Agenda.

→ REST practices.

→ Calling 3rd party API's.

REST practices.

↳ Good practices to be followed to write Web API's.

⇒ How the API's should be named?

/videos/upload

/users/create

/users/update

/users/delete

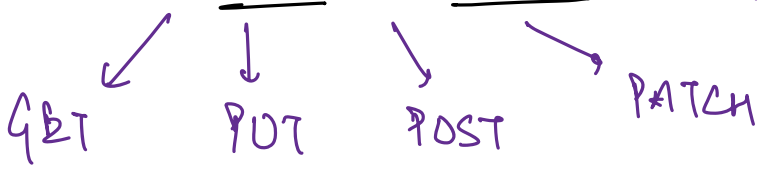
} ×

⇒ All the API's should be structured around the resources that they are trying work upon. (entities)

⇒ The type of the action that API is performing shouldn't be a part of API endpoint.

=> Every API is nothing but an action to be performed on one of the Models.

=> The type of the action shouldn't be the part of API endpoint rather it should be defined by the HTTP method. → DELETE.



GET : Fetch a resource from the server.

```
@GETMapping()
fetchUser(—) {
    ——
    ——
    ——
    ——
}
```

POST : Creating a resource.

DELETE : Deleting something

PUT : Complete Update / Replace.

PATCH : Partial Update

```
{
    "id" : "1",
    "name" : "Vishal",
    "Company" : "Amazon",
    "YOE" : "5"
}
```

PUT : Replace the existing object

PATCH : Update one or more params.

Partial Update.

PUT.

```
{
    "id" : "1",
    "name" : "Teja"
    "Company" : "Scaler"
    "YOE" : "4"
}
```

⇒ Think of every resource as a folder.

(entity)

/users
- GET
- POST
- PUT
- PATCH
- DELETE.

