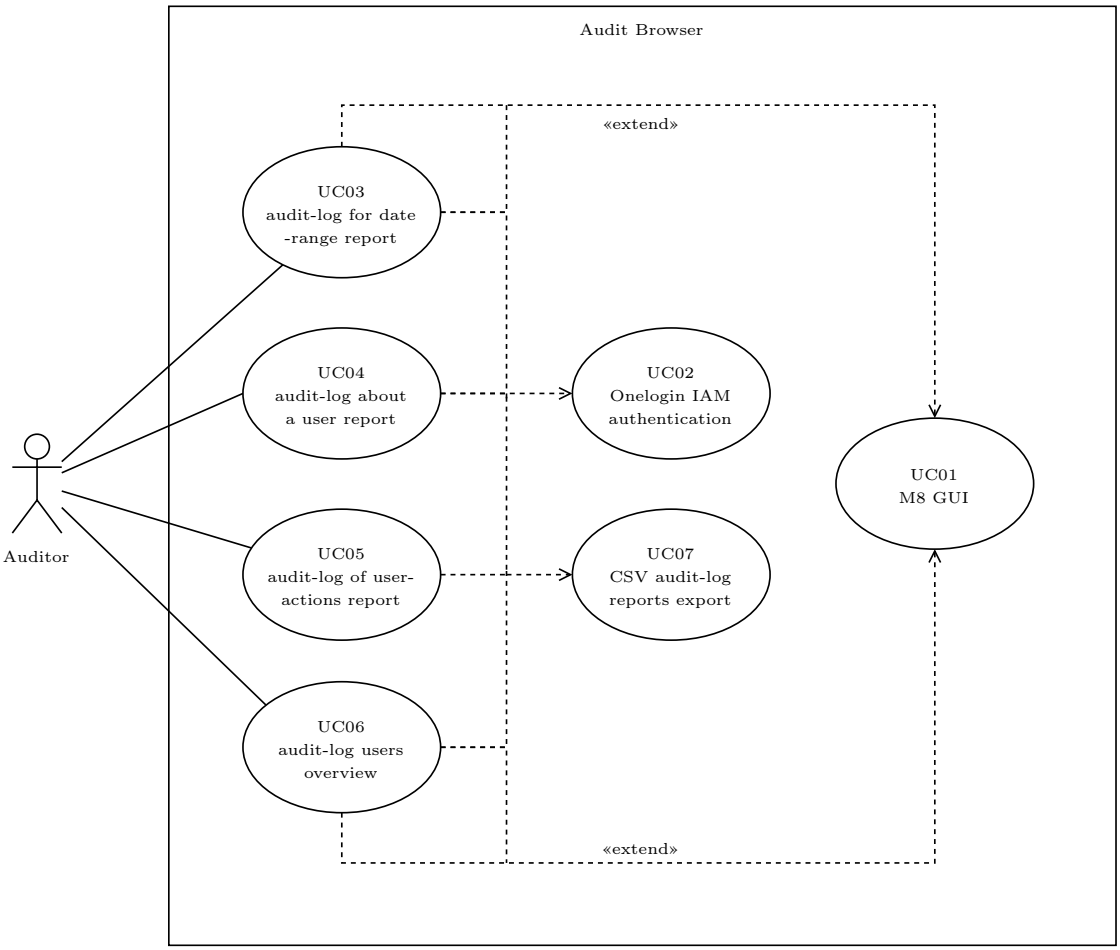

Figure 7.1: Audit Browser business context diagram

Technical Context

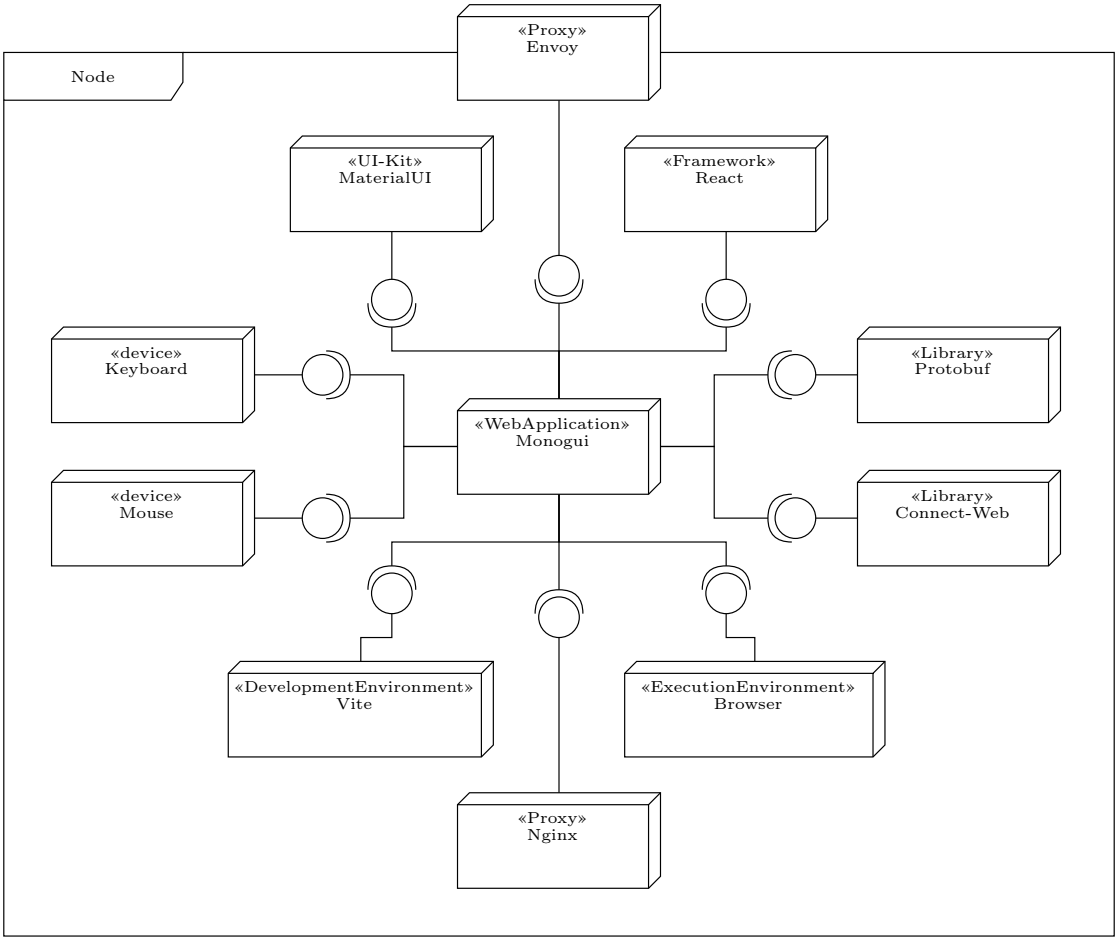

Figure 7.2: Audit Browser technical context diagram

7.2.2 Solution Strategy

Function	UCID	Semantics	Precondition	Postcondition
render(-components)	UC01	render components in the browser	user navigated to MonoGUI based URI	requested components are rendered or user is redirected to authenticate himself
signIn(-m8APIUrl)	UC02	Kick in m8 OIDC authentication flow	user clicked the sign in button	user is redirected to IDP authentication URL
requestAuth(-authCode)	UC02	request authenticaton token from m8	user signed in using his IDP account	user can access protected routes
getAuditLog(-from, to, kargs)	UC03-06	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
exportCSV(table)	UC07	export a table as CSV file	user requested audit log	CSV formatted file of the table is downloaded

