#ASLI ENGINEERING

Backend for Frontend Pattern in Microservices

SWIPE

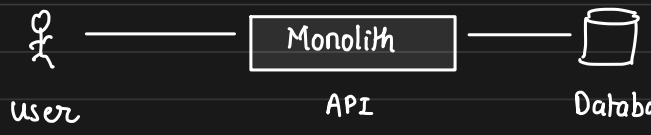
BY

ARPIT BHAYANI

Backend For Frontend

Say, you have a monolith that serves all the APIs and your users interact using Desktop Browser only

So, a typical 3-tier architecture would look like



Say, your app is an e-commerce app and to send product details

GET /v1/products/1729316

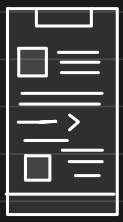
```
{
  name: "_____",
  description: "_____",
  sellers: [ {      }, { }, ... ],
  variants: [ ..... ]
  faqs: [      ],
  reviews: [
    ≡≡≡
]
```

In a single API, you are sending out a huge response that would help you render the complete page

- reduced n/w overhead
- easy to cache

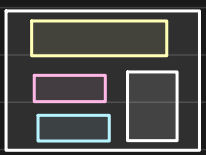
Foray into Mobile

Say, now you attained PMF, you thought of launching a mobile app. So, now your same monolith API will also power the mobile app.



Note: mobile has a limited real estate

So, you will not be rendering everything that you rendered on web (desktop)



More details shown in desktop browser

Only essential details are shown

- only critical components are shown
- others are loaded when explicitly requested

eg: Review shown when user clks and visits Reviews

Why should we, unnecessarily send data from backend to mobile?

- we save n/w bandwidth
- we save processing and memory
- reduce load / render time. good ux

Not just mobile

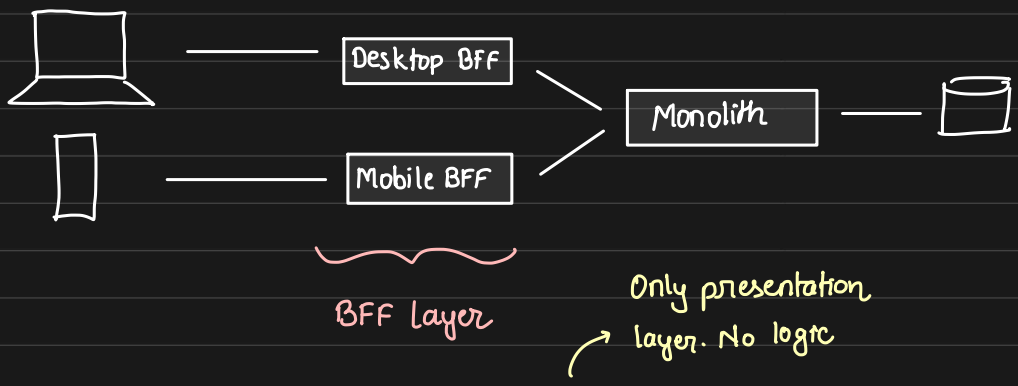
It is not just about supporting mobile, think about 3rd party services like Alexa, other customers, etc

You may want to hide some details from one type of client or show some extra info on some

How can we do this elegantly?

Backend For Frontend Pattern

The idea is to have an intermediate service that is dedicated for a type of client, for example



BFF are client specific and sits b/w backend and clients. They are similar to API Gateways

Instead of having a single point of entry,

BFF provides a separate entry for each type of client

- each BFF decides
  - ↳ what it needs to fetch
  - ↳ how it needs to fetch
  - ↳ what needs to be send in response

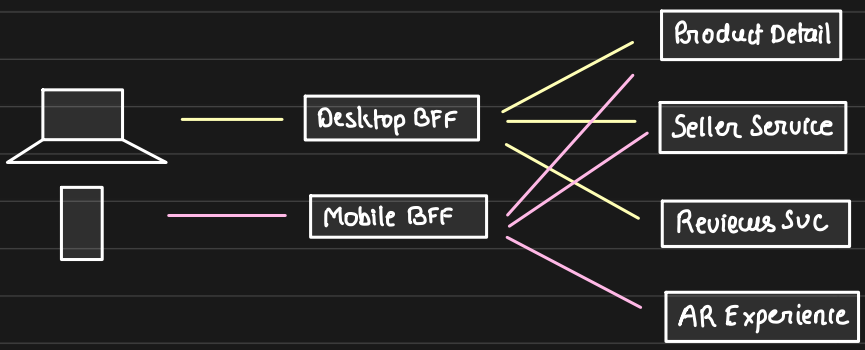
A mobile BFF can remove so much of bloat and junk

while a desktop BFF can fetch additional info and send

BFF and Microservices

BFF pattern fits perfectly with Microservices where

BFF just like API Gateway picks the downstream services to fetch data



Advantages

- Support for multiple isolated interfaces
- client specific tweaks are much faster
- hide sensitive information seamlessly
- picking right stack / protocols for client

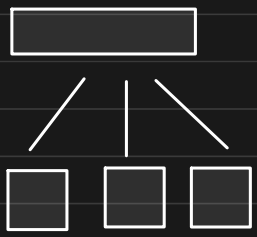
pick best protocol
eg. JSON, XML, etc

pick best protocol
eg: raw TCP / gRPC / REST

pick best network
eg: TCP / UDP / WS / gRPC / HTTP

- Improved security: apply client specific security at BFF (attacks)
- You need a single general purpose backend and BFF can take care of customizations
- BFF can also act as a request aggregator

Disadvantages



- FanOut: BFF services has to do a large fanout (to other services).
  So, BFF has to be designed to be network heavy & stack should support that
  eg: Python may not be the best choice for implementing BFF

- Code Duplication
  Most of the code across BFF would be very similar, given they would be interacting with same set services. This unnecessarily multiplies the dev efforts.

- More moving parts
  By adding a new BFF layer, we introduce new moving parts that need to be managed, maintain, and monitor, and deploy

- Slight increase in latency

When should you introduce BFF?

↳ When the interface are significantly different across diff clients then it is worth to have

- single backend
- multiple BFFs

to support multiple clients

↳ When the communication format of a client is very different

eg: one of the client only understands XML → Legacy integration while others understand JSON

So, we can