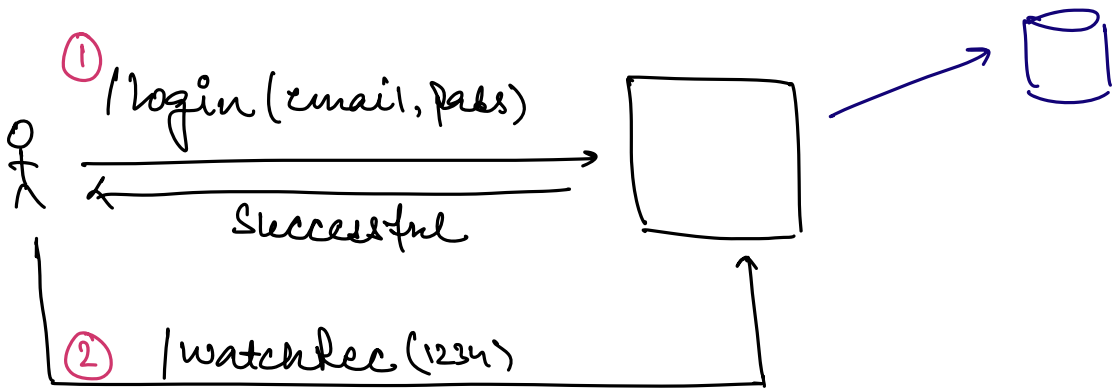
- GET
- POST
- PUT
- =
- =
- =

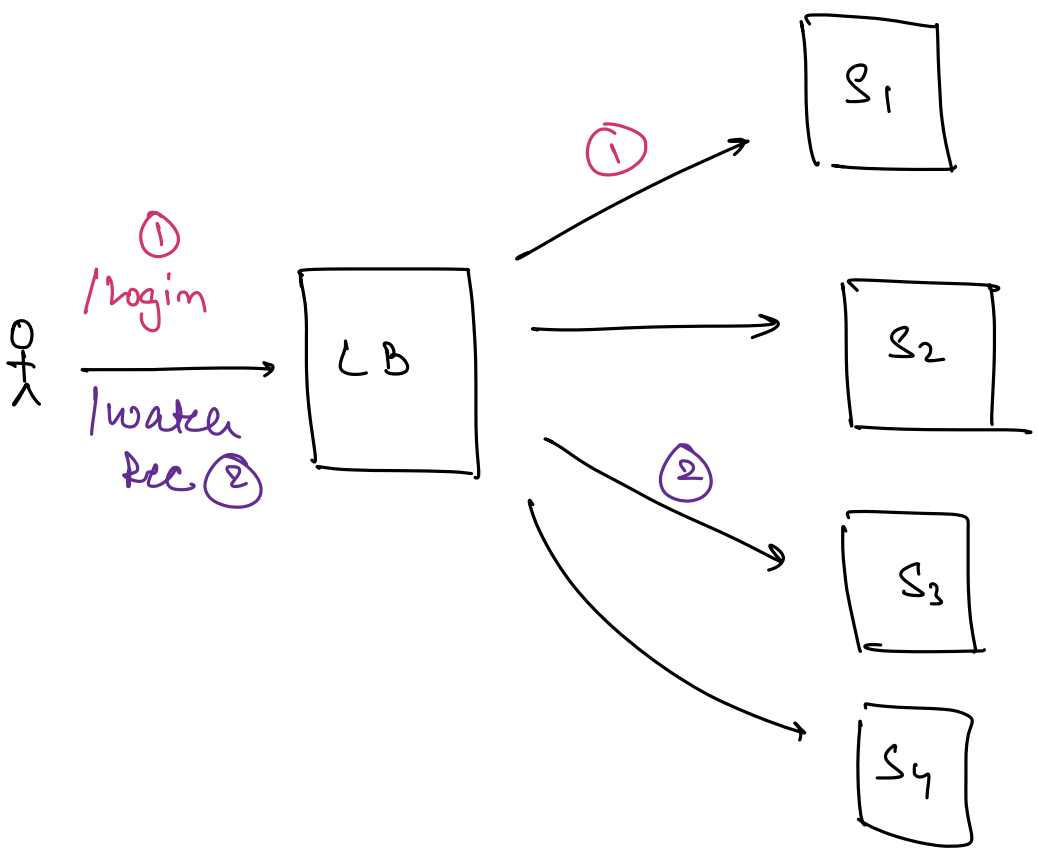
REST APIs should be **Stateless**.

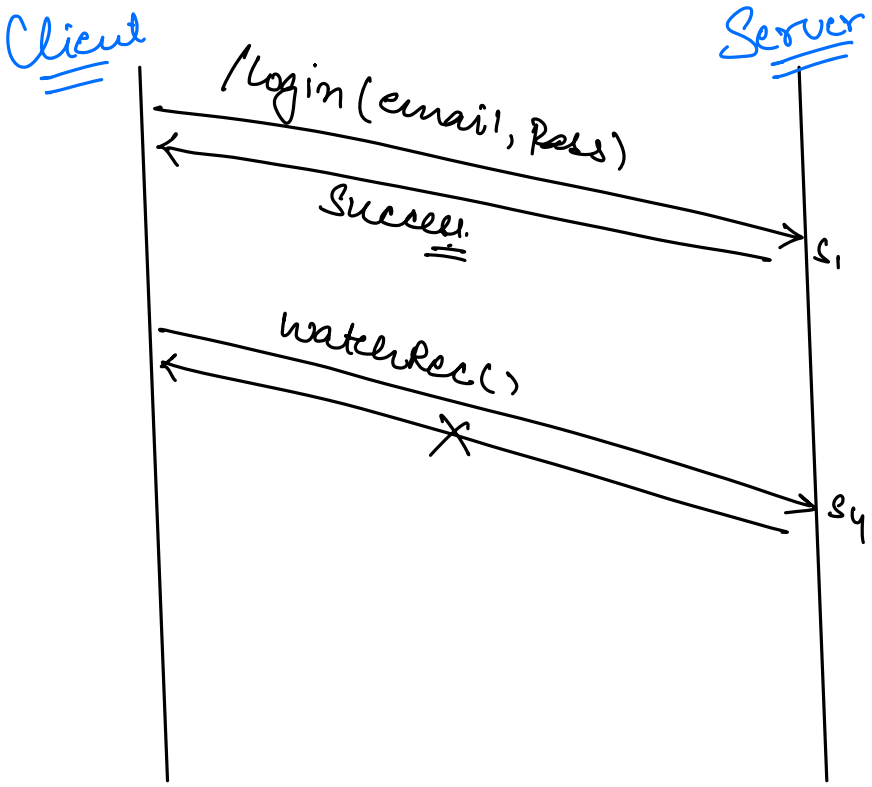
⇒ Every request should be **independent** of the previous request.

