Technical Document

Jordan Radushev - Semester 6

# Introduction

This document contains all the research I gathered during the semester on different topics, regarding the learning outcome I need to fulfill to pass the semester.

## About the project

PlayJ is a web-based music player. I was inspired by Spotify, and PlayJ is in fact like a Spotify clone for the educational purposes of Semester 6 – Enterprise Software Development. The app provides functionalities such as:

* Music streaming over the network
* Creating/modifying personal playlists
* Mocking payments for monthly subscription
* Statistics dashboard

Code Repository URL - <https://github.com/JordanRad/play-j>

## Project Goal

The goal of this project is to gain theoretical and practical knowledge in the field of enterprise software development. Answering the main research questions along with many other subquestions, will teach me the fundamentals of this topic.

## Technology stack

By choosing a technology stack I was driven by the following criteria:

* The frontend framework should be easy-to-use and very minimalistic in terms of size.
* The frontend framework should be popular enough, so there are some utility and component libraries available.
* The backend language should be suitable for microservices development.
* The backend language should be modern and very performant.
* I also prefer statically typed and compiled language
* The backend language should be easily adaptable to third-party services and providers
* The database should be a fast, relational database, because this matches perfectly with the project’s context, since most of the data is structured.

## Primary research outcome

### Frontend

My personal experience (over 2 years) with React is the reason I would not like to use it in my project. And now I still need to choose between Angular (I have used it before) and Vue (never used it before).

Clearly, the new version of Vue (3.x) made the framework even more powerful and React-like which fits the best with the criteria. It is also lightweight and developer friendly which was the main reason for my decision.

### Backend

Previously I have experience mainly with Java and Node JS as server-side languages. Python, Golang and .NET are other popular alternatives as well. But how can I choose the best for my project?

First and foremost for me was the verbosity of the code. Then comes the language specifics such as statically-typed and compiled language. Having this in mind I can cross out Node JS and Python. Java and C# are both statically-typed and compiled languages, however they seem quite verbose to me.

The winner for my project was **Golang**. There are many reasons why I choose exactly Go as main backend language:

* It is quite new and it is gaining more and more popularity
* It is perfect for developing microservices
* It can be easily integrated with many third-party solutions
* It is quite fast and performant which makes it suitable for developing enterprise software

### Database

I have previous experiences with different relational databases such as MySQL, PostgreSQL, H2 in-memory DB, QuestDB etc. I also have some experience with NoSQL such as Firebase and MongoDB and to be honest I would not choose them unless the project requires such an approach (e.g.- contains mainly unstructured data). The case with PlayJ is not like the example, so I focused my research more around the SQL databases.

My internship experience helped me choose a suitable database for the project, since I developed microservices during the internship using Java Spring boot and PostgreSQL. Also, PostgreSQL has many useful features such as JSON native support.

### Conclusion

The technology stack I choose is pretty much new to me in terms of programming languages and frameworks which is a nice challenge for me. What is also important is that both Vue JS and Go are gaining popularity rapidly in the software development field which is a nice opportunity for me to gain knowledge about trending technologies.

# Research

In this section you will get familiar with the main research question regarding the project as well as the theoretical approach towards the problem.

## Research questions

Main research question:

***“How to build a web-based music player in a scalable distributed system on an enterprise level?”***

How to build the application in an efficient way, so it can work smoothly with a huge request load (Imagine it has 750k active users). Application security is also a challenge, because it is very important to build the app in a secured and stable way. The separation of the business logic in the context of microservices is a key detail to achieve the scalability and reliability in terms of functionality.

For more information about functional and nonfunctional requirements, please refer to the [analysis document](https://docs.google.com/document/d/1SvlQG8xyBjV4l1XvzuRqh91Mj4ofaAoMlVE0mwfXufc/edit?usp=sharing).

Along with the initial question, there are many more questions, so the project can be completed properly and successfully. The successive questions that appear immediately are:

* What kind of approach should I use for the project? (Code first, Design first, etc.)
* Should I use an ORM and why?
* Should I place the code in one repository or not and why? (Monorepo vs Polyrepo)
* Should I use some kind of backend Go framework?
* What is Kubernetes and how to use it in the context of microservices?
* What is Docker and what are its applications in software development?
* What is microservice architecture?
* How should I handle the security risks in my code design?
* How to separate my microservices efficiently?
* How to containerize the application effectively?
* What is internal microservice communication and what types of communication is there?

Finding answers to these and many more research questions will give a summarized answer to the main research question.

## Research methods

To find answers to the above mentioned questions, I need to create a strategy of the research methods, provided by the DOT framework.

I conducted my research using:

* Since I make a clone of Spotify, **available product analysis** was a nice starting point. It gave me a perspective of where I should go throughout the development process.
* I needed to gain a lot of theoretical knowledge about microservices, security, best practices in enterprise software development, etc. For this part of the research l**iterature study** and researching some **good practices** was very important for the outcome of the research.
* Then came the **task, security and problem analysis** - start the development process and respectively to fulfill some of my learning outcomes.
* Once the development started growing, I started adding some automated **unit tests** to justify my work and make it part of a CI pipeline. Along with that, I always kept adding E2E manual tests, using tools like Postman.