Table 7.4: Audit Browser solution strategy

7.2.3 Building Block View

Overall System White Box

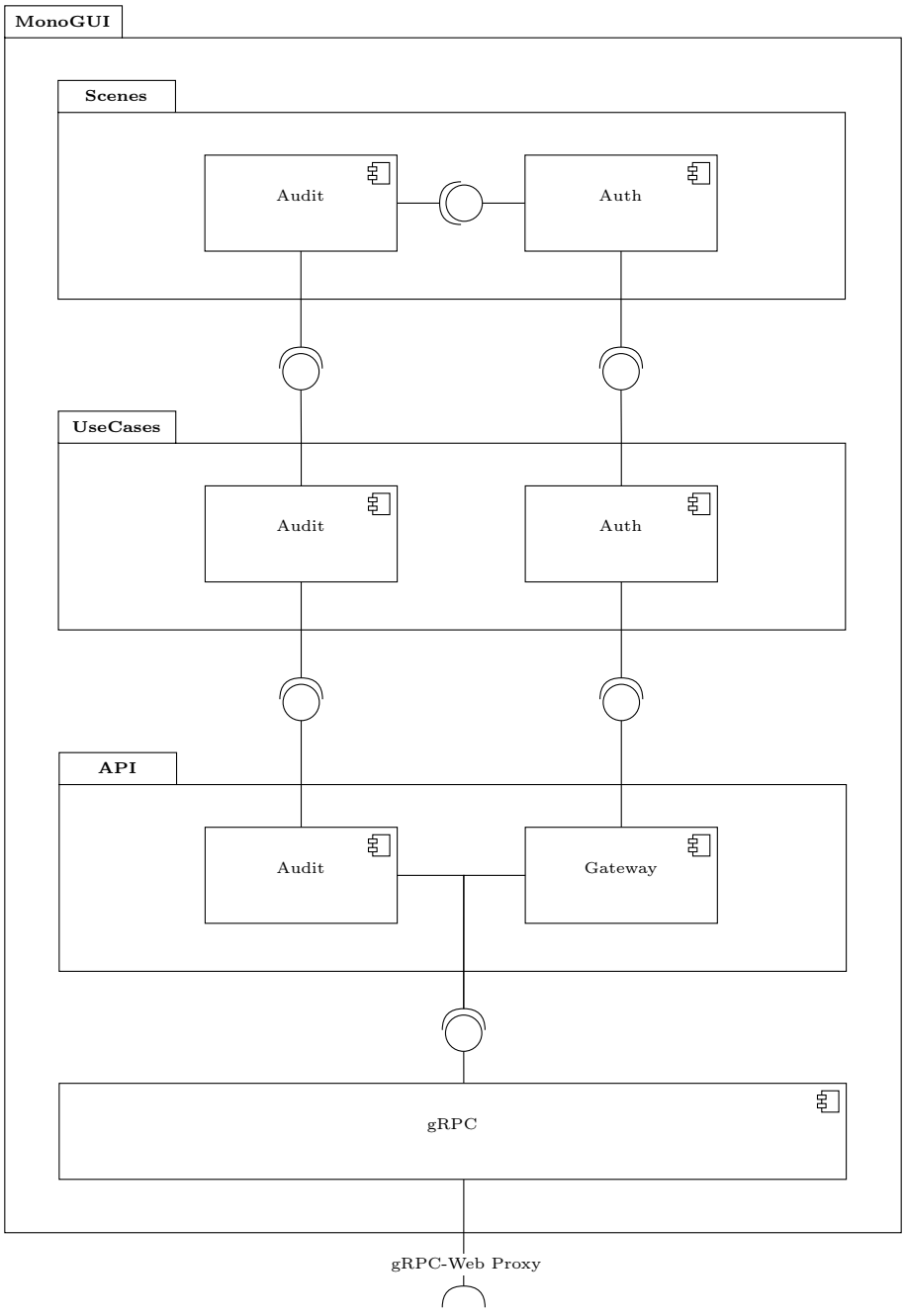

Figure 7.3: Audit Browser overall system component diagram

Level 1

Contained Building Blocks

MonoGUI

Component	Description
gRPC	handles comunication with gRPC-Web Proxy

Table 7.5: Audit Browser contained building blocks MonoGUI 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 contained building blocks scenes 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 and the upstream IDP

Table 7.7: Audit Browser contained building blocks use-cases 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 contained building blocks API black box

