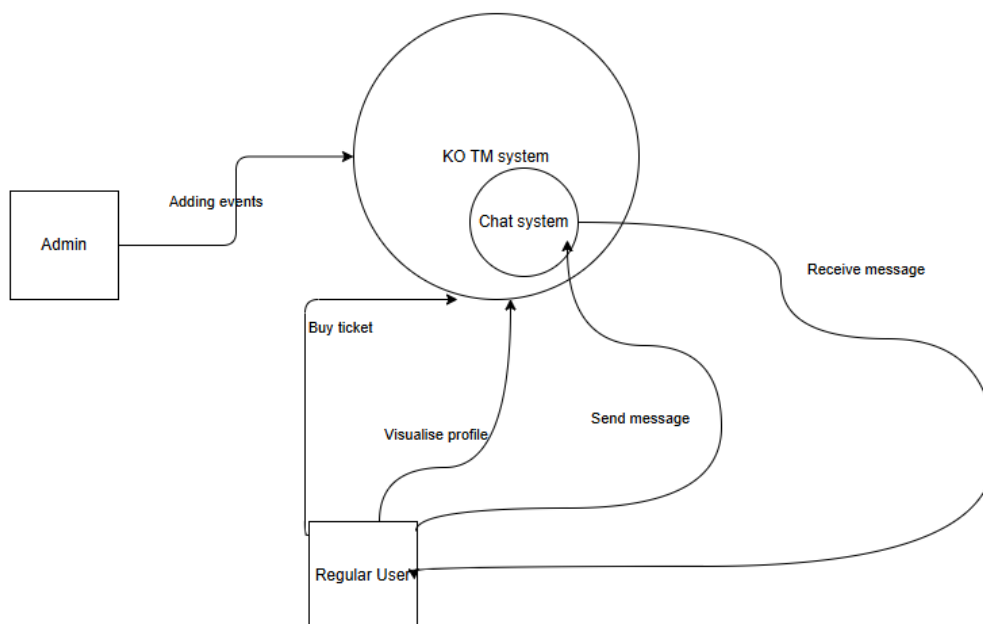# Knock-off Ticket Master App

The KO Ticket Master App is a web application that lets users browse through a number of events that are available. In order to be able to purchase tickets for a certain event, a user must register or log into his account. After logging in, the user can buy a ticket and visualize all the tickets that he bought on his profile page. Additionally, users may chat regarding certain events in the announcement section, which can be used by organizers to engage with the community.
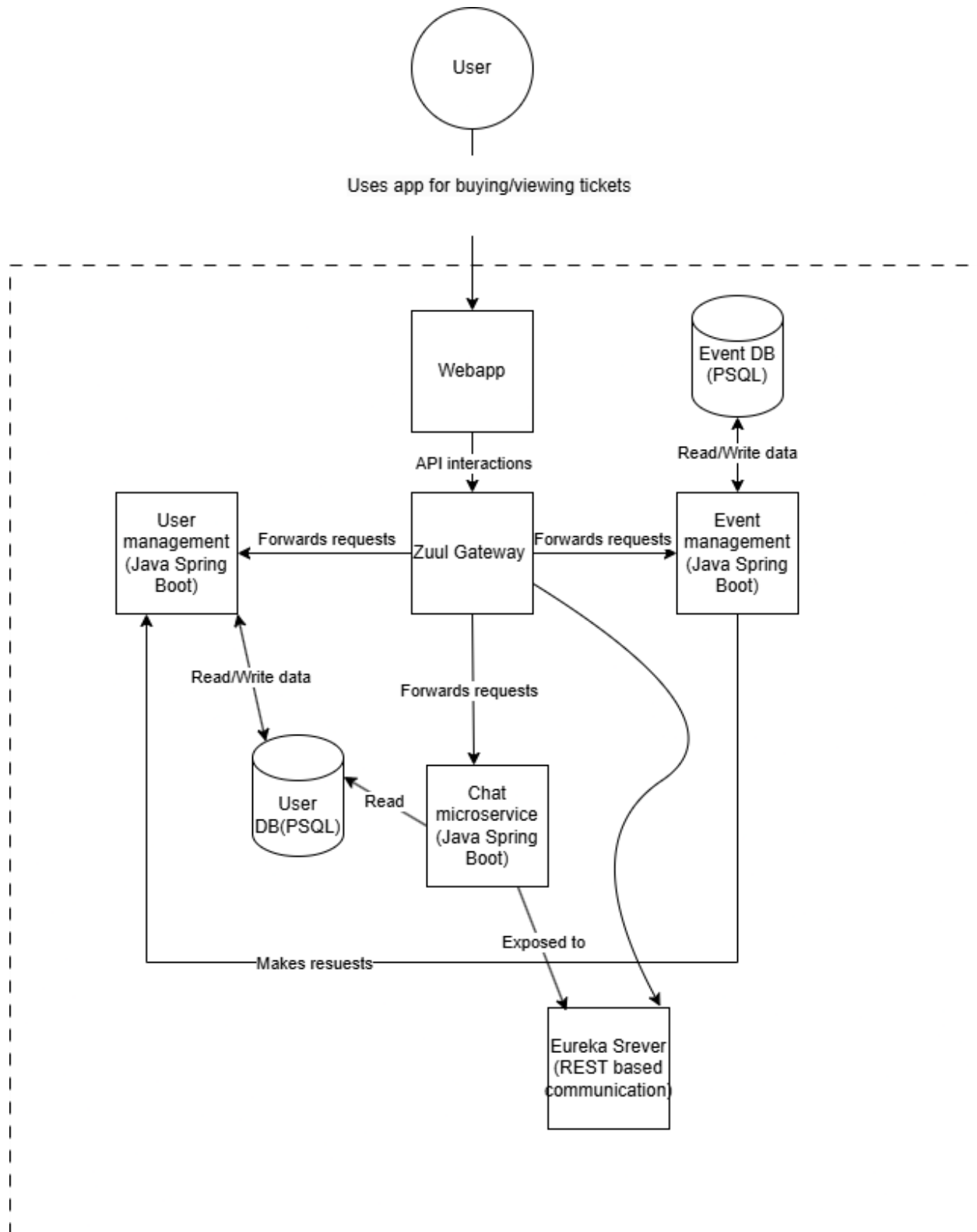
Technologies:

