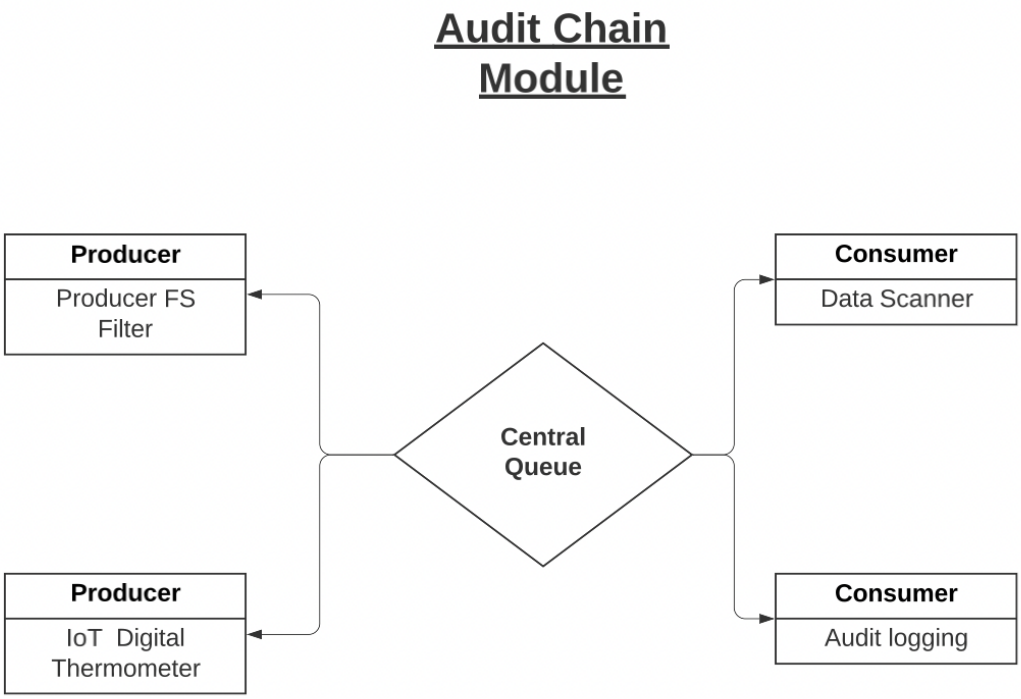
SOFTWARE ARCHITECTURE DESCRIPTION

AMOS SS2022

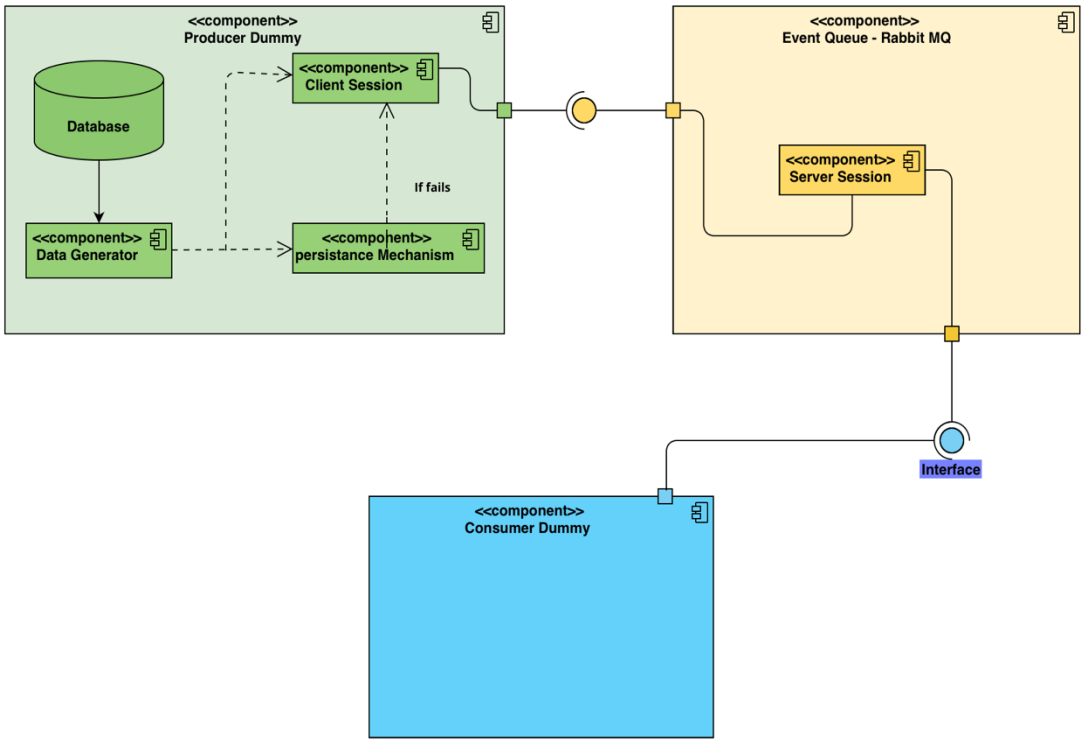
Project 2 – Audit Chain

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) a defined.
File structures are recorded as an event. These are securely transmitted to the central queue on another node, taken from the main 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 possibilities 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, securely cached, and offer the possibility for producers and consumers to register.
Finally, the consumer uses the events from the queue to trigger further actions based on them. In the simplest case, these actions can be the secure permanent storage of events and additional processing steps such as process control.

Code Components



We have used the component diagram to show different components of our code solution. Our code comprises three significant members, producer dummy (client), RabbitMQ (Event Queue), and consumer dummy.
The producer dummy operates through the dependencies among its three 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 failure. After the producer dummy successfully generates an event, it goes to RabbitMQ (event queue). RabbitMQ is a 3rd party program we use as a component for queuing events. Furthermore, we checked for data replication in RabbitMQ. Lastly, after RabbitMQ queues events successfully, the data event moves to the consumer dummy.

So, the first step was to build the consumer dummy by developing the prototype firstly and giving the acknowledgment that message had been received. The consumer receives messages from RabbitMQ, and those messages are submitted in sequenced numbers. Also, we followed a specific strategy to save messages in the correct order. And then, we integrate blockchain data structure in consumer data, where we hold the received messages in data structure and ensure the correct sequence of incoming messages. Finally, we provide the flow of the consistent message.
Furthermore, we set up Docker because we wanted our system resources to be used efficiently, which means consuming less main memory and being accessible to port so that the software works better. Also, we implemented GUI by using Python.

Technology Stack

TOOL	TYPE	VERSION
Java 16(SDK)	Software Development Kit	16 of the Java SE
Python	Software Development Kit	3.8.0
RabbitMQ	An open-source message-broker software	TBD
Docker	Deployment	
JUnittest	Dependencies	
Maven	Build Tool	
GUI	Graphics-based operating system	

A Summary of the underlying technology stack

First, a tech stack combines technology companies use 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 to Kafka because we would introduce way more complexity with Kafka. It sends 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 an interpreted language that provides many string-related parts. Finally, in the 4th week of the project and after consultation with our industry partner, we decided not to proceed with the python programming as it would be too time-consuming to implement every feature in python.

