**ARCHITECTING SCHOOL MANAGEMENT SYSTEM USING MICROSERVICES ARCHITECTURE**

Microservice architecture is an architectural style that structures an application as a collection of services. Microservice architecture ensures continuous delivery and or deployment of large complex applications. It also ensures scalability and fault isolation and tolerance.

In considering micro service architecture, analyst must identify and design services and find out how the work together or collaborate.

Each service has an API that consists of operations.

Operations consist of two types; commands and queries.

A command is an operation that mutates data. Eg: in the school management App, commands in the student data service might implement commands like addStudent().

A query is a command that retrieves data. Eg: findStaffStatus() in the staff data service.

## A service might collaborate with other services

## Services collaborate through API rather than through database. A service might invoke the operation of another service example: students data service can invoke the financial data service

## A service’s database is private

A service has a database, DB1, which stores its data and sometimes data replicated from other services. In order to ensure loosely coupled services, services ought to communicate through their APIs and not their database tables.

**Breaking the school management application into small loosely coupled services.**

Teaching /non-teaching staff service

School Management App

Parents’ access and information service

Students’ information service

Financial information service

Student data Service

A

P

I

G

A

T

E

W

A

Y

Staff data service

Client request

Parent info and access service

Financial information service

Considering four functionalities; students’ data service, staff data service, parents’ access service and financial service. Each service has its own database, DB1 and DB2, where DB2 is serving as a redundant database for each service.

Each of the service is deployed separately and can be accessed through an API gateway.

An API gateway serves as an entry point for the application such that it takes the user request and determines which micro service the user wants to access and gives the user the response.

Each micro service has its own database and thus it can work independently.

With microservice architecture, there’s flexibility and agility as users can still access other services while another service is been worked on or updated. Example, assuming there is an upgrade of financial data service, users can still access other services while financial data is been updated.

This flexibility in micro services helps in fault isolation as other services can still function properly and independently when there is a fault in service without the other services to being down.

In architecting the school management app using microservice architecture, there can be a mixed technology stack. That is, different languages and algorithm can be used to implement each service. Considering the complexity of a school management app, you could have different developers working on each service in languages they are proficient in without having to worry about it not working as a result of mixed technology. Example; a developer working on a financial information service could work in node js whiles the other working on staff data can work with another language and they all will work perfectly fine.

While architecting school management app using microservice architecture, there can be granular scaling. Example, as Students are always been admitted into schools, there would be a high demand of resources in the students data service whilst the others may not be demanded that much. In such cases in microservice architecture, the students’ service can be scaled based on the demand without having to scale the whole application or the other micro services.

**MICROSERVICE TOPOLOGY**

There are mainly three topologies in implementing microservice architecture. The API REST based topology, application REST-based topology, and the centralized messaging topology. In architecting the the School management App, the API REST based topology was considered.

This topology, consists of very fine-grained service components that contain one or two modules that perform specific business functions, independent from the rest of the services. In this topology, these service components are typically accessed using a REST-based interface implemented through a separately deployed web-based API layer.

**CACHING**

In order to make the school management app faster, we introduce caching into our system. That is with caching, there is reduced number of trips a microservice will make to a database upon user request. This tends to improve availability as users can still get data from cache when system is down.

When service is requested for the first time, the cache is populated for the first time. Then all subsequent requests for the same data is served from the cache.

While Caching is supposed to make the system faster, it has got its own drawbacks. Amongst them are;

* Update latency: In the School Management App, let’s say, students data was requested and replied. Its data will be stored in the cache. Suppose there’s an update on the student data and the cache is not validated, it will give the user old values. Thus the cache will need to be configured the validation of the cache out of date.
* Another thing about caching is knowing how to manage your cache. Considering which whay and how much data to be cached. Nevertheless, there certain microservice strategies to avaoid overloading a cache to avoid stale and maintain high microservices performance.

MANAGING CACHE

What to cache?

There are two types of caches; lazy loaded cache and preloaded caches. In preloaded caches, data is populated ahead of the start of a service and is ready before a service requests it In Lazy loaded cache, data is kept on hold until a service request it.

Considering the school management system, services that might need to preload frequently used items in the cache so that a new order is processed using an item residing in the cache. However, applications that don't need to access data this way can be best be left to lazy-loaded caching so as to avoid data overloads.

How long to keep data

When data changes, there is a need to make sure that data stored in cache is always in sync with current data. Since cache data can go stale for a while, it is important to decide the duration to which data is stored in cache. When data is expired, a request to it will fail. As long as there failed request don’t outnumber the requests that are successful.