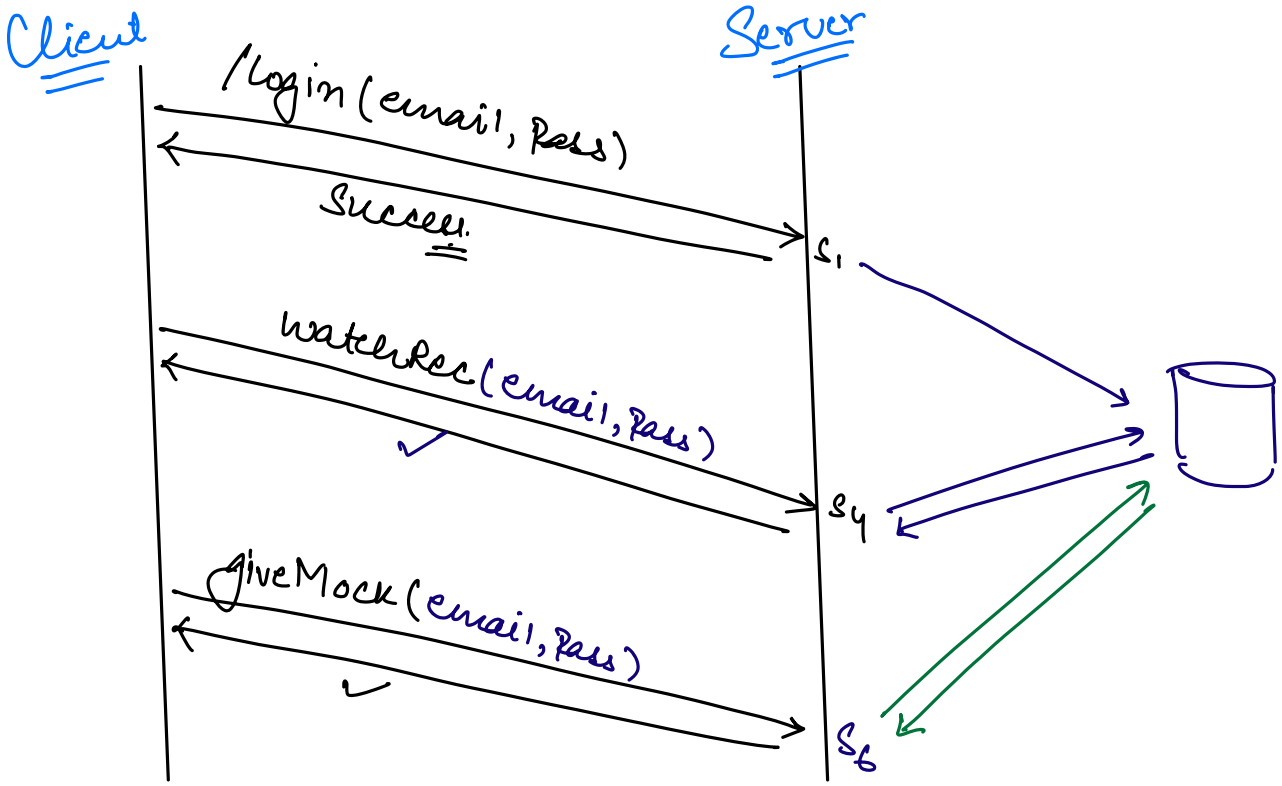
⇒ Each request should be self sufficient.

Distributed System.