The building block of the System is Spring Boot, which is used in all the sub-systems of KO Ticket Master. All the subsystems are individually maintained as micro-services. Communication between the sub-system is realized through the usage of Eureka (Spring Cloud Netflix Eureka) together with Zuul (Zuul API Gateway), for appropriate routing, micro-servicies communication, load-balancing and overall resilience of the system as a whole. The community chatting functionality is achieved through the usage of a pre-implemented tool, which under the hood makes great usage of kafka and websockets in order to achieve real-time communication between multiple users. In order to relieve the system of high computational efforts, AWS Lambdas are the perfect much for such tasks. In order to wrap things up and bundle them together, docker is used in order to package each sub-system into it's own container and under a common project providing consistent and scalable deployment environment.
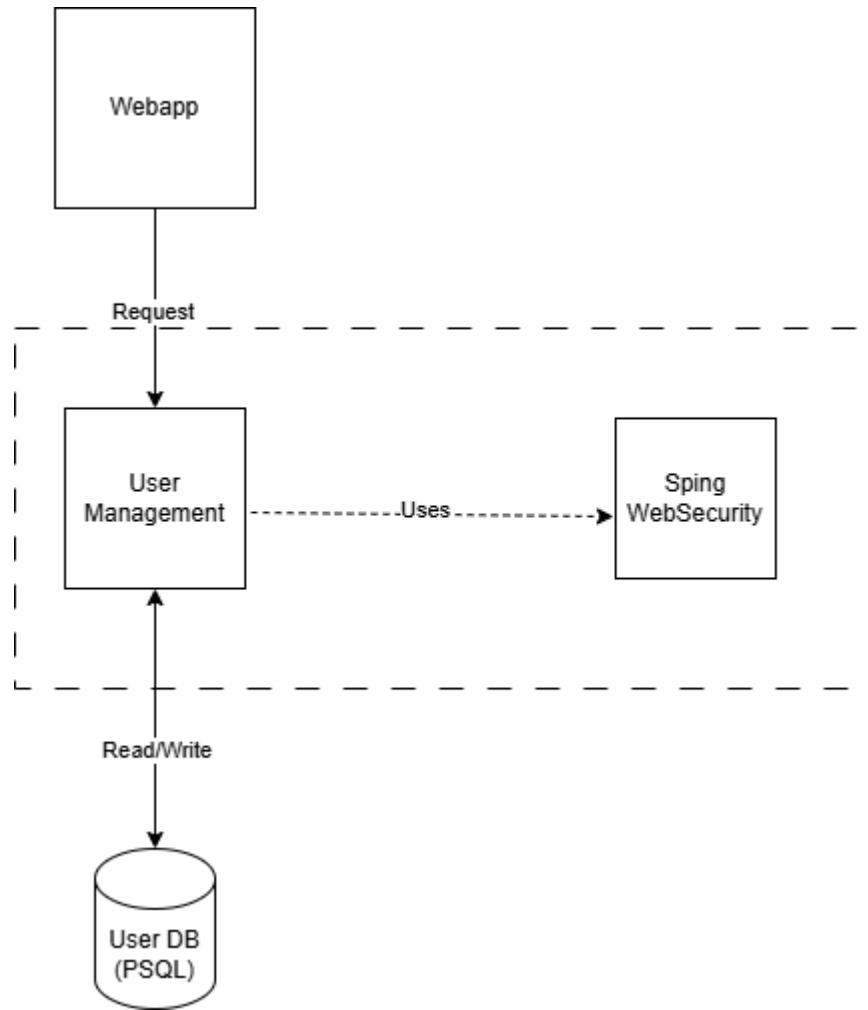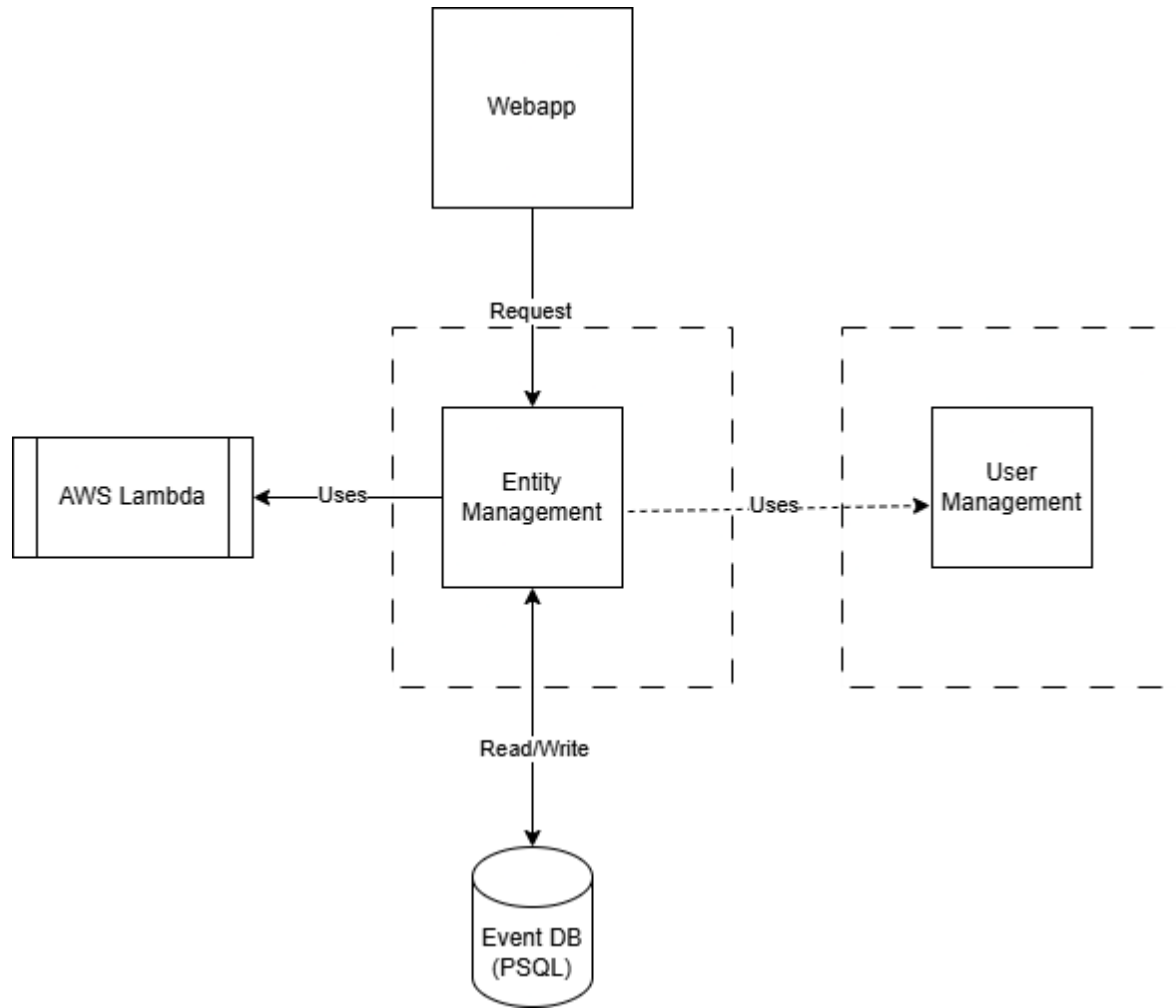
Diagrams:

System diagram:

Container Diagram:

User

Uses app for buying/viewing tickets

Webapp

Event DB
(PSQL)

API interactions

Read/Write data

User
management
(Java Spring
Boot)

Forwards requests

Zuul Gateway

Forwards requests

Event
management
(Java Spring
Boot)

Read/Write data

Forwards requests

User
DB(PSQL)

Read

Chat
microservice
(Java Spring
Boot)

Makes resuests

Exposed to

Eureka Srever
(REST based
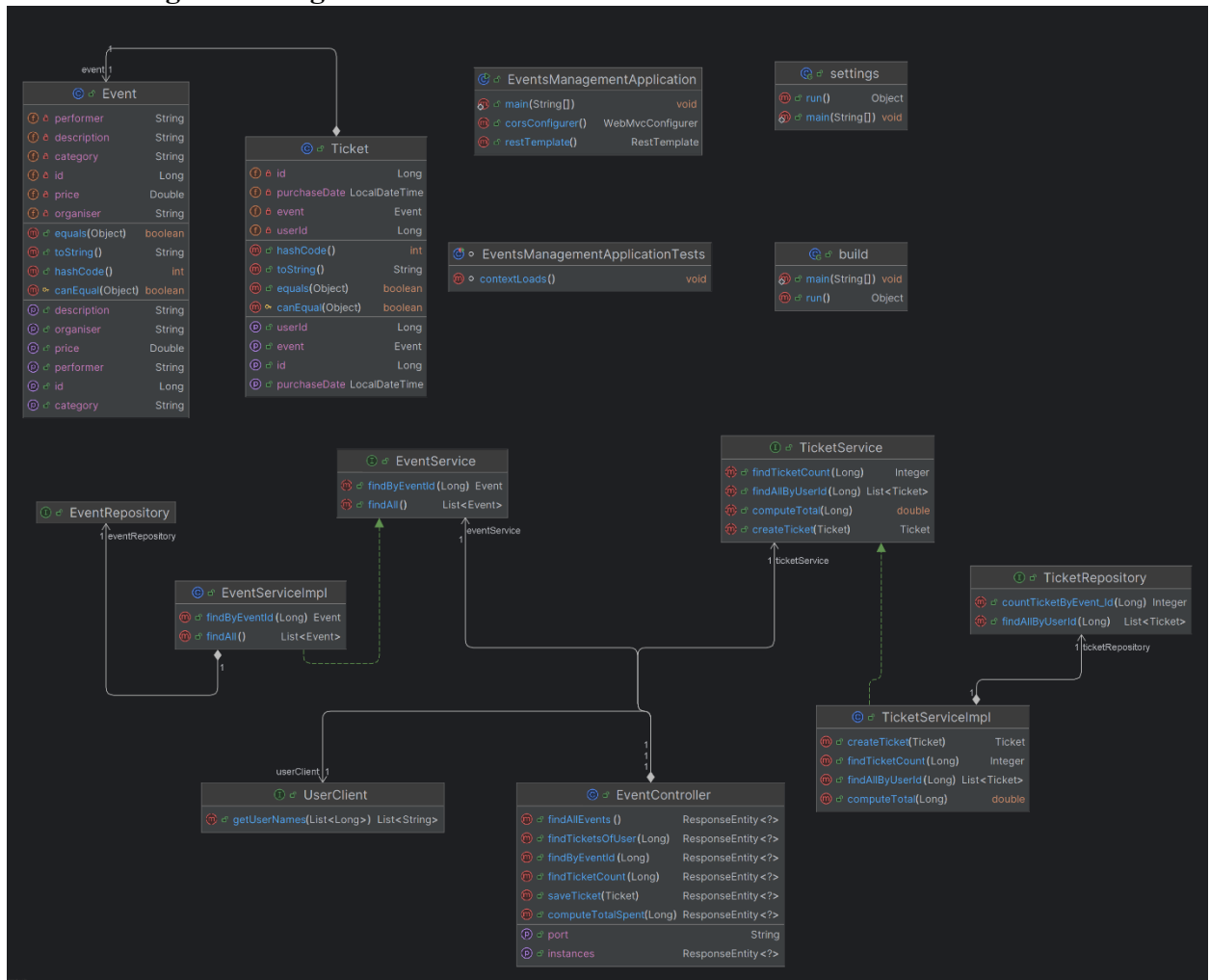communication)