Furthermore, we implemented JUnit as a unit testing open-source framework for the Java programming language. We use this framework to write and execute automated tests. However, in Java, test cases must be re-executed every time a new code is added. This is done to make sure that nothing in the code is broken.
At this point, we want to emphasize that as users, we wanted to ensure the efficiency of our system's resources, i.e., consume less main memory, easy to port, etc., to make the software work better. So, we proceeded to create and implement the docker. The Docker platform is open-source for developing, shipping, and running applications. By using Docker, you can separate your applications from your infrastructure and deliver software more quickly. You can manage your infrastructure using Docker like you work your applications.
After that, we created a Maven repository. Maven repositories are places where maven artifacts are stored and managed (either locally or remotely). Antiques can be retrieved and included in other maven projects in a maven repository.
It is possible to store different types of build artifacts and dependencies in Maven repositories, but the truth is that there are two kinds of them:

- local and
- remote

The local storage is a directory on the computer where Maven runs; it caches remote downloads and contains temporary build artifacts that you have not yet released.
So, we added the JaCoCo maven plugin to have a report of the test coverage and a plugin to the maven project of the producer dummy to auto-generate the Javadoc (Plugin Name: maven-Javadoc-plugin v3.2.0). Moreover, we added a README to know which maven command to execute to generate the Javadoc.
However, we realized that it was needed to create a big Maven project, which contains all of them, and to throw out the others existing (after having put them in the big new maven project) because our code was spread in the project over some maven repositories. This did not allow us to re-use code in another part of the project.
Having achieved our goal, we decided to implement a graphics-based operating system that uses icons, menus, and a mouse to control interaction with the system.
With this friendly visual environment, the user can perform any action without knowing how to program it. Windows, macOS, and Android environments are examples of GUIs in which commands are sent using gestures and mouse movements without requiring any coding. We tried to do with GUI to enable the user to send orders to the consumer. We wanted the communication between the Consumer Dummy with RabbitMQ and GUI to be concurrent so that the SW is reliable. Finally, except for the friendly GUI, we wanted to implement the GUI methods to improve the Blockchain's functionality.