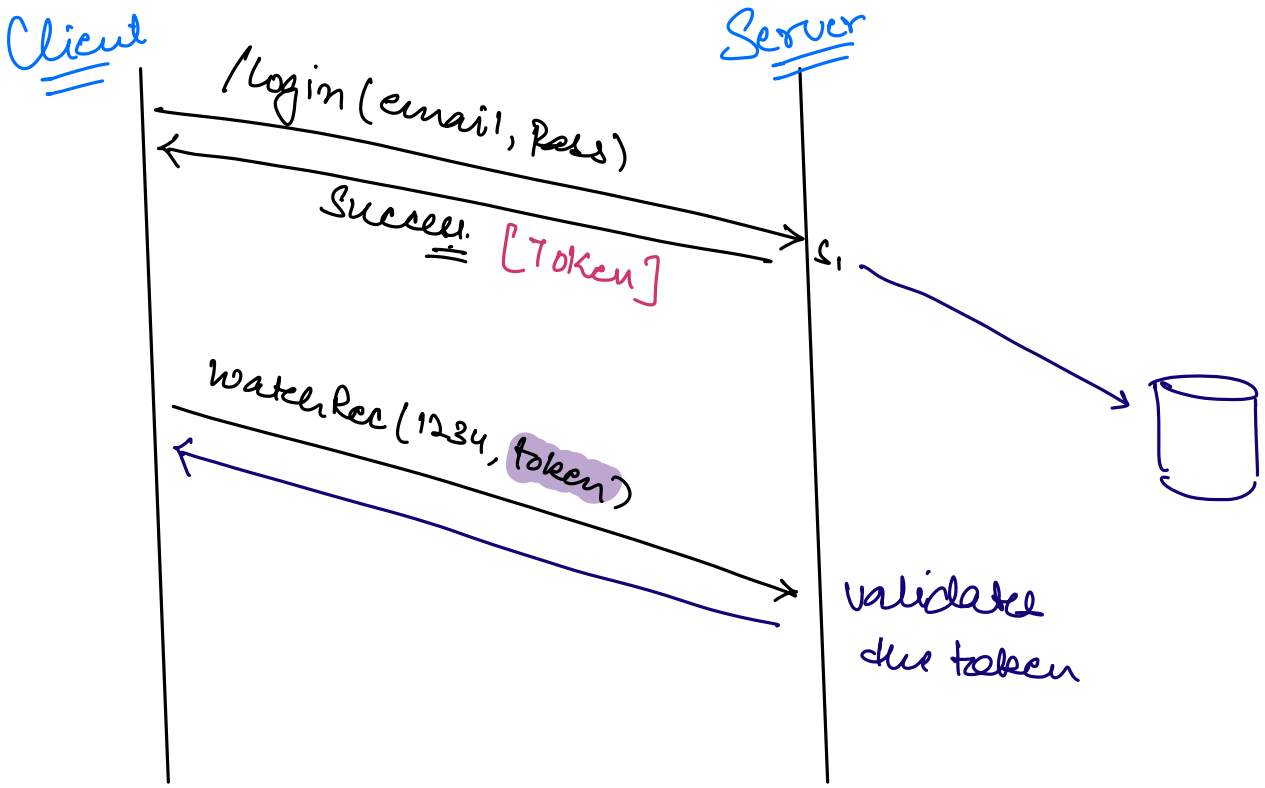





Client
Server
/login (email, Pass)
Success.
s_1
waterRec()
s_4


Client
Server
/login (email, Pass)
Success.
s_1
waterRec (email, Pass)
s_4
giveMock (email, Pass)
s_6

⇒ Every request should contain all the params which are necessary to execute a particular request.



#

mentors

id	Company	sessions#	avgRating	user-id
10	Amazon	50	4.8	1

users

id	name	email	phone
1	Deepak	—	—

/GET

/mentors/10

{

"mentorId" : "10",

"Company" : "Amazon",

"avgRating" : "4.8"

"Sessions" : "50"

"userId" : "1"

}

/GET

/users/1

{ —

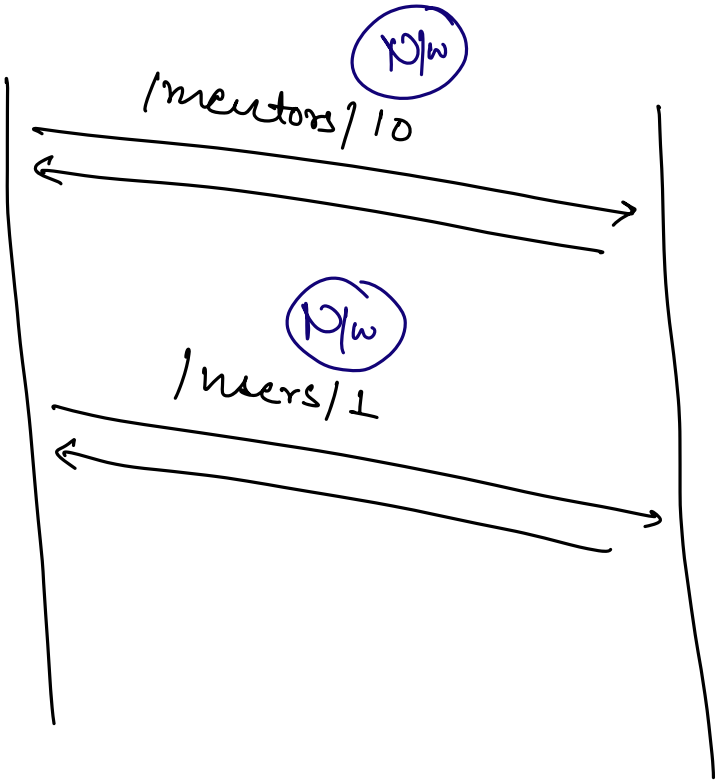
—

—

—

—

}



Chatty API's.

