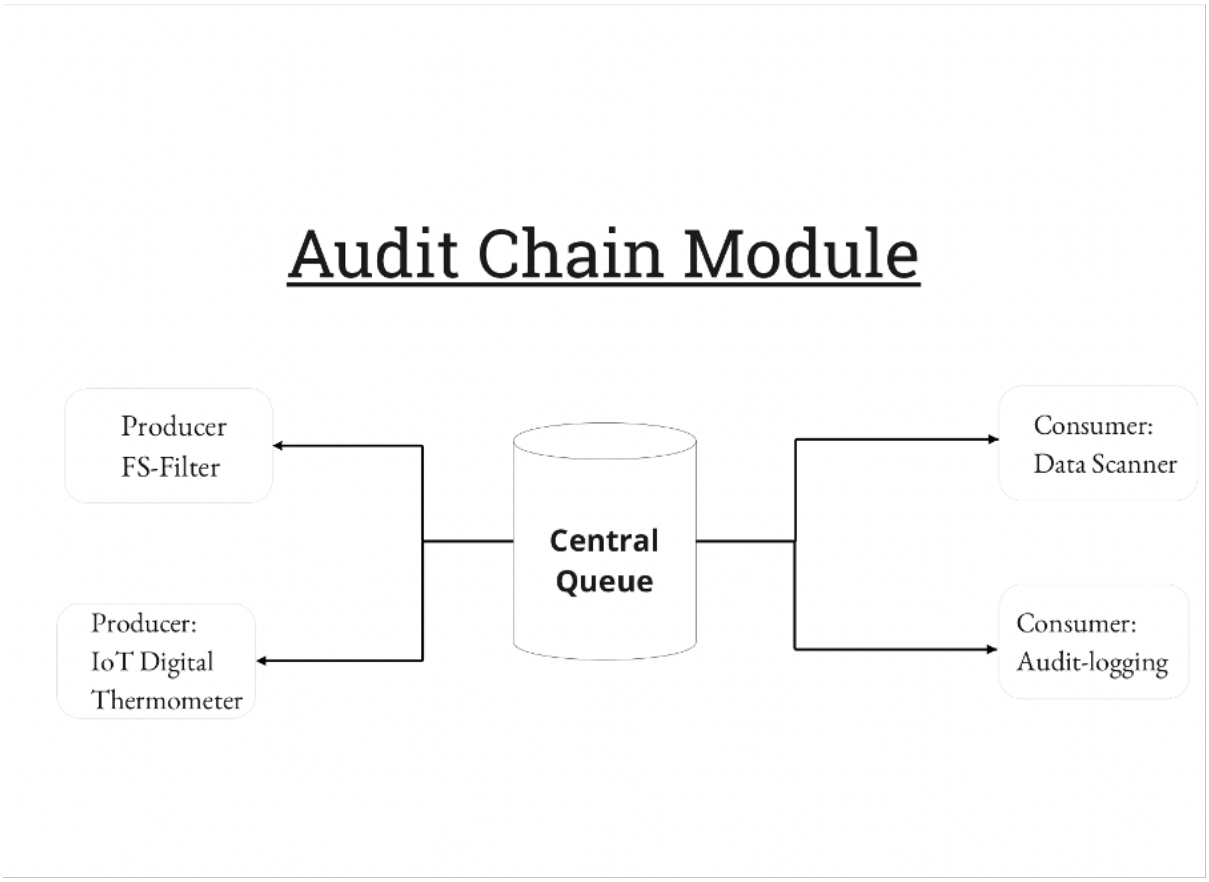
SOFTWARE ARCHITECTURE DESCRIPTION

AMOS SS2022

Project 2 – Audit Chain

Runtime Components



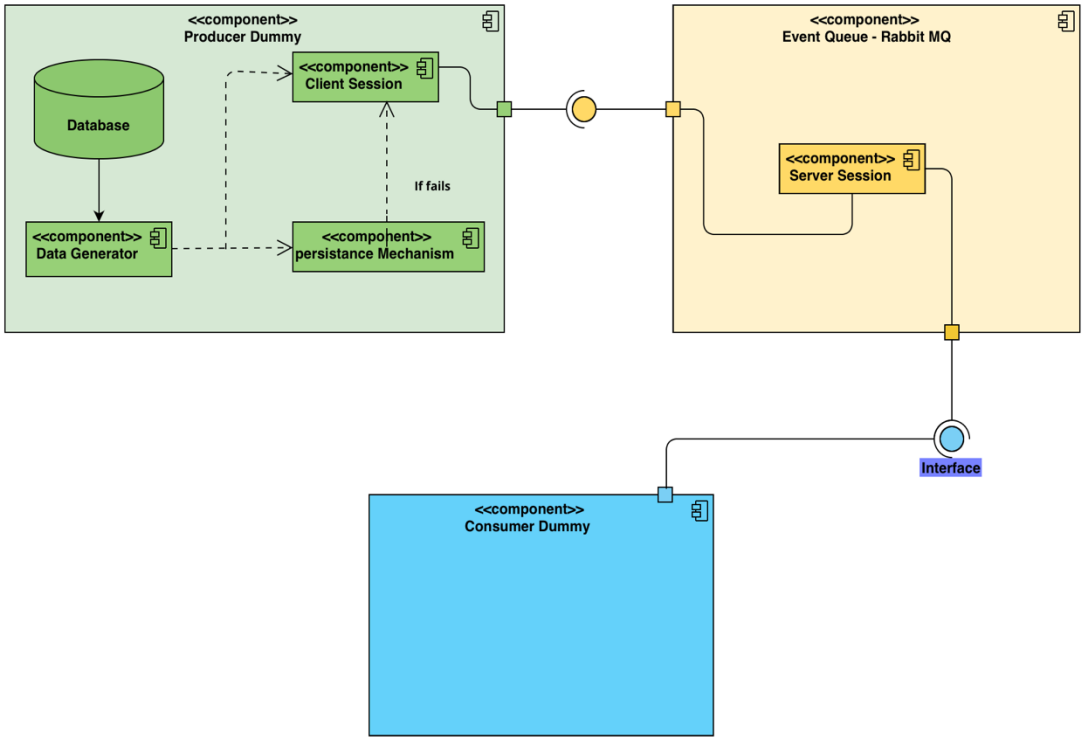
The main idea of our project is that events of any kind like IoT, file systems, and measurement loggers should be transmitted securely via the network. So, there is also a central event queue, which records these events and for further steps provides. Furthermore, services can then register at the event queue to pick up the events and to be able to process, also via a network. All changes to files (timestamp, size, permissions) in a defined

File structures are recorded as an event. These are securely transmitted to the central queue on another node, and they are taken from the central queue by a process (from a third Node off), stored audit-proof (e.g., in a blockchain technology secured Storage), and then deleted from the queue.

The producer is the creator and the sender of events. Also, the events are recorded serially, and the central event queue is transmitted via the network queue. Moreover, as it concerns the queue, in this case, events are accepted over the network and securely cached and offer the possibility for producers and consumers to register.

Finally, we have the consumer who uses the events from the queue to trigger further actions based on them. These actions can be, for example, in the simplest case, the secure permanent storage of events, but also further processing steps such as process control.

Code Components



We have used the component diagram to show different components of our code solution. Our code is basically comprised of 3 major components, producer dummy (client), rabbit MQ (Event Queue), and consumer dummy.

The producer dummy operates through the dependencies among its 3 sub-components database, data generator, and persistence mechanism (storage buffer). When the producer dummy is triggered, the data generator catches data events (message) from the database and then forwards it to the client session and persistence mechanism. The persistence mechanism is a contingent component that stores the last data event and gets triggered in case of any failure.

After the producer dummy successfully generates an event, it goes to rabbit MQ (event queue). Rabbit MQ is a 3rd party program that we are using as a component for queuing events.

Lastly, after rabbit MQ queues events successfully, the data event moves to the consumer dummy whose details are still in discussions with the industry partner.