Level 2

MonoGUI White Box

gRPC

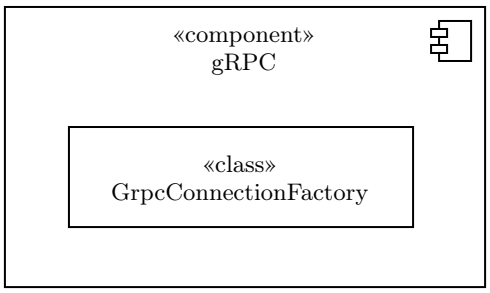

Figure 7.4: Audit Browser MonoGUI gRPC class diagram

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

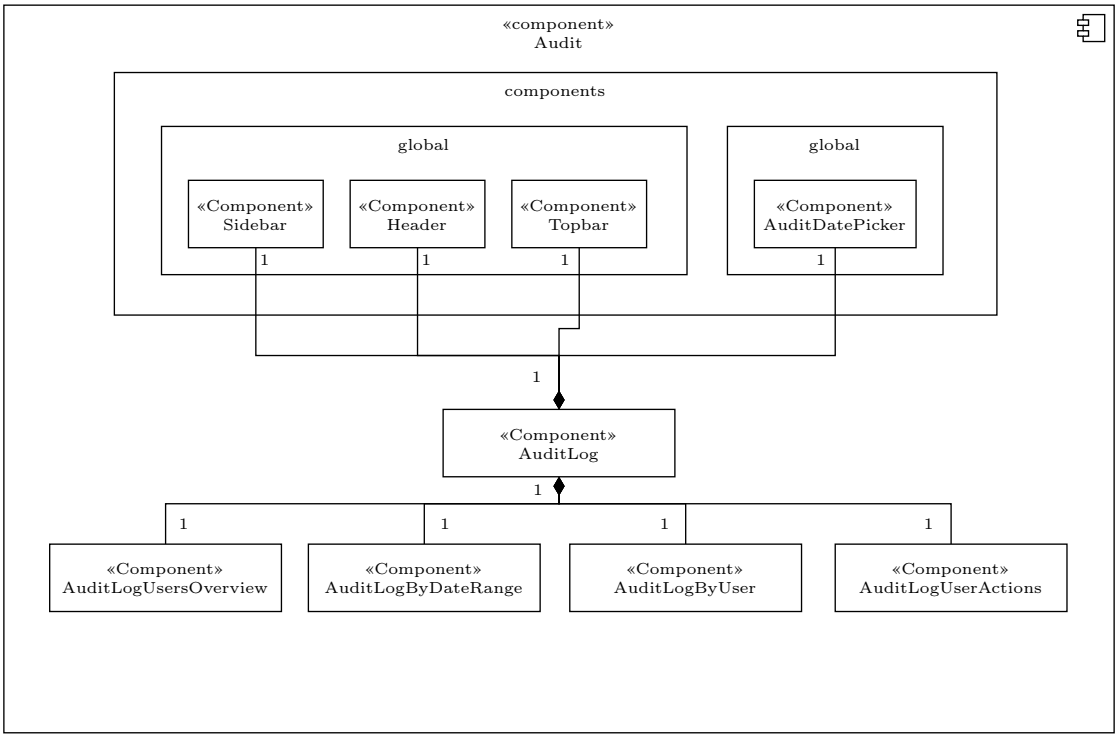

Figure 7.5: Audit Browser scenes audit class diagram

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
AuditLog	composed component for all audit log usecases presentation and interaction
AuditLogByDateRange	composition component for get by date-range use-case presentation and interaction
AuditLogByUser	composition component for get by user use-case presentation and interaction
AuditLogUserActions	composition component for get user actions use-case presentation and interaction
AuditLogUsersOverview	composition component for get users overview use-case presentation and interaction

