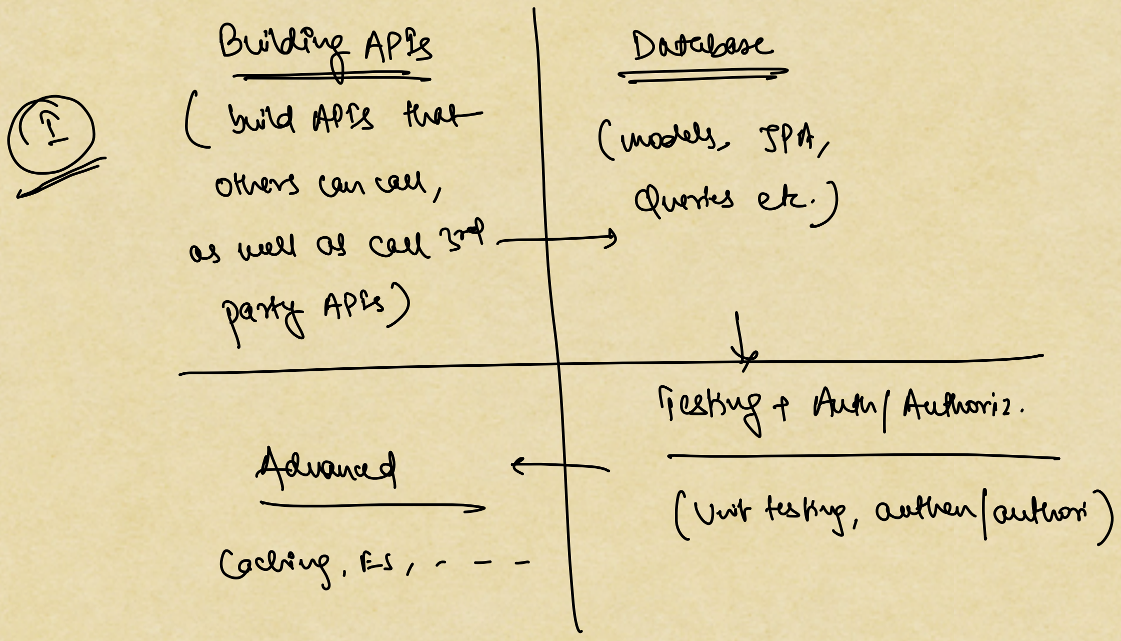

I
Building APIs
(build APIs that others can call, as well as call 3rd party APIs)
Database
(models, JPA, Queries etc.)
Testing + Authn/ Authoriz.
(Unit testing, authen/author)
Advanced
Caching, ES, - - -

Agenda

✓ i) What is an API ?

ii) What is REST ?

iii) Good practices for REST

iv) LLD of Product Service

v) MVC Pattern

vi) How a request is served in Spring ?

⇒ What is an API ?

API ⇒ $\underbrace{\text{Application Programming Interface}}_{\text{interface for an appn}}$

```
interface Runnable {
        void run();
}
```

Any class that implements Runnable has to implement the run method.

↓

sort of like a contract b/w the interface and the class implementing it.

API ⇒ contract, definition

↓

if anyone wants to connect/comm. to our service it has follow the API contract

API ⇒ major 4 things

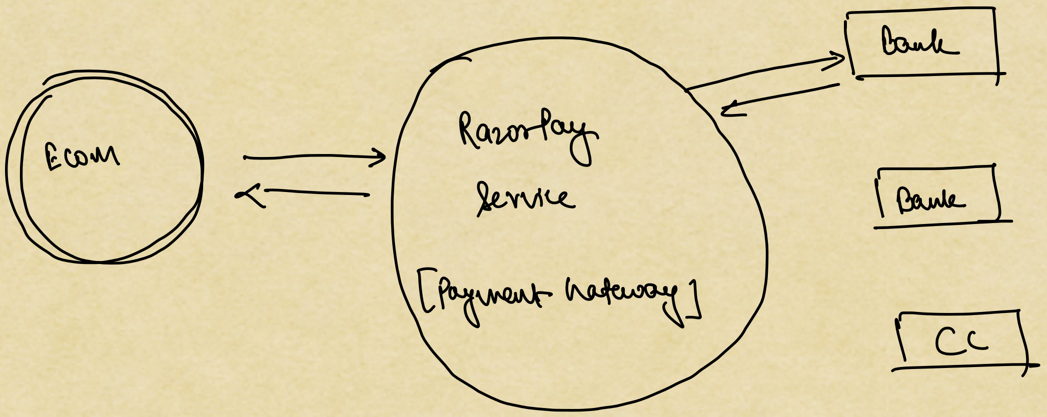
↓

- HTTP ⇒ PUT, POST, GET, . . . _
- endpoint url ⇒ ____________
- i/p * request ⇒ ____________
- o/p * response → ____________

H.W
Read about HTTP methods

⇒ Razorpay

↳ Create order



→ Whenever 2 apps are talking to each other, they must have a well defined contract :

