CQRS - ES

Technical Architecture - Backend

**Creation date: 07th February 2020**

**Modifications**

|  |  |  |
| --- | --- | --- |
| **Author** | **Date (mm/dd/yyyy)** | **Note** |
| Athénaïs | 02/07/2021 | Document created |
| Athénaïs | 02/08/2021 | Document reviewed and edited |

// This document should give a technical overview of the project, the goals and the steps to achieve it.

SERVICES

* **CQRS :**
  + How ? a segregated architecture from the start (from the API controllers)
  + Why ? to avoid an overloaded DB request system
  + What language? Golang, as it is utterly maintainable, fast to compile and the API needs to be highly capable of dealing with multiple requests.
* **Event Sourcing :**
  + How? With every action an event is created. Those events should be precisely defined by us, thus creating a highly modulable system
  + What language? Golang, as it part of the same API as the CQRS architecture
* **Message Broker**
  + What? A system to handle all the events ( which are of course more numerous that if we were calling each step differently )
  + How? Using Rabbit MQ, an open-source, widely-used message broker. This technology seemed to fit the best, as it has proven to be reliable and stable.
* **Projectors:**
  + What ? Projectors are a finite state object at a certain point in time. We chose to separate the projector logic into another project, thus becoming the consumer to the Rabbit MQ?
* Data management:
  + Every user will be saved in a MYSQL DB. A SGDBR is best suited for an entity that can and might have numerous relationships.
  + Every other data ( orders for now ) will be stored in elastic search, which allows quick and efficient search.

MAIN STEPS

Define the data structure (which types and MySQL scheme)

Define the API architecture

Implement driver SQL connection

Implement Elastic Search connection

Implement router

Implement basic CRUD

Implement auth system (JWT)

Set up message broker

Create and set up consumer

Link all services together

Implement tests