Table 7.10: Audit Browser scenes audit class diagram

Auth

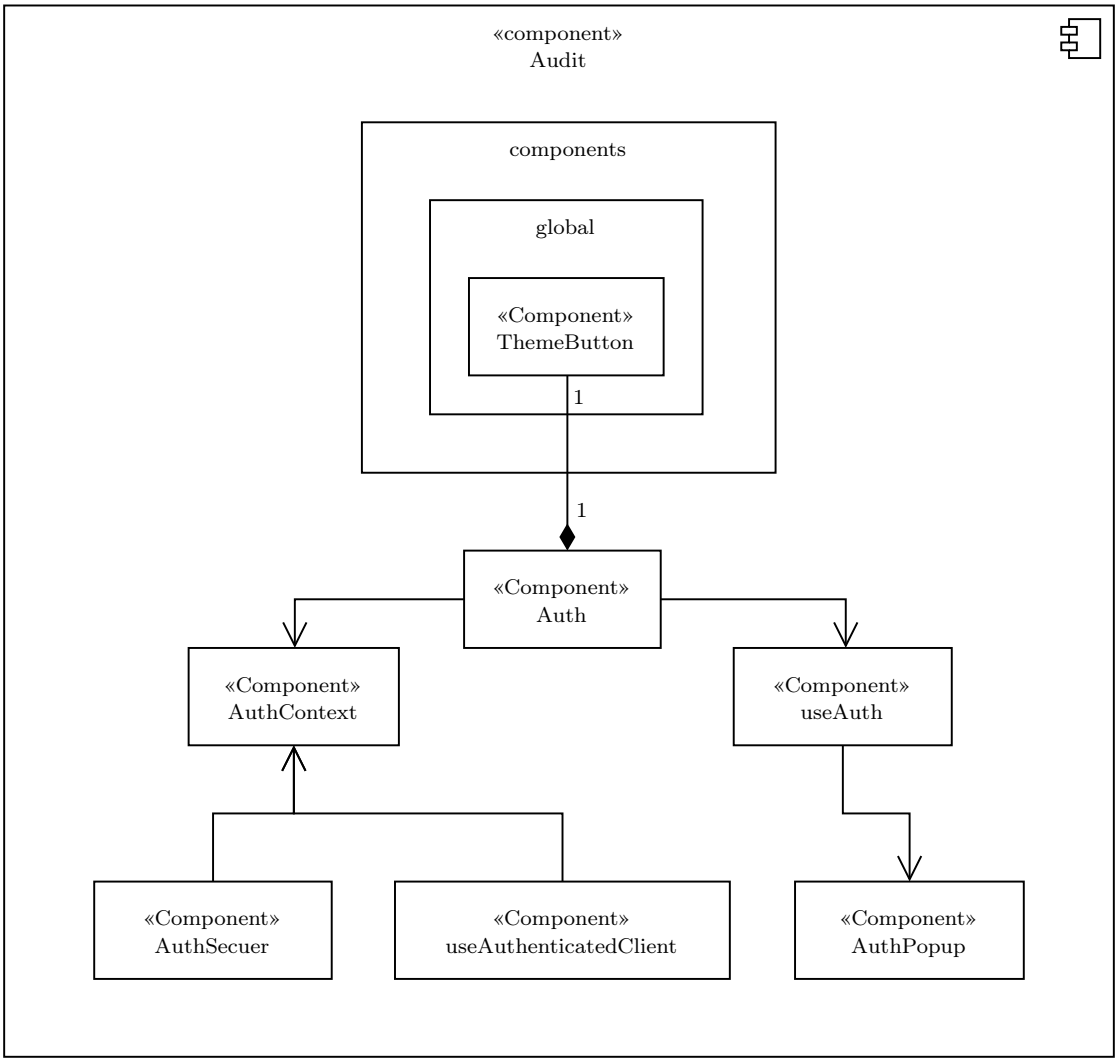

Figure 7.6: Audit Browser scenes auth class diagram

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

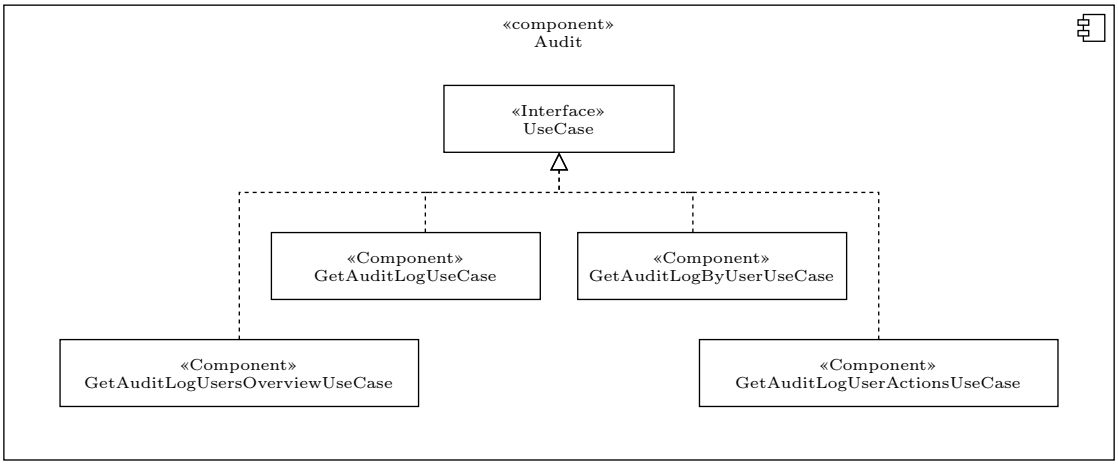

Figure 7.7: Audit Browser usecases audit class diagram

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
GetAuditLogByUserUseCase	Request audit log events about a user from the API and prepare them for presentation components
GetAuditLogUserActionsUseCase	Request audit log events of a user's actions from the API and prepare them for presentation components
GetAuditLogUsersOverviewUseCase	Request audit log overview events of all users from the API and prepare them for presentation components

