**Context and Problem Statement**

AML needs a new application with scalability and robustness as their major factors.

The library will provide several services to users, many of which can work independently of each other (Adzic, 2019).

**Decision Drivers**

● System needs to be scalable as it will have at least 11 million users at launch with a 10% annual increase. [1] (Fowler, 2016).

● Different services need to be split between different developers to ensure a smooth development cycle. (Newman, 2021).

● Services shouldn’t need to rely on each other (Adzic, 2019).

**Considered Options**

● Microservices

● Service Orientated

● Monolithic

**Decision Outcome**

The chosen option in this case would be to go with microservices. The decision was made with the scalability factors in mind. With microservices, we would have the option to spool up another instance of a service if it were to get overwhelmed (Fowler, 2016).

**Consequences**

● Good: Provides scalability and resilience (Newman, 2021).

● Bad: Will be more difficult with a more complex development cycle (Fowler, 2016).

**Pros and Cons of the Options**

**Microservices**

● Good: Each microservice can use a different language or framework. This allows external developers to assist with development or allows teams with different skill sets to collaborate (Newman, 2021).

● Good: Microservices are scalable. New features can be added without taking down other services. Can also add new instances of a service to increase capacity (Adzic, 2019).

● Good: Fault resistant, each service is independent so if one instance of a service goes offline, others can handle requests while it’s brought back online. Further from this, an entire cluster of servers can down without taking down other clusters.

● Neutral: Modern, Microservices is a more modern standard that is increasingly being adapted. (Fowler, 2016).

● Bad: Microservices need to send requests between each other, which could introduce latency issues (Newman, 2021).

● Bad: Microservices are difficult to develop. Solid design documents are needed to ensure the services can communicate with each other effectively.

● Bad: While each service can be written in a different language, this can cause the project to take on too much technical debt, making it hard to maintain (Fowler, 2016).

**Service Orientated**

● Good: Scalable, New features can be added easier. Can make changes to accommodate increased usage (Newman, 2021).

● Good: Each service is contained separately, providing strong resistance to faults.

● Neutral: Industry standard widely used architecture so is familiar across industry. (Adzic, 2019).

● Bad, because the Enterprise BUS connects all services, provided a single point of failure [2]

● Bad: All services use the same database which makes it difficult to manage each individual component and acts as a single point of failure.

**Monolithic**

· Good: Monolithic architecture is a lot simpler and easier to develop as it doesn’t require planning for multiple de-coupled services (Fowler, 2016).

· Good: Everything is running under the same program, eliminating additional latency from sending requests to other services (Newman, 2021).

· Bad: Low fault tolerance. Failure of a single system can have a cascading effect, taking out the entire application [2] (Adzic, 2019).

Appendices

[1] Project overview

References

[1] https://aws.amazon.com/compare/the-difference-between-soa-microservices/

References

- Adzic, G. (2019). \*Running Reliable Microservices\*. LeanPub.

- Fowler, M. (2016). \*Microservices: A Definition of This New Architectural Term\*.<https://martinfowler.com/articles/microservices.html>

- Newman, S. (2021). \*Building Microservices: Designing Fine-Grained Systems\* (2nd ed.). O’Reilly Media.

- [1] https://aws.amazon.com/compare/the-difference-between-soa-microservices/