# Architecture Decision Records

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

**Table of Contents**

# 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**:
  - **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
  - **SaaS**: Rejected because limited control and lack of experience
  - **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.
  - **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.
- **Links and References**:
  - IP - What API Gateway should I choose: In this research I compare API Gateways with each other.

# ADR 003: Choose React-TypeScript for Frontend

- **Context**:
  - My system requires a well performing frontend.
- **Consequences**:
  - **Positive**: High performance, high community support, I already have a little experience with it.
  - **Negative**: Complex.
  - **Future Considerations**: -
- **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.
- **Links and References**:
  - https://www.simform.com/blog/best-frontend-frameworks/

# ADR 004: Choose C# for Backend

- **Context**:
  - My system requires a well performing backend.
- **Consequences**:
  - **Positive**: High community support, I have years of experience in C#.
  - **Negative**: Possibly better solutions that I didn't consider.
  - **Future Considerations**: When the project is running and we run into performance issues it might be valuable to research if there are better performing backends.
- **Alternatives Considered**:
  - -
- **Links and References**:
  - -

# 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
  - **Negative**: Challenging and higher costs.
  - **Future Considerations**: -
- **Alternatives Considered**:
  - -
- **Links and References**:
  - IP - Microservice Research: I briefly go over the pros and cons of db per service.