# API Class Notes

#### 2022-06-21

API are interfaces plus a protocol

- RPC - CORBA – JAVARPC

- SOAP

- Restful

- GraphQL

Principles

- Open close

- Imperative versus declarative

## The Speech Contract

APIs to a certain extent are intuitive

## Ancient APIs – Remote Procedure Calls

1960s – Programming libraries – APIs looked like function calls

* Produced a mindset that APIs HAD to look function calls
* Imperative interface – each item in an interface is a command

1980s – distributed systems

* Corba – RPC Remote Procedure Calls – C++ type apps

1990s – Java distributed systems Java-RPC RPC-IIOP

End of this approach – rise of web based apps.

## SOAP – Simple Object Protocol

* Still thinking imperatively
* RPC but over HTTP
* SOA – let’s connect all computing via a networking layer
* https://www.cs.usfca.edu/~parrt/course/601/lectures/programming.by.contract.html
* UUDI – service discovery
* WSDL – “wizzle” Web Service Definition Language” – XML
* SOAP – Simple object access protocl
* https://stoplight.io/api-types/soap-api
* Archaic but it is still around.
* XML as the primary tool – very difficult to work
* Tooling was bad – hard to work with
* Lots of boiler plate
* RPC – imperative programming
* Aspect Oriented Programming
* Web service over HTTP
* No clear way to implement SOA design principles – talking at the service level – SOAP is code and the transition is hard.

## REST

* The explosion of the web
* Push applications out to end users.
  + They can’t be trained on how to use the applications
  + Apps had to be intuitive- Iconic
  + People do not think in terms of RPC, they think in terms of doing things to domain objects
* APIs -> Service interfaces
* Declarative – specify what we want to happen versus explaining how to something
* Increasing complexity means larger system
* System failures due to complexity
* https://restfulapi.net/

## Engineering

* Coupling – loosely coupled/ modular /
  + Use interfaces – use service interface
  + Decomposition of a service into microservices
  + How do they communicate? Through interfaces
* Cohesion
* Suppleness – Interfaces should not break
* https://www.ics.uci.edu/~fielding/pubs/dissertation/rest\_arch\_style.htm

npm install --location=global json-server

git clone https://github.com/ExgnosisClasses/JSON.git

cd into directory with db.json

json-server db.json

### Lab Exercise

Doctors office

Appointment app

- enable patients to book appointments on line

- track the appointments – on time or delayed

- modify or cancel on either side

- office to be able to confirm appts online etc.

1. How many clients/interfaces should there be and why?
2. What are the resources we are working with
3. what operations should our rest interfaces have
4. Extensibility – open to additional functionality.

Notes:

1. 3-- doctors, patients, admin

insurance

calendar – different calendars for different interfaces

transactions -

3. get available appointments (takes params like days/time), set appointment for the patient, get patient's scheduled appointments

## Problems with Rest

1. Over reporting: /appointment -> ALL the appointment (10,000 all returned)
2. Under reporting: we need execute multiple calls to get the data we
   1. *appointment/{id}*
   2. *patient*
   3. *can’t follow relationships in the data*
3. Processing is done by the client
   1. Client has to construct the data
   2. Client has to parse response

## GraphQL:

1. Defines independently of the actual internal representation the resource types through a schema.

1. Uses a single endpoint:

1. Process queries – returns a result
2. Process mutations (any that changes a resource)

2. Validate queries and mutations against the schema – strong typing on resources

3. Providing resolvers that do the actual on a field by field basis

4. Can deal with relationships between resources.

5. Needs of clients to over and under fetching/reported

6. Iconic if the schema are designed correctly.

7, removes dependency on HTTP

types

Doctors, Appointment, Patient etc...

# Api Best Practices

https://www.moesif.com/blog/api-guide/api-design-guidelines/

# Spring and MicroServices

Do as little work as possible.

https://12factor.net/