Architecture Decision Records

Decisions made during the design process and the reasoning behind them.

Table of Contents

ADR 001: Choose Microservices as Architecture	2
ADR 002: *Choose Azure API Management as API Gateway	
ADR 003: Choose React-JS for Frontend	4
ADR 004: Choose Python for Backend	5
ADR 005: Choose Database per Service as Data Pattern	6
ADR 006: Choose PostgreSQL for Order Management	7
ADR 007: Choose MongoDB for Product Management	8
ADR 008: Choose Solr for Search Engine	9
ADR 009: Choose Render as hosting platform	10
ADR 010: Choose Auth0 as Authentication Provider	11
ADR 011: Choose Cloudflare as Reverse Proxy	12

ADR 001: Choose Microservices as Architecture

• Context:

 My system requires a very scalable architecture since I must be able to scale to 1 million concurrent users.

• Consequences:

- o **Positive**: Efficient Scaling, very modular, very scalable.
- Negative: Complex, high costs.
- Future Considerations: As the number of microservices grows, managing dependencies and interactions between services can become more complex. Over time it may be necessary to implement a service registry and a centralized configuration management system to ensure reliable service discovery and configuration.

• Alternatives Considered:

- Monolithic, Two-Tier, Three-Tier: Rejected because less efficient in scaling horizontally
- o SaaS: Rejected because limited control and lack of experience
- o **Headless**: Rejected because lack of experience

• Links and References:

 IP – Research: There is a dedicated section in this research about the software architecture.

ADR 002: *Choose Azure API Management as API Gateway

Context:

 My system requires a free to use API gateway with built in features such as rate limiting, server discovery, authorization, authentication, auto-scaling, load balancing, low latency, DDoS protection, firewall and most importantly, free to use since I don't have a budget

Consequences:

- Positive: Free to use (100 free credits) and many built in features: rate limiting, server discovery, authorization, authentication, auto-scaling, load balancing, low latency, DDoS protection, firewall.
- Negative: Not as many available regions as AWS.
- Future Considerations: If I run into scalability issues, I might have to consider adding a load balancer in between the API Gateway and the services.

Alternatives Considered:

- Google API Gateway, IBM API Connect, Kong Gateway, MuleSoft
 Anypoint Flex Gateway, Boomi API Management, WSO2 API Manager:
 Rejected because not free to use.
- o **Cloudflare**: Rejected because not as many built in features as Azure.
- AWS: Rejected because a credit card is required to use AWS and I don't have one.
- Azure: I have used Azure, but after spending all my credits I cannot use it anymore.

Links and References:

 IP - What API Gateway should I choose: In this research I compare API Gateways with each other.

*Notes

 I currently don't have an API Gateway because I currently only have one deployed backend and it would be overkill to deploy and configure an API Gateway in this context.

ADR 003: Choose React-JS for Frontend

• Context:

My system requires a well performing frontend.

• Consequences:

- Positive: High performance, high community support. Minor experience with React. Revolutionary live debugging.
- Negative: Complex.
- Future Considerations: Dedicated research on performance on frontends to decide if it's the best choice.

• Alternatives Considered:

- Vue.js: Rejected because I don't have prior experience. I have tried it out briefly hoping it was easier than React but it wasn't.
- Flask: Good Alternative, not as good performing as React but I have experience in flask and python.
- React-TypeScript: I was initially going to choose React-TS, because I had very little experience with typescript. But the community support for React-TS is limited, and since I do have some react experience it would make sense to switch to React-JS since it does have a lot of community support.

• Links and References:

o https://www.simform.com/blog/best-frontend-frameworks/

ADR 004: Choose Python for Backend

• Context:

My system requires a well performing backend.

• Consequences:

- o **Positive**: High community support, I have years of experience in python.
- o **Negative**: Possibly better solutions that I didn't consider.
- Future Considerations: When the project is running and I run into performance issues it might be valuable to research if there are better performing backends.

• Alternatives Considered:

 C#: I gave C# a try, but I feel more comfortable developing an API in python.
 In my opinion python is easier to use, and I find it easier to solve problems/ errors in python

• Links and References:

0 -

ADR 005: Choose Database per Service as Data Pattern

Context:

The system being developed is based on a microservice architecture.
 Database per service is not only a best practice in the microservice architecture, but it also satisfies my quality requirements.