Table 7.12: Audit Browser usecases audit class diagram

Auth

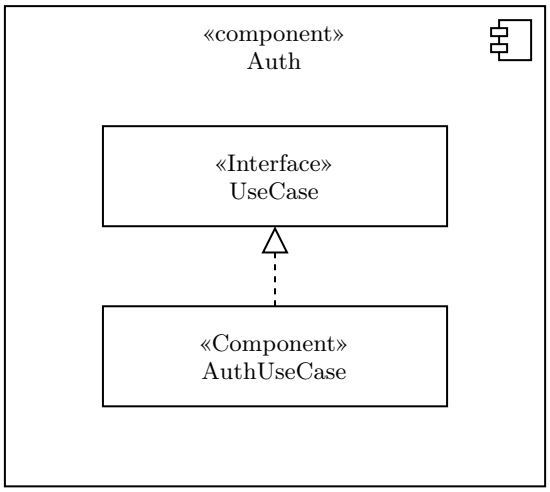

Figure 7.8: Audit Browser usecases auth class diagram

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

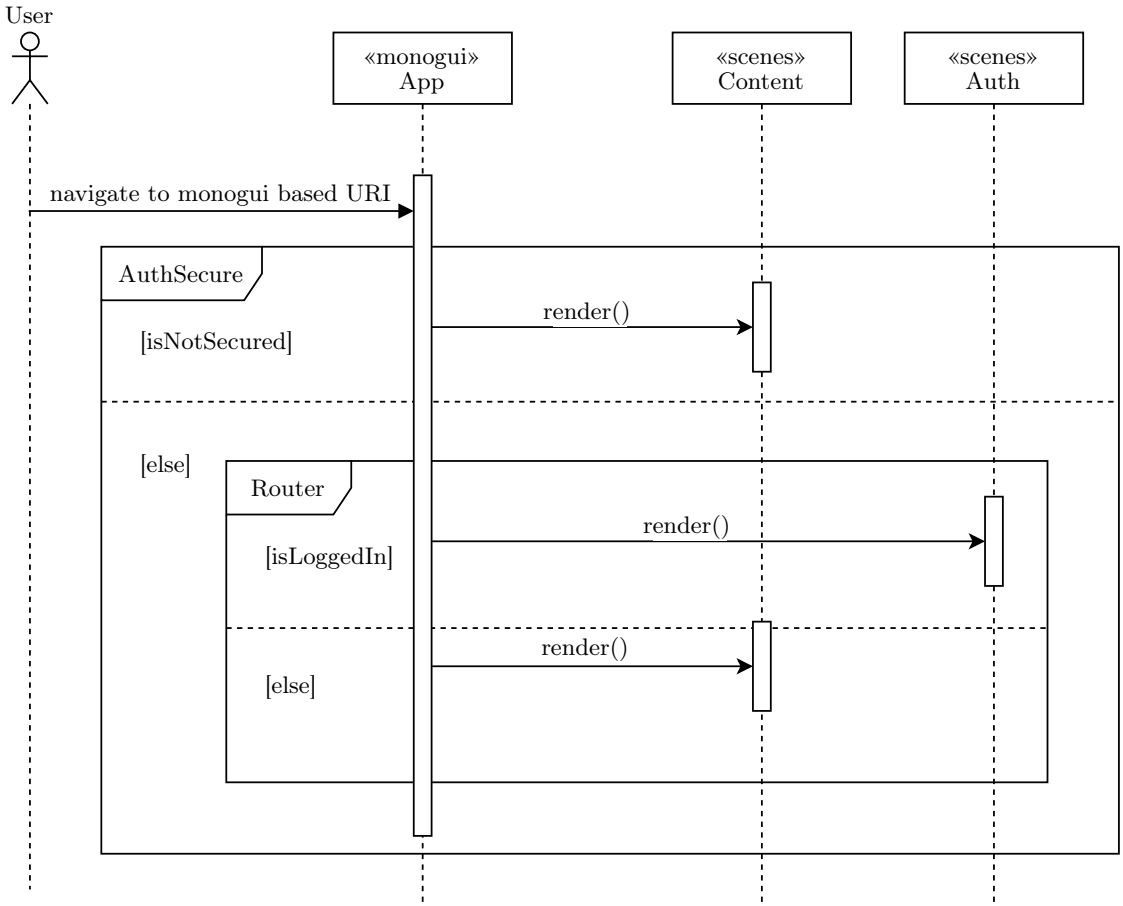

Figure 7.9: Audit Browser UC01 sequence diagram

UC02: OIDC authentication

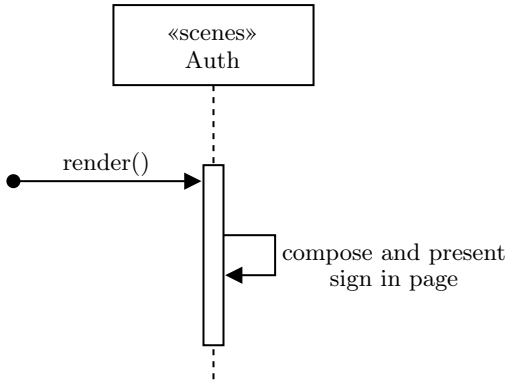

Figure 7.10: Audit Browser UC02 sequence diagram

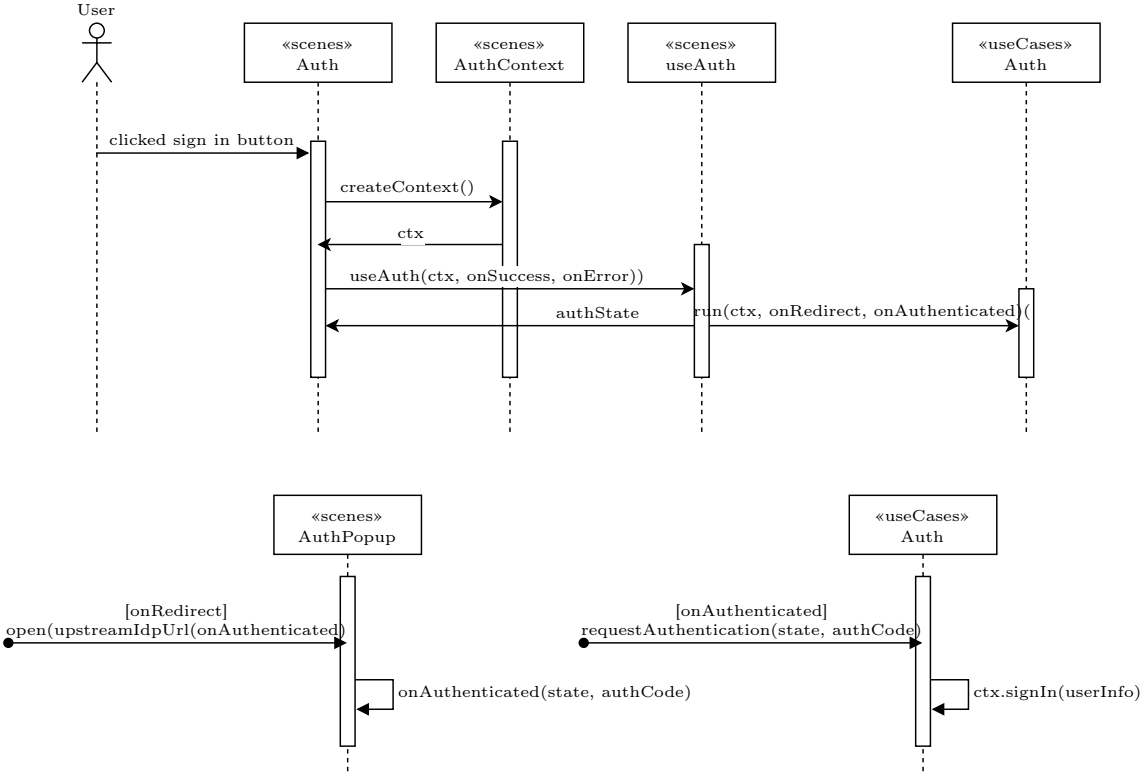

Figure 7.11: Audit Browser UC02 sign in sequence diagram

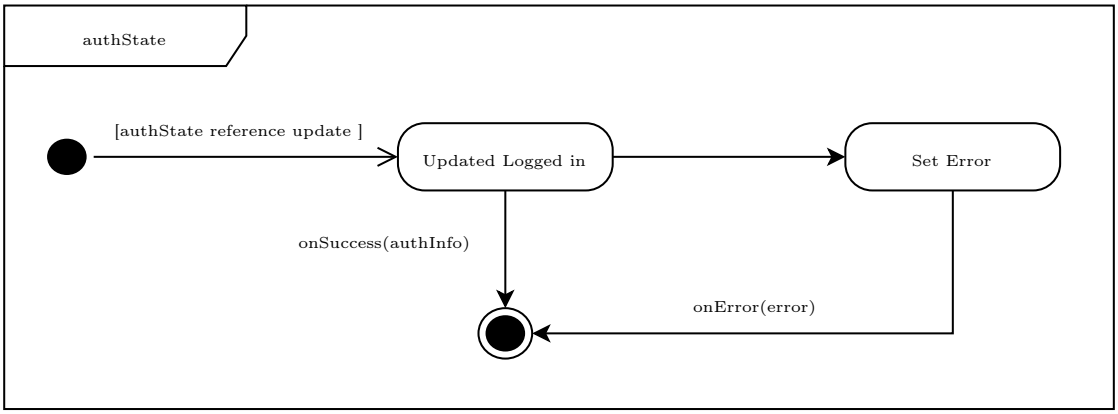

Figure 7.12: Audit Browser UC02 auth state machine

UC03-06: Audit Log Use-Cases

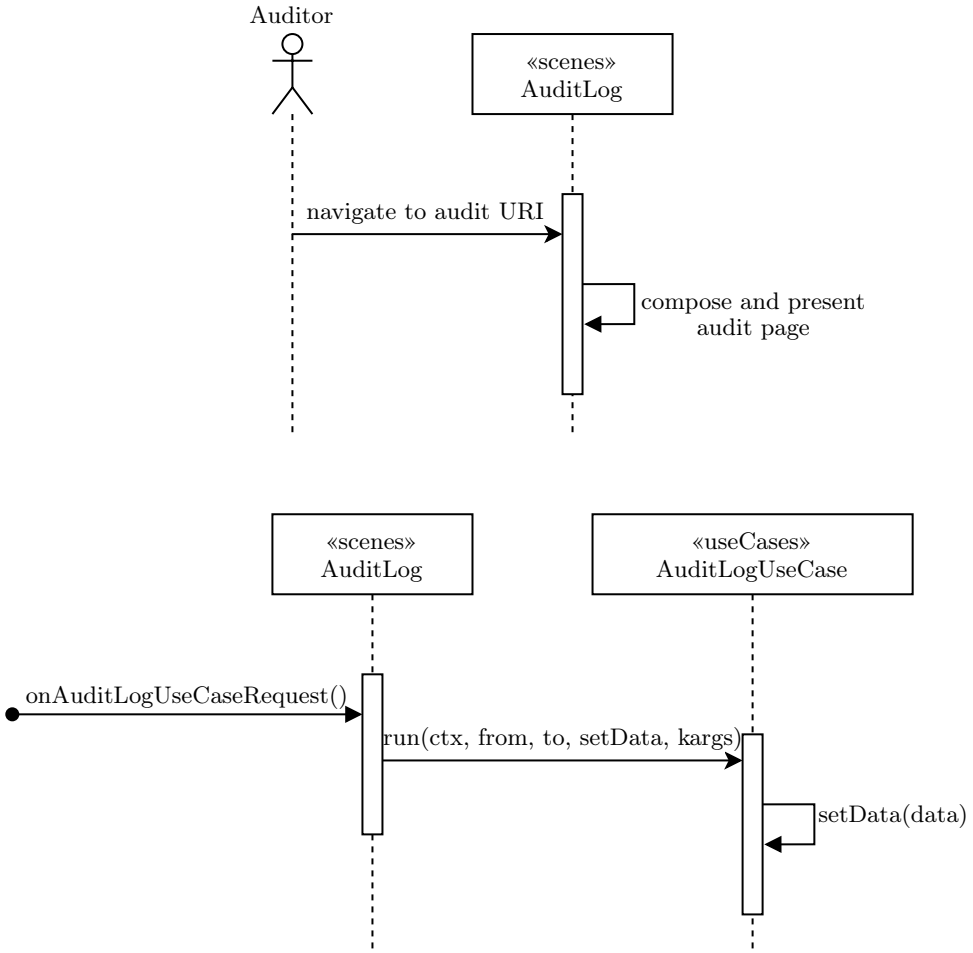

Figure 7.13: Audit Browser UC03-06 sequence diagram

7.3 Design Decisions

7.3.1 DD01: gRPC Client-Server Communication

