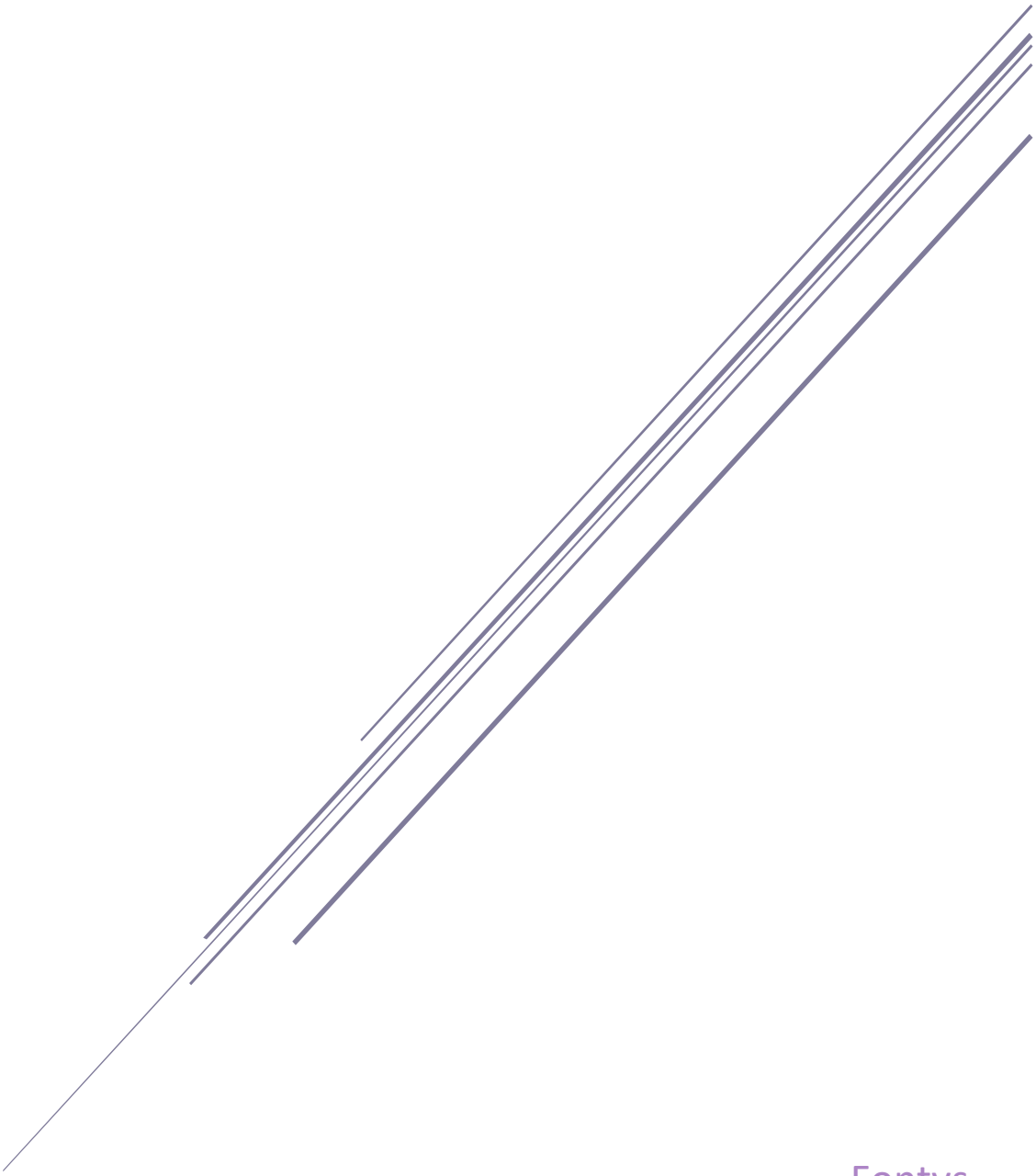
SCALABILITY

Reasoning, Conclusion & Recommendations



Fontys

HBO ICT

OPENING

In this document I will explain why I made certain decisions in my implementations and why I did not go for a different approach. The topics that we will be going over are Architecture, Frameworks and Scalability.

CHALLENGE

Water polo is a niche sport in the Netherlands. Unlike football, hockey, or tennis where each town might have one club of the three sports and big cities might have multiple within the same sport. This results in clubs easily finding opponents to play practice matches outside the main competition. Water polo on the other hand besides having a smaller group of athletes also has the problem that it is restricted to towns and cities with a swimming pool which is also competition approved. This combination leads to difficulty finding practice matches, the idea for this project is an integrated system where teams can find each other to play practice matches, contact each other, find relevant personnel like referees, keep track of matches themselves and create an environment which promotes competitive play among the regions.