User Microservice Component Diagram:

Event Microservice Component Diagram:

UML Diagrams:

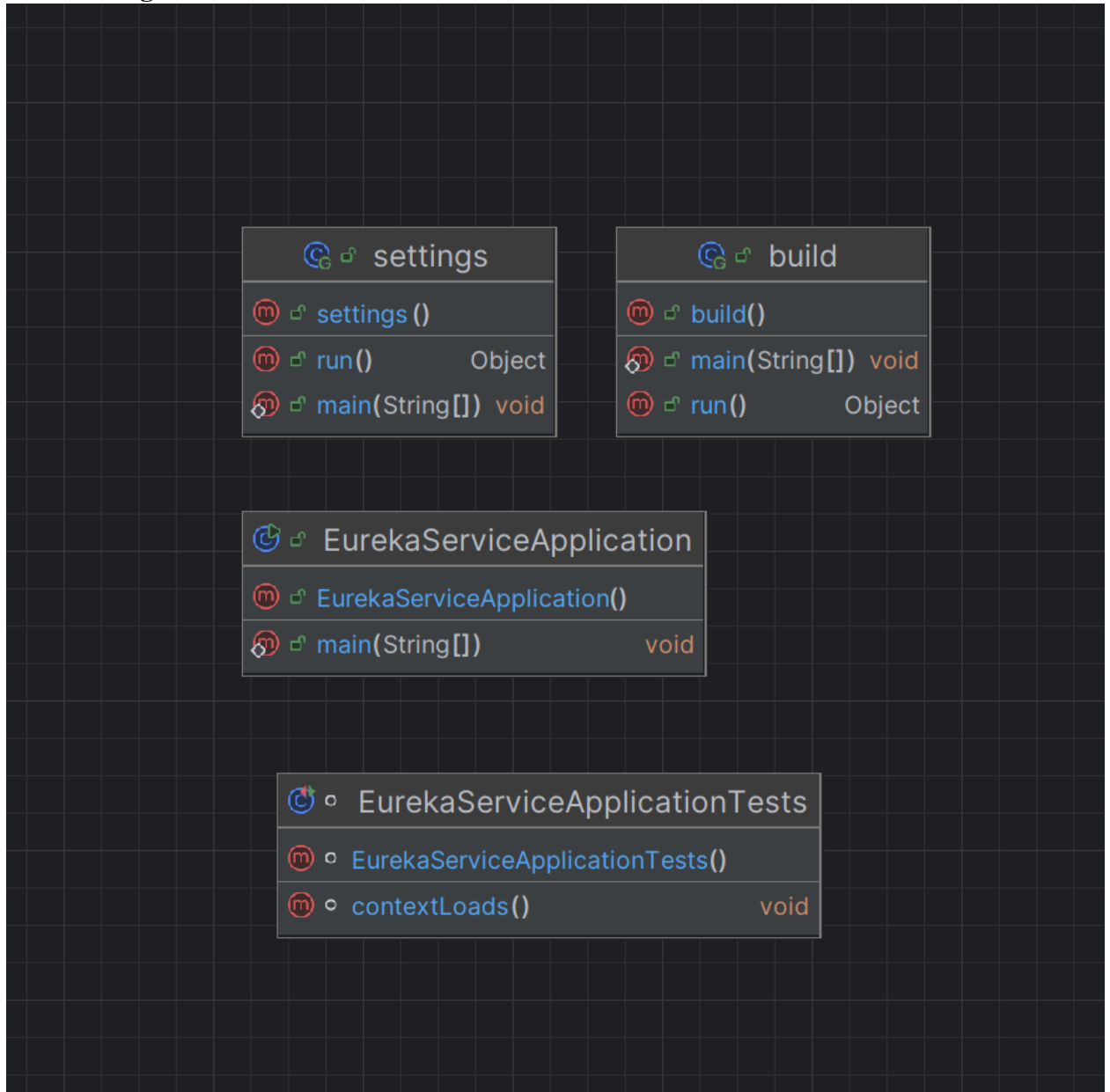**1. Event management diagram**

2. User management diagram
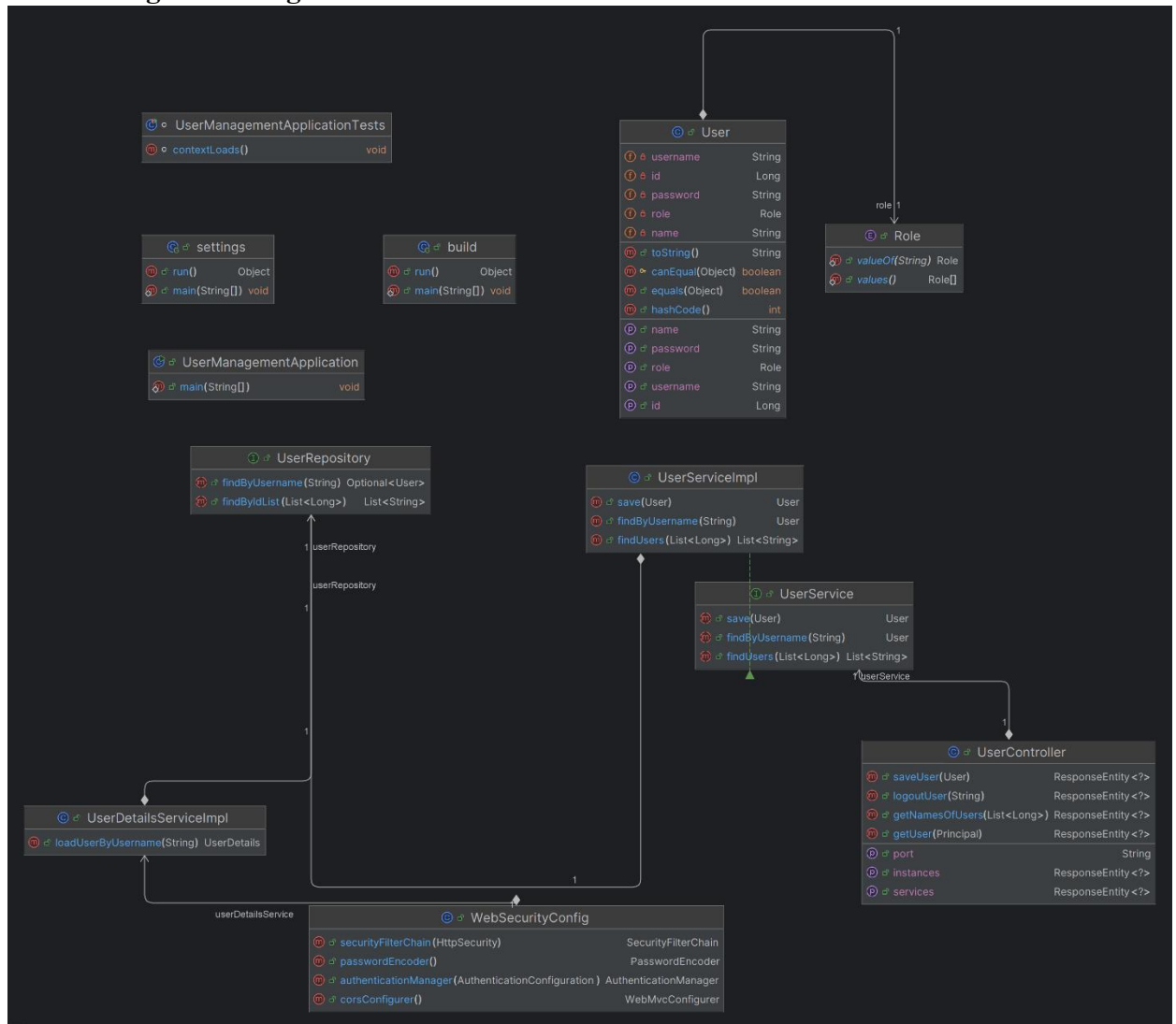
3. Messaging service diagram

KafkaApplication

ⓜ main(String[])   void
ⓜ chatTopic()   NewTopic

KafkaApplicationTests

ⓜ contextLoads()   void

MessageEndpoint

ⓜ send(Message)   void
ⓜ consumer(Message) void
ⓜ join()   Flux<Message>

Message

ⓕ time   Instant
ⓕ userName String
ⓕ text   String
ⓟ text   String
ⓟ userName String
ⓟ time   Instant

MessageSerializer

ⓜ serialize(String, Message) byte[]

MessageDeserializer

ⓜ deserialize(String, byte[]) Message

4. Eureka diagram

5. Zuul Diagram