⇒ Not returning all the relevant data in one go.

⇒ Client needs to make multiple API calls to get the complete data.

⇒ Try to NOT have chatty API's in the system.

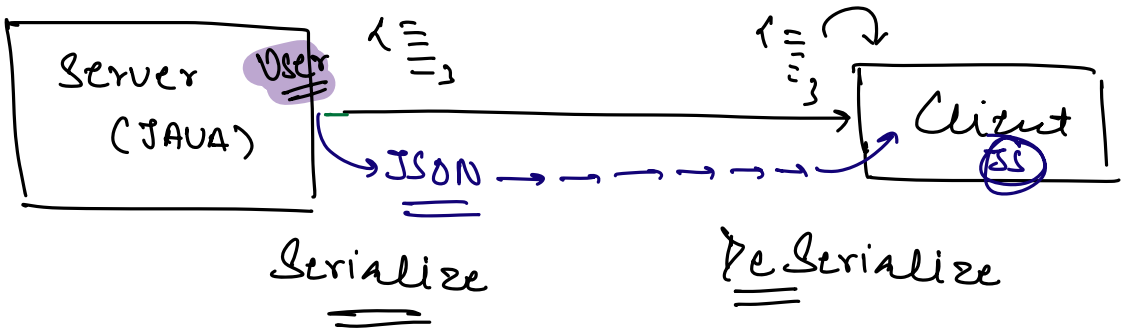
Response Type

⇒ REST don't have any restriction on the type of response.

JSON / XML / protobuf.

```
{
  "mentorId" : "10",
  "Company" : "Amazon",
  "avgRating" : "4.8"
  "Sessions" : "50"
  "userId" : "1"
}
```

Most widely used



⇒ Protobuf

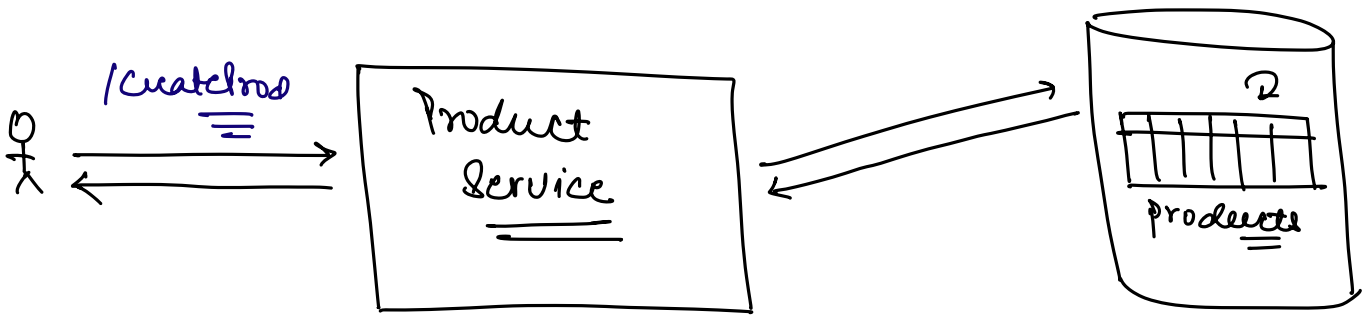
Pb

Binary form.

⇒ Product Service.