→ where & How

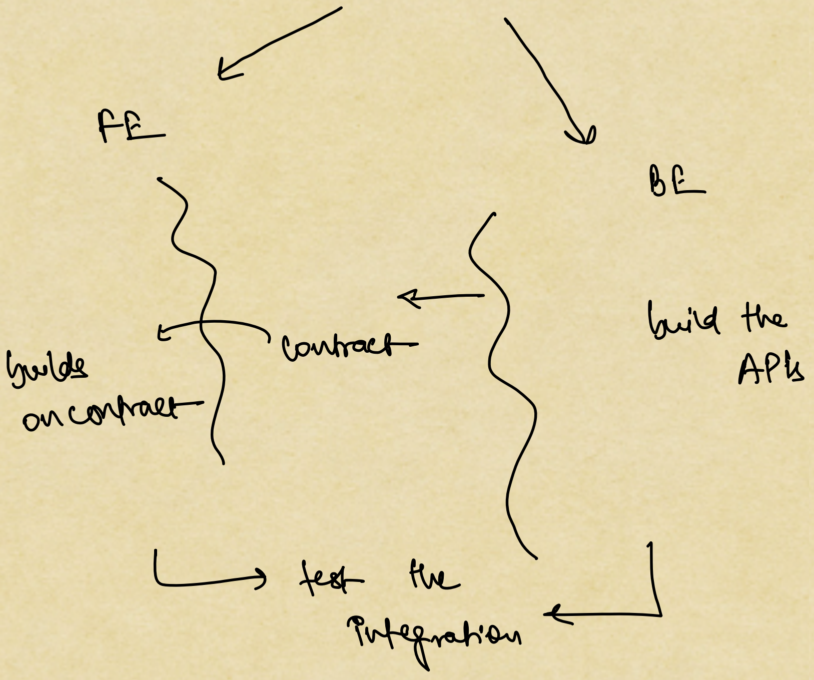
→ what to send

→ what will be recieve

API ⇒ Set of methods | endpoints | functionalities that are provided by system for us to use.

⇒ Product Manager :

Defines what is required to be built



11) what is REST ?

REST => Representational State Transfer

↓

{Stateless}

↓

it doesn't remember anything from previous calls

↓

every call feels like a new/unique call to server

REST uses HTTP

APIs → REST APIs

⇒ <u>Best practices for REST APIs</u> :-

1) <u>Use Descriptive Naming</u> :-

API for order details for a particular user

bad ⇒ $\longleftrightarrow$ /get-orders

good ⇒ $\longleftrightarrow$ /users/{userId}/orders

2) <u>Name around resources & methods should define action</u>

<u>User</u>

~~Create~~	~~/createUser~~
~~Read~~	~~/getUser~~
~~Delete~~	~~/deleteUser~~
~~Update~~	~~/updateUser~~

User

Create	/users → POST
Read	/users → GET
Delete	/users → DELETE
Update	/users → PUT

3) Include proper HTTP codes

4) Maintain versions

GET ⇒ /users ⇒ gives list of all users

↓

/v1/users → ✗

/v2/users ⇒

REST has no mandate over the exchange format

✓JSON / ✓XML / ✓ProtoBufs

⇒ LLD of Product Service :-

Class diagram

Product
- id
- title
- price
- description
- imageURL
- Category

Category
- id
- name
- description
-

$$\begin{tikzcd} 1 \arrow[r, bend left] & 1 \\ \text{Product} & \text{Category} \\ M & 1 \arrow[l, bend left] \end{tikzcd}$$

P : C ⇒ M : 1
C : P ⇒ 1 : M

Schema Design

Product

id	title	price	description	imageURL	cat_id

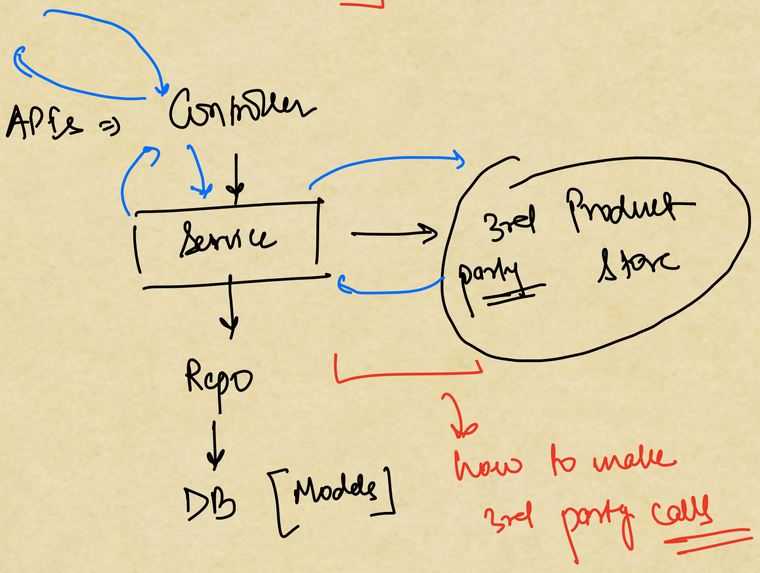
Category

id	title	description

GET => /products/{productId}

GET => /products

build our own APIs



DB as a Services => DAAS

DBAAS

Platform Service

↓

DB

C

R

U

D

APIs