HyperText Transfer Protocol (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.

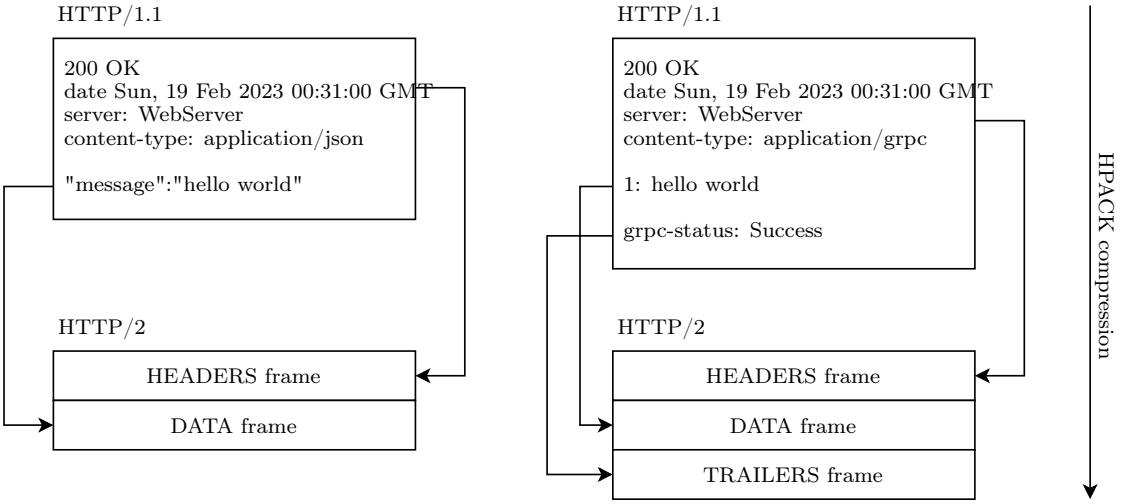

Figure 7.14: HTTP/1.1 compared to HTTP/2

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

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

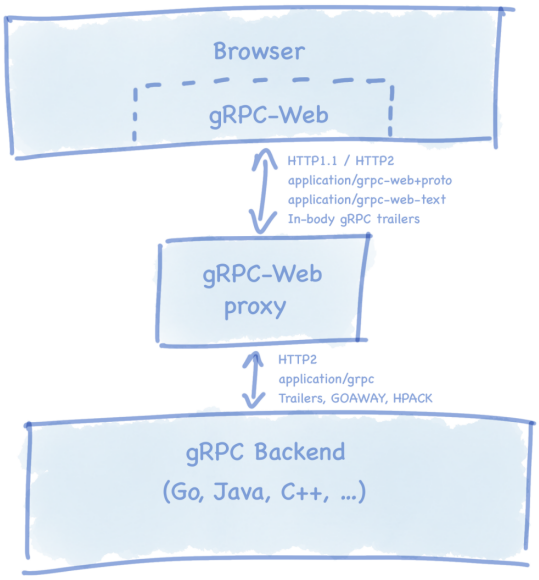

Figure 7.15: gRPC-Web proxy to allow browser gRPC support [Brandhorst, 2019]

Why stick to gRPC?

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

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

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

Other Options

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

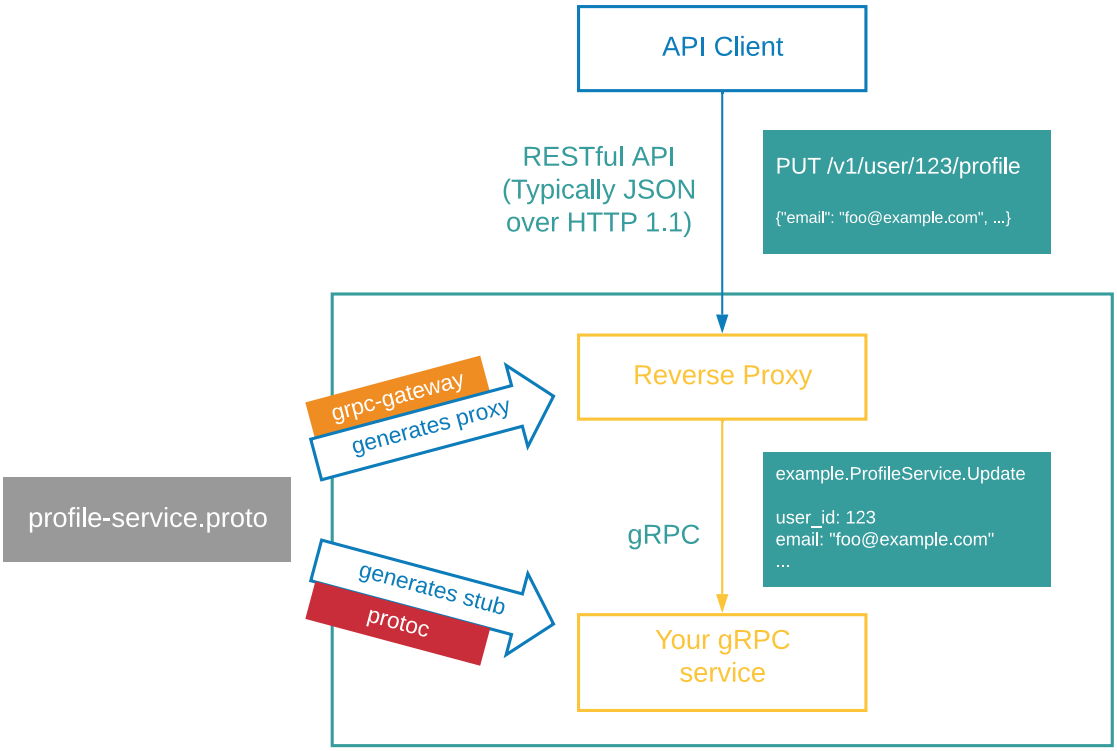

Figure 7.16: gRPC-Gateway architecture diagram [gRPC Gateway Authors, 2023]

