Guiding Principles

The Problem

Obtaining a comprehensive picture of users' financial health is oftentimes difficult when they have accounts at multiple financial institutions.

The Solution

Help users improve their financial health with categorised transactions

So a couple of things to get us to par:

Conway's law- *A system's structure and design will most likely mirror the communication structure of the organisation that designed and built it.*

This has always been the guiding principle in every team I work with. To me, this principle is so important because I believe that if the communication between members is fractured, the result always appears as a cobbled-together solution. After all, everyone would want to leave their mark on the final product.

Hence,

It is always good to develop from the ground up, build robust communication structures first, and then begin the design process. This way, you'll get a more streamlined outcome that feels like a team effort rather than a fragmented solution.

Or what they call inverse Conway manoeuvre, which in my translation, I usually put it this way:

Build the organisation you want, and then your design and architecture will follow screaming and kicking.

Now with that out of the way, let's go back and refreshen our knowledge on microservices.

What are microservices?

Small ***Autonomous*** *services that* ***work together****, modelled around a* ***business domain***

Based on the above definition. I have always found that the following principles will enable you to get the most out of any microservice system.

- **Modelling your design around the business domain**- This gives us more stable APIs
- **Factoring the culture of automation**- This comes in very handy when we have a lot of deployable units
- **Hiding implementation details**- This allows one service to evolve independently of another.
- **Decentralizing everything**- This applies to both the decision power and architectural design concepts
- **Deploying independently**- One of the most important markers, if not the most important. The idea that one can make a change to a service and deploy it into production without making changes to anything else.
- **Consumer centred –** services are meant to be called
- **One should be able to isolate failure-** Microservice systems are not meant to be flaky like their monolithic counterparts. *This is the reason behind my obsession with load balancers* ***@Job***
- **Systems should be highly observable**- Anyone, even one with no technical background should be able to understand how the system works. Building in things like correlation id, and aggregating stuff in a consistent and standard way is very important. *This Will come in handy for some of our stakeholders who are not technical but their input on the feature set we intend the product to have is highly valuable.*

So how do we apply the above set of principles to our own design? Well, for starters since we are just starting, looking into getting the design of the product right and eventually scaling it. The following in my view should be our guiding principles:

Strategic goals	Architectural Principles	Design and delivery practises
Enable Scalable business • *More customer transactions* • *Self-service for customers- This is the reason behind my right-heavy system argument @silvester*	**Reduce Inertia** *The design concept should favour rapid change and reduce dependencies across the services*	**Standard REST/HTTP**
	Eliminate accidental complexities *We should be able to aggressively retire and replace unnecessary complex processes and integrations. This in turn enables us to focus on the essential complexities*	**Encapsulation of legacy environments** – *the success of the system is predicated on integration with legacy systems*
Support entry into new markets • *Flexible Operation Process* • *New Product and operational process* *@Job this is the reason I asked you to check on how we can achieve multi-currency support*	**Consistent Interfaces and data flows** *Our Design should eliminate the duplication of data and create a clear system of records with consistent integration interfaces.* *@Silvester this is the reason I was bringing up the argument that we should have a single system of record for the data that will be owned by each service*	**Consolidation and cleansing of data**
Support innovation in existing markets • *Flexible Operation Process* • *New Product and operational process*	**No silver bullets** *Most off-the-shelf solutions normally deliver early value but create inertia and accidental complexities later on* *@Job your design captured this. You were looking at the problem scope from each use case scenario which is a good thing. The key is starting simple and then scaling it as you progress*	**Having a clear published integration model** **Creating small independent services** **Continuous deployment** **Minimal customization of COTS/SAAS- *if we decide to go this route*** **Elimination of integration databases**