## Research outcomes

The answer to the main research question may sound something like:

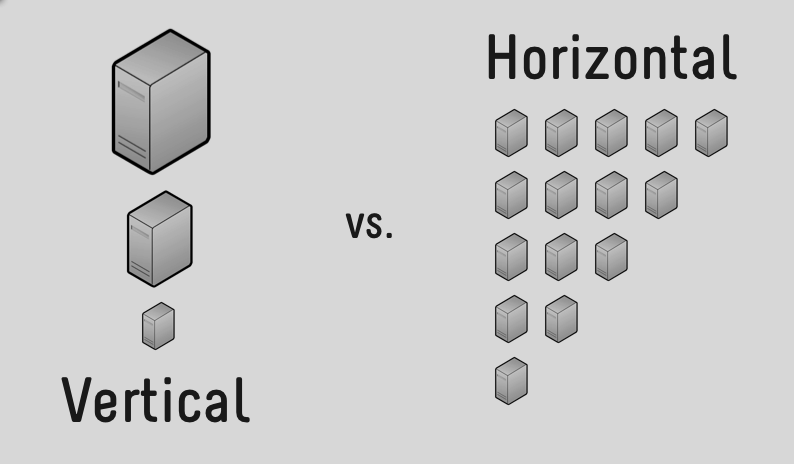
An enterprise level web player consists of many small and efficiently working components. Being combined all these components form a scalable distributed system.This system makes an extensive usage of cloud services like (in my case) Google Cloud Platform and in particular, Cloud SQL for building and maintaining highly scalable databases. Google Kubernetes Engine provides the opportunity to deploy my microservices in an isolated environment on the cloud. GKE offers many features for easy, reliable and secure up/downscaling.

The short answer is:

**Powerful combination of already existing cloud solutions and efficiently built microservices with appropriate tools.**

# Architecture

The project’s architectural structure is vitally important, so the application can be powerful and scalable. This is why the developers have to make wise choices before diving into the code of the application.



The application for my project could be scaled pretty straightforward . There are 2 main types of scaling (horizontal and vertical) which I will apply to make my application reliably scalable. First let me explain the working units of my application:

* The project contains 1 working client application which does not include any scaling at this point
* The project contains 3 working microservices which could be both horizontally and vertically scaled. **Vertical scaling** in terms of adding more hardware to the working machine or respectively adding more workforce (pay more) to the cloud provider and also using **Kubernetes** for automated **horizontal scaling**, meaning that there would be many working replicas of one microservice instance and the traffic will be distributed fairly equal to all working units.
* The project also has 3 databases for the services which could be easily **vertically scaled** up/down by the cloud provider when the amount of data requires it. **Horizontal scaling** is the next step in scaling up databases when we have a huge amount of data. Once we reach the full potential of vertical scaling for the databases, a horizontal one can be introduced.

# DevOps

Development Operations are as important as the development process itself. Utilizing different DevOps tools can save you a lot of time and error handling.

What I managed to add to my project is the following DevOps tools and techniques:

* Extensive usage of Docker in the development process, I have a Docker compose along with images for each microservice
* CI pipelines which trigger common tests of the backend as well as the tests for a particular microservice. I set up a pipeline per service
* CD pipelines which build, tag and push Docker images to a registry (DockerHub in my case)

Apart from the technical aspect of this topic, I managed to set up a suitable software development life cycle during the semester. At any point I knew what I did successfully and mostly covered it at least with some automated and manual tests to verify it and what I should do next. I separated the project into core features (epics in Jira) and tried to implement from the hardest features for me all the way to the easier ones.

# Cloud Services

I have used some cloud services during the development, not only for deployment, but also I used cloud tools to help me do my job easily such as SonarCloud which scans the codebase for bugs, errors and vulnerabilities. Along with the test, it is a nice way to catch the bugs at an early stage and fix them as soon as possible.

I used Google Cloud Platform extensively during the semester as it follows:

* Google Kubernetes Engine for deployment
* Google Cloud SQL for the database
* Google Cloud Storage as a file system to store files such as music files, scripts, and pictures.

# Literature list

**Technology stack**

1. <https://www.monocubed.com/blog/vue-vs-angular/>
2. <https://www.mindinventory.com/blog/technologies-for-mircroservices-architecture/>
3. <https://www.g2.com/categories/relational-databases>
4. <https://thenewstack.io/6-things-for-developers-to-know-about-postgres/>
5. <https://hackernoon.com/postgresqls-exciting-features-you-should-know-a516a441b8c4>

**Database**

1. <https://www.quora.com/Should-I-use-an-ORM-for-my-large-database-driven-application>
2. <https://medium.com/@mithunsasidharan/should-i-or-should-i-not-use-orm-4c3742a639ce>

**API**

1. <https://blog.stoplight.io/api-first-api-design-first-or-code-first-which-should-you-choose>
2. <https://goa.design/>
3. <https://swagger.io/blog/api-design/design-first-or-code-first-api-development/>

**Cloud**

1. <https://cloud.google.com/blog/products/databases/to-run-or-not-to-run-a-database-on-kubernetes-what-to-consider>

**NGINX**

1. <https://medium.com/@allevo/nginx-for-surviving-in-microservices-era-28fa65295feb>

**Microservices**

1. <https://articles.microservices.com/monolithic-vs-microservices-architecture-5c4848858f59>

**Streaming**

1. <https://stackoverflow.com/questions/14703627/websockets-protocol-vs-http>
2. <https://www.pubnub.com/learn/glossary/what-is-http-streaming/>