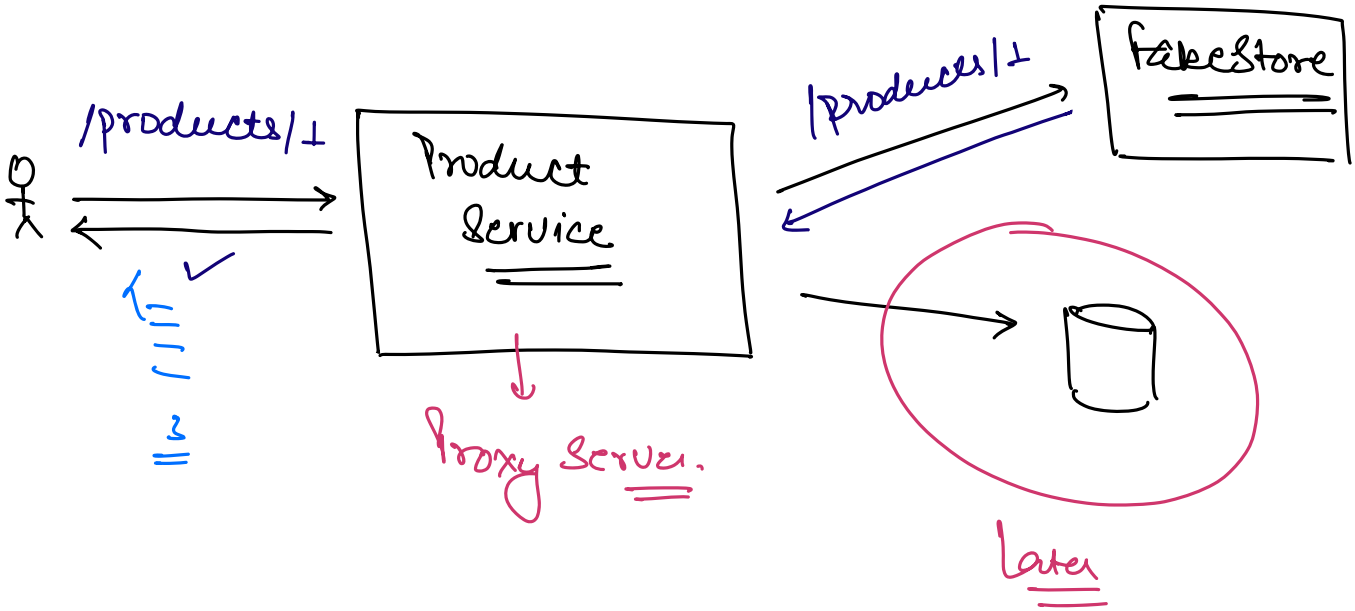
- createProduct
- getProduct
- updateProduct

⇒ DB



Implement ProductService which uses a 3rd party API behind the scenes

⇒ How to use 3rd party API's.



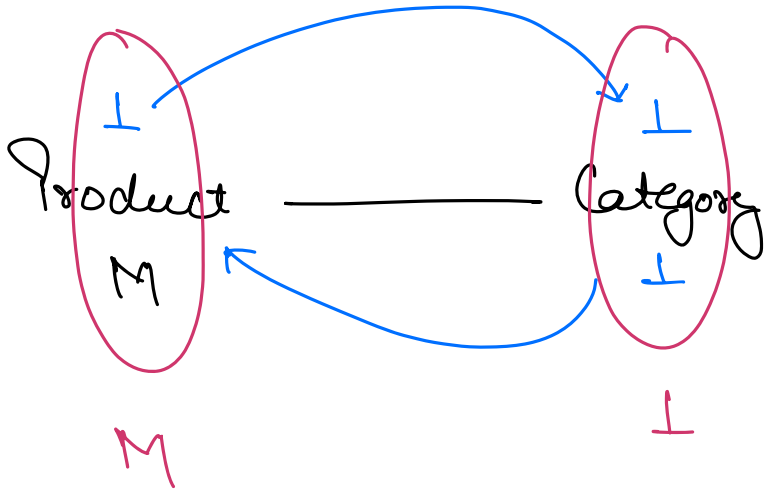
Design.

Requirements

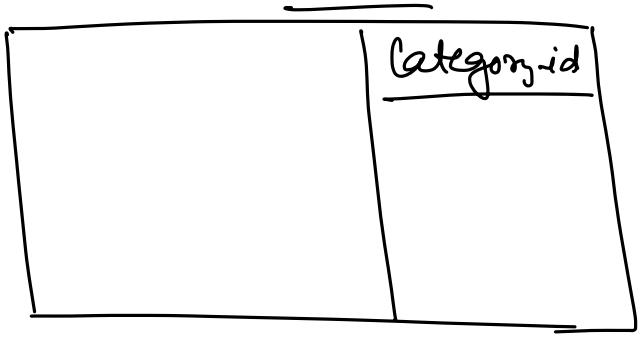
↳ CRUD operations on Products.

Class Diagram.

Cardinality.



products.



Categories.