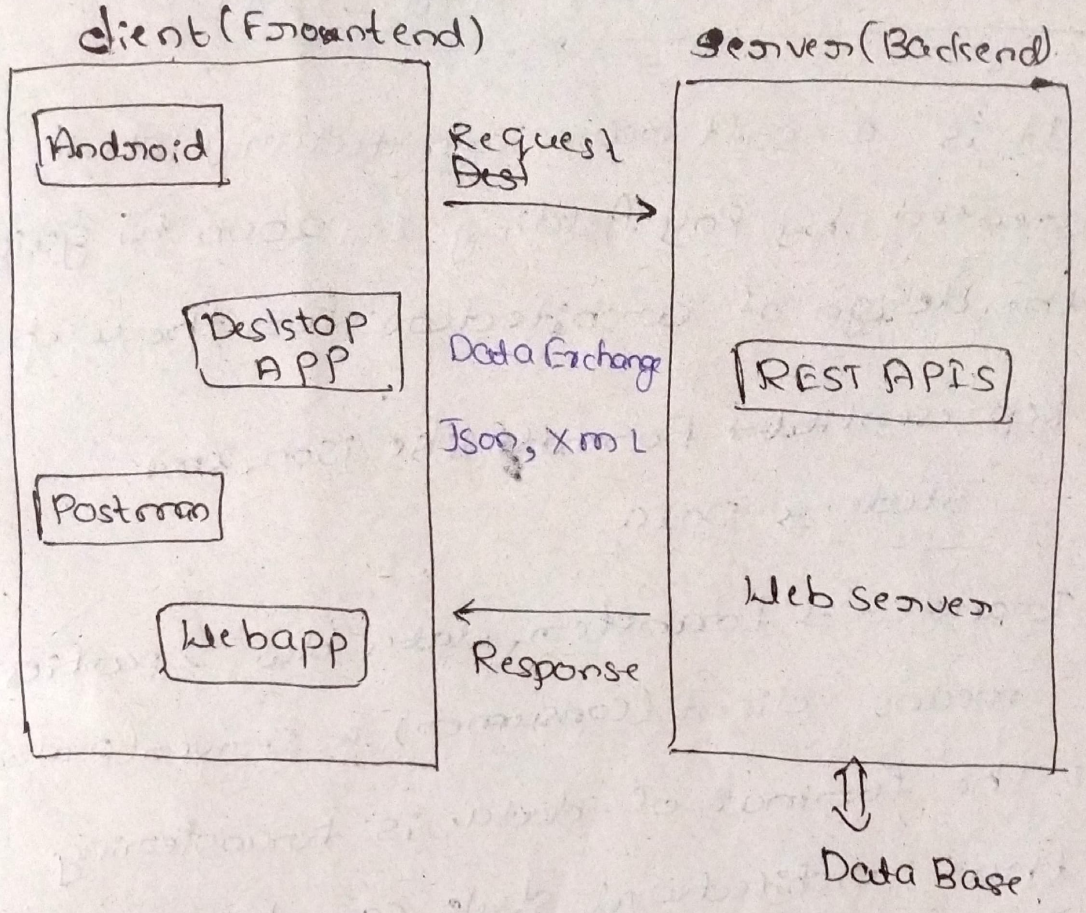
Spring Boot



* Inside Server we deployed the applications, here we write REST api's.

* Client we use the api's, we can send request to the server.

→ When client send the request to the server, then api's we exicute, if api's are need then it uses database access, fetch the data, then response send back to the client.

→ Data Exchange if client send the data is json from as request, then server take that json do the all processing and response back in json format.

*REST (Representional state transfer):

→ It is a software architectural style created by Roy fielding in 2000 to guide the design of architecture for the web.

→ Represential ⇒ Format like json, xml.
State ⇒ Data.

Transfer ⇒ transfer data blw 2 parties means: client (consumer) & Server (producer)

→ The format of data is transfering that architectural style can be represent by REST

→ The format of data is transfer blw 2 parties as client & Server

* REST Guidelines:

* client Server architecture (does not depend frontend & backend on each other)
* stateless (does not store any data in server)
* cacheable (if we are calling same api; & fetch the data again & again from database, we use cache means we can store some data then we can use that whenever requried)
* Layered System
* Uniform Interface - interaction blw client &

server must be uniform

* code on demand (optional)

* REST concepts:

→ Resource (REST api)

→ Sub-resource

→ URI (Uniform resource identifier)

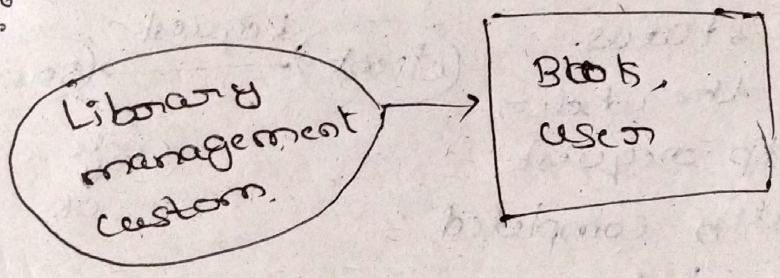
→ Http methods

→ Http response code.