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

Conclusion

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

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

7.4 Technical Decisions

7.4.1 TD01: Javascript 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 choice 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 systems, 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 Audit Component and Audit Browser

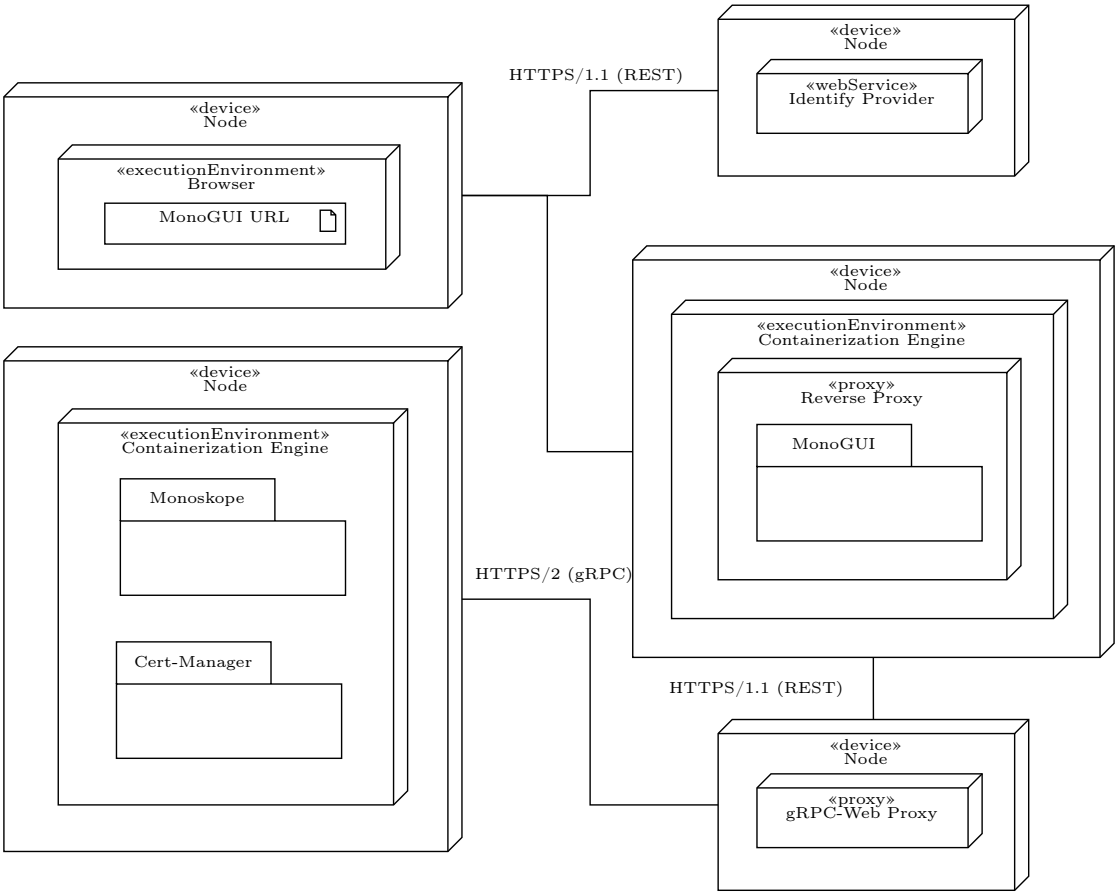

Figure 8.1: Deployment view diagram

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 k8s node. Since m8 already uses Emissary-Ingress it will be used as the gRPC-Web proxy. Dex 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. 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 should work but not tested)

For a sandboxed run VirtualBox and Ubuntu can be used. Providers like LinuxVMImages offer a ready to boot images, that runs with zero installation and configuration efforts.

- Standard userspace utilities like `bash`, `curl`, `tar`, etc... should be available natively but can always be changed to alternatives by setting the corresponding environment variable. For example replacing `curl` with `wget` will be as follows:

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

Shell 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

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

All variables can be found under the directory `deploy/setup` and set with defaults for ease of use.

8.1.2 Install

Open the directory `deploy` and follow the following instruction[1].

1. Create a local cluster and deploy all resources:

[1] If for some reason a command should fail the system and commands are ensured to be idempotent. Just retry.

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

Shell 8.2: Deploy all resources to a local cluster

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

```
# MacOS
$ make turst-m8-ca
# on Ubunte please add the \gls{ac:ca}
# to the browser specific store
```

Shell 8.3: Trust m8 certificate authority

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 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 8.5: Create port-forwards to route local request

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

```
$ make mock-data
```

Shell 8.6: Populate the Event Store with some data

9 Conclusion

IT-Auditing is a part of every regulated organisation lifecycle. Not only dose it insure the organisation compliance with regulators requirements it gives the organisation credibility and trustworthiness. However awaiting periodic audits and hopping for the best outcome is not advisable nor reflect the real world. Auditors have an important role in ensuring confidentiality, integrity and authenticity of various systems.

Supporting Auditors is their mission not only shows teamwork, it is almost always mandatory. Building a compliant system requires a well thought out architecture, that is adequate, efficient, and effective in meeting the organization's objectives. Architecting and developing such system, especially the critical ones, requires a good understanding of the requirements. Mainly the non-functional ones, that might not be of a problem at the beginning.

Auditability tend to be one of them. A great care should be takin when handling such decisions. Choosing the right architecture might make or break the system. As discussed in chapter 5 auditing might be build as a feature, which delegates the complexity on the developer and comes with its owen risk, or built in by design.

