# 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/>

Predicates for doctor, patient, appointment

doctor??? -- medical practioner “a staff member who is authorized to see patients”

appointment – a specific date and time when one more patients are slated to see one or more medicals

purpose

date

links to practioners

location or virtual

Patient: “ Any individual who is registered as a patient at our clinic”

- purpose is an attribute - range of possible droplist of typic appointment but with an option for other.

In java this would be an arraylist of values predefined as a enum;

Add some more entities -

(Java – suggest inheritance of some)

Basic relations and what predicate for each

----

Core attributes that we have for a patient

unique idenitifier

What attributes are associated with a patient

medical information

#### contact info

insurance

<https://github.com/spring-projects/spring-data-book>

## Data Tech Generations

#### Gen 1: Hard coded executing SQL directly

1. Moving data in and out of existing RDBMS
2. Vendor dependent

#### Gen 2: JDBC

1. Portability across RDBMS

#### Gen 3: Interfaces

1. Spring data manages the connectivity
2. Different sorts of database

#### Gen 4: Data service

1. Web services fulfill data
2. Infrastructure from the code (microservice)

1-4 ETL – extract, transform, load in repositor

#### Gen 5: Big Data

1. Data lake – honking mess of database
2. extract – load -transforn
3. Hive, Spark, HFSD Kafka

# Spring and MicroServices

Do as little work as possible.

<https://12factor.net/>

<https://spring.io/guides/gs/rest-service/>

<https://www.eclipse.org/community/eclipse_newsletter/2018/february/springboot.php>

<https://oauth.net/2/>

## ORM and the Data Problem

ORM – Object Relational Mapping

real world data is fuzzy

black/dark and white/light

red

green/yellow

blue

dark blue – navy indigo

-------------

-

## Testing and TDD

0 – 0 = 0

1. LP – SP 5 – 4 = 1

2 SP – LP 2 – 3 = -1

3. LNN - SNN -5 - -2 = -3

## Matchers

Gen 1: Part of Java – debugging tool.

Gen 2: Junit

Gen 3: Hamcrest

# DevOps

June 27

https://medium.com/bb-tutorials-and-thoughts/understanding-docker-volumes-with-an-example-d898cb5e40d7

https://minikube.sigs.k8s.io/docs/

---

1. Creating a docker – pull from a repo

2. Building from docker file

3. Running a docker

----

use minikube to create k8s cluster

configure

deploy a pod

https://kubernetes.io/docs/tutorials/hello-minikube/

Lab for 15 more minutes until 3:45

# June 28

On the fly lab

1. Install Jenkins (if necessary)

Downloading and running a war file

2. Create a hello world project

What Jenkins does.

* Link together the stages in development pipeline
* Pipeline is an automated development process
* Coordinates the work of others like repos, build tools, deployment
* Ensure that as one tool finishes (event) the next tool can pick and start workin
* All the tools we use were generally in isolation.
  + Git – to do scm
  + Maven – to do Java build
  + CMAle – C++ builds
  + Sonarquve
  + Autotest – Cucumber

Steps for the lab

1. Install the Maven Integration plugin

2. Configure global tools to specify where Mavens is

3. Create a Maven project

https://github.com/ExgnosisClasses/MavenReferenceProject.git

Add goals and options “clean install”

resume at 2:35

Steps for the lab lab until 3:30

1. Create a pipeline
2. Copy and paste the pipeline
3. https://github.com/ExgnosisClasses/JenkinsFile.git
4. Run it and experiement
5. Delete the script and run the pipe right from the repo
   1. main branch
   2. JenkinFile

Tomorrow is cloud computing

into Thursday

AWS to set up a jenkins cluster

Automatic code quality Sonarqube

multi branch builds