Technology Stack

Tool	Type	Version
GitHub + Git	Version Control	
Java 16(SDK)	Software Development Kit	16 of the Java SE
Python	Python3	3.10.4
RabbitMQ	An open-source message-broker software	TBD
Linux	Ubuntu	18.04

A Summary of the underlying technology stack

First, a tech stack combines technologies a company uses to build and run an application or project. It typically consists of programming languages, frameworks, databases, front-end tools, back-end tools, and applications connected via APIs. It has become essential for building easy-to-maintain, scalable web applications.
In our project, we use different technologies to fulfill our aim.
First, we use RabbitMQ, an open-source distributed message broker that facilitates efficient message delivery in complex routing scenarios.
Also, it employs a push model and prevents overwhelming users via the consumer configured prefetch limit. RabbitMQ natively implements AMQP 0.9.1 and uses plug-ins to offer additional AMQP 1.0, HTTP, STOMP, and MQTT protocols, and it officially supports Java, JavaScript, PHP, Python, Spring, etc. It also supports various dev tools and clients using community plug-ins. We use RabbitMQ to process high-throughput and reliable background jobs, plus integration, intercommunication between and within applications, perform complex routing to consumers and integrate multiple applications and services with nontrivial
routing logic. We prefer it instead to Kafka because with Kafka we would introduce way more complexity. It sends messages to users that these messages are removed from the queue once they are processed and acknowledged.
On the other hand, Kafka is a log that uses continuous messages, which stay in the queue until the retention time expires. Finally, RabbitMQ employs the innovative broker/dumb consumer model while Kafka uses the dumb broker/innovative consumer model. Kafka doesn't monitor the messages each user has read. Instead, it retains unread messages only, preserving all messages for a set time. Consumers must monitor their position in each log.
Moreover, we use Java, a multi-platform, object-oriented, and network-centric programming language. We chose it as the first option instead of Python because Java is a statically typed and compiled language that offers limited string-related functions. Python is a dynamically typed and interpreted language that provides many string-related parts.