Event-Sourcing and the event-sourced architecture discussed in chapter 4 tend to be on top of the list, when talking about audit first systems. Systematic, reliable, and trustworthy recording of events is insured. Events can be easily retrieved from the Event Store and manipulated as desired. In chapter 6 a real world use-case of an event-sourced system missing the auditing functionality was examined and further developed to support a FinTech organisation in its compliance needs. The Audit Browser implemented in chapter 7 showcase how the implemented Audit Component can be utilised as well as offer a friendly and intuitive GUI for the auditors of the organisation. The Audit Component can be taken as is and integrated into another Event-Sourcing system. The only thing to adapt are the domain specific formatters. Satisfying all Audit Log specification as

described in chapter 3 make it also possible for all kind of integrations like the case with Auditing 2.0 discussed in section 3.4.

However one of the auditors request was answering questions like who attempted accessing restricted information or tried deleting business objects, without having any business doing so. Such questions were intentionale skipped, as this is a known limitation of Event-Sourcing and mainly as it contradict with the definition of Audit Log [3.3]. And even if required, this limitation must be handled with or without Event-Sourcing as discussed in section 5.1.3.

If done correctly (which is the hard part) the implementation of other options compared to Event-Sourcing shouldn't differ much. Collecting log entries is the same wether they are coming from an Event Store or an Audit Log database. It all comes down to the system requirements. Is audit logging an added feature? and human errors and bugs are not much of a harm and other architectures are of a greater benefit, then there is no extra advantage to use Event-Sourcing when it comes to audit logging. However having a log of all changes by design and the assurance that the history audited is the actual system history and represent the actual business objects with ability to restore it and test it on a real system without needing to replay commands or script any behavior gives Event-Sourcing the bigger advantage.

It goes without saying, that the mentioned is valid for all options and ways of implementing an audit first system. The only thing that differ is the complexity delegation. Either at the beginning, as is the case with Event-Sourcing, or throughout the entire lifecycle of the system. Other features like Event-Sourcing's complete rebuild, temporal query and event replay are much harder to implement if not planned correctly.

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. (2017). – URL https://www.infoq.com/presentations/netflix-scale-event-sourcing

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

[Fowler 2004b] Fowler, Martin: *Effectivity*. 03 2004. – URL https://martinfowler.com/eaaDev/Effectivity.html. – (Accessed on 02/22/2023)

[Fowler 2004c] Fowler, Martin: *Temporal Object*. 03 2004. – URL https://martinfowler.com/eaaDev/TemporalObject.html. – (Accessed on 02/22/2023)

[Fowler 2004d] Fowler, Martin: Temporal Property. (2004), 03. – URL https://martinfowler.com/eaaDev/TemporalProperty.html. – (Accessed on 02/22/2023)

[Fowler 2005] Fowler, Martin: Event Sourcing. In: *martinfowler.com* (2005). – URL https://martinfowler.com/eaaDev/EventSourcing.html. – (Accessed on 12/24/2022)

[Fowler 2017] Fowler, Martin: *What do you mean by "Event-Driven"?* 02 2017. – URL https://martinfowler.com/articles/201701-event-driven.html. – (Accessed on 02/21/2023)

[Gantz 2014a] 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

[Gantz 2014b] Gantz, Stephen D.: Chapter 1 - IT Audit Fundamentals. In: Gantz, Stephen D. (Hrsg.): *The Basics of IT Audit*. Boston : Syngress, 2014, S. 1–19. – URL https://www.sciencedirect.com/science/article/pii/B9780124171596000018. – ISBN 978-0-12-417159-6

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

[Google 2015] Google: 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)

[Monoskope-Authors 2021] Monoskope-Authors: *finleap-connect/monoskope: AuthN & AuthZ for Kubernetes multi-cluster, multi-cloud environments*. 10 2021. – URL https://github.com/finleap-connect/monoskope. – (Accessed on 02/10/2023)

[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`

Glossary

RESTful the REST architectural style defines six guiding constraints. When these constraints are applied to the system architecture, it gains desirable non-functional properties, such as performance, scalability, simplicity, modifiability, visibility, portability, and reliability. A system that complies with some or all of these constraints is loosely referred to as RESTful.

Angular is a TypeScript-based, free and open-source web application framework.

Audit Browser is a POC implementation to utilise the audit API provided by the Audit Component... more under chapter 7.

Audit Component is the technical implementation of an audit log API for Event-Sourcing in m8... more under chapter 6.

Auditing 2.0 also knowen as continuous auditing, describes the usage of advanced process mining techniques to enable a new form of auditing. More under section 3.4.

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

Command Query Responsibility Segregation a pattern that separates read and update operations for a data store..

Dex is an identity service that uses OpenID Connect to drive authentication for other apps..

Docker is a platform designed to help developers build, share, and run modern applications..

Domain-Driven Design is an approach to software development that centers the development on programming a domain model that has a rich understanding of the processes and rules of a domain..

Emissary-Ingress is an open-source Kubernetes-native API Gateway + Layer 7 load balancer + Kubernetes Ingress built on Envoy Proxy..

Event Store is the database behind Event-Sourcing to store the state events... more under section 4.1.2.

Event-Sourcing is a pattern for storing data as events in an append-only log... more under chapter 4.

FinTech a portmanteau of "financial technology", refers to firms using new technology to compete with traditional financial methods in the delivery of financial services..

GO is a statically typed, compiled high-level programming language designed at Google. It is syntactically similar to C, but with memory safety, garbage collection, structural typing, and CSP-style concurrency..

gRPC-Web A JavaScript implementation of gRPC for browser clients..

Helm helps you manage Kubernetes applications — Helm Charts help you define, install, and upgrade even the most complex Kubernetes application..

HPACK a compression format for efficiently representing HTTP header fields, to be used in HTTP/2.

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..

Kind is a tool for running local Kubernetes clusters using Docker container “nodes”..

Kubectl The Kubernetes command-line tool, kubectl, allows you to run commands against Kubernetes clusters.

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

LinuxVMImages Boot ready os images to run with VirtualBox or others. See for more information.

Make is a tool which controls the generation of executables and other non-source files of a program from the program's source files..

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 is a simple identity layer on top of the OAuth 2.0 protocol. It allows Clients to verify the identity of the End-User based on the authentication performed by an Authorization Server, as well as to obtain basic profile information about the End-User in an interoperable and REST-like manner..

React is a free and open-source front-end JavaScript library for building user interfaces based on components..

Step-cli is an easy-to-use CLI tool for building, operating, and automating Public Key Infrastructure (PKI) systems and workflows..

The Twelve-Factor App is a methodology for building software-as-a-service applications. These best practices are designed to enable applications to be built with portability and resilience when deployed to the web..

Version Control also known as source control, is the practice of tracking and managing changes to software code.

VirtualBox ox is a powerful x86 and AMD64/Intel64 virtualization product for enterprise as well as home use..

Vue.js is an open-source model–view–viewmodel front end JavaScript framework for building user interfaces and single-page applications..

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