• Consequences:

- Positive: independently scalable, freedom in database choice per service which allows me to choose a fitting db per service, prevent concurrency conflicts
- o **Negative**: Challenging and higher costs.
- Future Considerations: -
- Alternatives Considered:

0 -

Links and References:

 IP - Microservice Research: I briefly go over the pros and cons of db per service.

ADR 006: *Choose PostgreSQL for Order Management

• Context:

 The Order Management service needs its own database. This database has to be ACID compliant and is preferably scalable horizontally.

• Consequences:

- Positive: ACID complaint, providing very strong consistency. Also high performance. Also free to host on Render, the hosting platform which I am using
- Negative: Needs additional setup to scale horizontally.
- Future Considerations: This is a free option, and there are most likely better options that aren't free such as CockroachDB, TiDB and YugabyteDB since these are ACID complaint and made to be scalable horizontally.

• Alternatives Considered:

 CockroachDB, TiDB, YugabyteDB: Rejected because not free, or free trial isn't long enough

Links and References:

 IP - SQL vs NoSQL Research: Research about the difference between relational databases and document-oriented databases, also dive into ACID databases and BASE databases. This document further explains why I choose a database for a service

*Notes

 The PostgreSQL database is currently not deployed. It was being hosted but my free trial is over. I haven't made the corresponding backend yet. So it's not really an issue.

ADR 007: Choose MongoDB for Product Management

• Context:

 The Product Management service needs its own database. It is a best practice to use a document-oriented database for product management

Consequences:

- o **Positive**: Document oriented BASE database. Free to host.
- Negative: -
- o Future Considerations: -
- Alternatives Considered:

0 -

• Links and References:

 IP - SQL vs NoSQL Research: Research about the difference between relational databases and document-oriented databases, also dive into ACID databases and BASE databases. This document further explains why I choose a database for a service

ADR 008: *Choose Solr for Search Engine

• Context:

 Usually a Search Engine service is used it combination with something like Elastic Search. However, Elastic Search isn't free to use. An alternative I managed to find was Solr.

• Consequences:

- o **Positive**: Free to use.
- Negative: Not as much community support as Elastic Search.
- Future Considerations: Consider if Elastic Search is a better choice, and research if it's worth changing to Elastic Search.

• Alternatives Considered:

o **Elastic Search**: Rejected because not free to use.

Links and References:

 IP - SQL vs NoSQL Research: Research about the difference between relational databases and document-oriented databases, also dive into ACID databases and BASE databases. This document further explains why I choose a database for a service

*Notes

o I didn't get far enough to start building a search engine

ADR 009: Choose Render as hosting platform

• Context:

 The system needs a platform wherein I can host containers so that they can communicate with each other independently from my local machine.

• Consequences:

- Positive: Free to use, easy to implement. Additional support for python and postgressql
- Negative: Very limited. Advanced features under which auto scaling are locked behind a credit card paywall.
- Future Considerations: Research if there are better paid platforms (whenever I have a budget).

Alternatives Considered:

- o Heroku: Rejected because Heroku isn't free anymore.
- o AWS, Fly.io, Akamai: Rejected because free tier requires a credit card.
- o Railway: 5\$ worth of credit. Decent alternative.
- o **Azure:** Free 100\$ Credits, Hard learning curve and I wasted all my credits.

• Links and References:

 IP - What API Gateway should I choose: In this research I compare API Gateways with each other.

ADR 010: Choose Auth0 as Authentication Provider

• Context:

o The system needs a way to authenticate.

• Consequences:

- o **Positive**: Free to use. Easy to implement. High community support.
- Negative: I can only use advanced features in the 1-month free trial (under which MFA).
- Future Considerations: Research if there is a better alternative. Maybe Azure AD B2C?

• Alternatives Considered:

 Okta, Azure Entra ID, Azure AD, AWS Cognito: Both are organizational authentication providers which are developed for a different type of use case. I just need a small authentication which provides role based access.

• Links and References:

 GP - Identity Platform Research: This document is the reason why I briefly scanned through the other options but there isn't much documented in it.

ADR 011: Choose Cloudflare as Reverse Proxy

• Context:

The system needs security measures

• Consequences:

- Positive: Free to use. Easy to implement. DDoS protection, Caching, Global server load balancing.
- O Negative: -
- o **Future Considerations**: Research if there are better alternatives.
- Alternatives Considered:

0 -

• Links and References:

o https://developers.cloudflare.com/learning-paths/zero-trust-web-access/concepts/reverse-proxy-server