REASONING

FRAMEWORKS

For this individual project I had planned to use a combination of Vue.JS and Java Spring. As frontend work was not used much during the project, I will go over it briefly but Vue.JS was chosen over Plain JS and ReactJS due to its recently new released version, which improved security and performance for larger scale applications over its previous iterations.

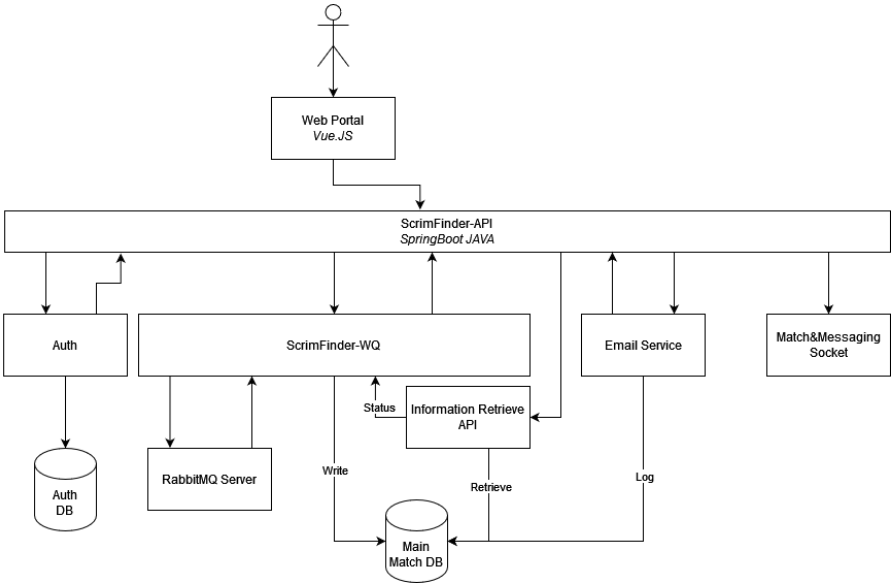
For backend work I used Java Spring. The alternatives for this project came down to a decision between it and Python Django. Spring was chosen over Django or alternates because of the following three reasons.

Compatibility: Both Java and Python are commonly used in Data science projects to evaluate, create, and deploy models. For this reason, these two where a standout over alternatives like C# who provide a similar development pattern to Java.

Ease of implementations: Java is an object-oriented language which can be a drawback, it leads to complicated statements for smaller applications which can be more easily done in Python. However object-oriented Java comes with rules and due these rules code quality is more easily ensured.

Security: Java Spring Security is a built-in package within Java Spring which allows not only for security of your applications from foreign applications but also provides a robust authorization package that works well with cloud services.

ARCHITECTURE



For this project I went for a worker-queue to optimize the way SQL requests are managed. The reason for this over alternatives like an event driven system used in my group project came down to two factors.

Batching: The project revolves around the ability to create large batches of matches to create tournaments. These tournaments do not need to be provided in real-time or as soon as the tournament is created because these are often planned plenty of time before the event happens. This removes the need for an event driven system for creating these batches. Instead, a web-worker queue was used through RabbitMQ which allows for a large batch of matches to be created if a batch becomes too large a new worker can be created to manage newly incoming batches without slowing down other workers.

RabbitMQ moto is *“Messaging that just works.”* and for that same reason it is used in this project over alternatives for messaging like Apache Kafka. Event though Kafka can stream about 10x the messaging per second over RabbitMQ, RabbitMQ excels in simple use cases which leads to easy to maintain code. The reasoning why the performance improvement wasn’t much of importance in the decision is due to the number of active players in the Netherlands. The latest number from 2014 there were a confirmed 27 thousand active players adjusting for population growth there wouldn’t be more than 50 thousand active players in the Netherlands in 2023 with there being 130 thousand member in the whole of the KNZB.

Furthermore, Spring comes with a package providing extra support for RabbitMQ. Lastly privacy RabbitMQ discards its messages as soon as the message is acknowledged, unlike Kafka which defaults to logging messaging.

CONCLUSION & RECOMMENDATIONS

This chapter I will put my own opinion on how I would have managed things differently if had more time or another chance. A substantial portion of decisions I had to make regarding this topic were made in the initial stages of the semester as a result lots of them were made on assumptions I had on other topics. For example, I assumed authorization would be important if a part of the project would be overseen by a cloud service.

Overall, I did learn about the differences between architectural styles when and when not to use the different styles, and what kind of research needs to be done before starting a project. A large discovery regarding this research is the usage of requirements in making decisions. The decisions to use the more lightweight and simpler RabbitMQ was made because I research how many user I would need to service.