* Resource:

→ Any thing that we want to expose to outside world, through our application.

ex:



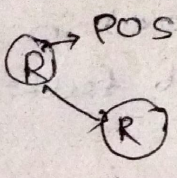
* URI (Uniform resource identifier):

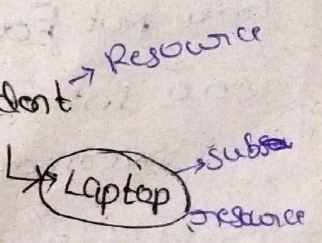
→ URI is used to identify resource

ex: GET http://localhost:8282/students/

* Sub-resource:

ex:





Syntax:

method | http://localhost:8282/resource/{id}/sub-resource

Ex:

Resource / RSD / sub-resource

GET http://localhost:8282/students/10/laptops

Return the list of laptops of student 10

→ Sub-resource cannot exist without resources

* Http Request Methods:

(Hypertext transfer protocol)

→ Http defines a set of request methods to indicate the desired action to be performed for a given resource

→ Http request methods are

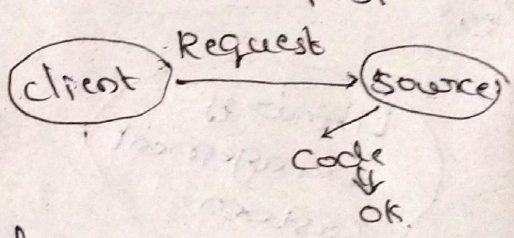
GET, POST, PUT, delete.

↓

create.

* Http Response codes:

* Http response status code indicate the whether a specific http request is successfully completed.



200 OK	Request is successful
201 created	Request is successful & new resource is created
401 unauthorized	Authentication is required for resource
404 Not Found	Resource not Found
500 Internal server error	Error occured on server & request can not fulfilled.

* Client Requirement:

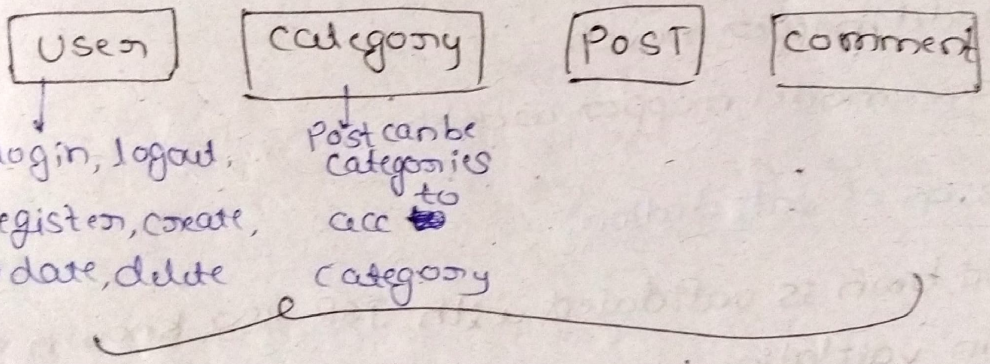
** We have a build simple blogging application.
→ user should create, update, delete & list posts.

→ user should add, update, delete, comments on post
→ categories the posts according to categories
→ New user should able to register on our application
→ user should able to login to our application
→ post include one picture too

* Technical terms:

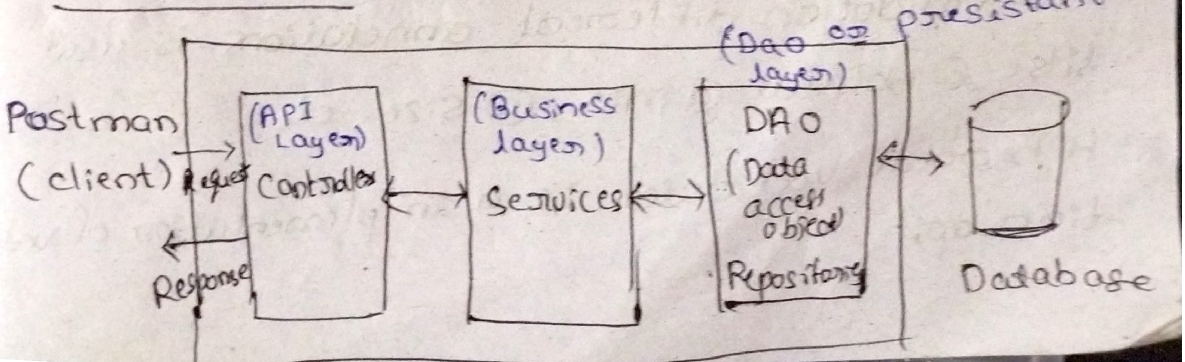
→ proper Register & login api
→ post, comment api
→ post API includes pagination & sorting
→ proper user i/p validation handling
→ proper exception handling
→ Role based authentication - role base security with api's.
→ JWT based authentication (JSON web token)

* Resources for Blogging Applications:



Resource for our application

* Best architecture while using spring boot for backend:



* Controller is API layer
* Service is Business layer or Business logic
* Dao is Dao layer or persistance layer which store the data

* Add dependencies:
→ Spring web (spring boot)
→ Spring data JPA (Hibernate)
→ lombok
→ Mysql
→ Spring boot dev tool (To avoid fresh project, it is reload the page atomatically)

* Imptant points:
→ draw.io
→ model mapper maven
→ How model mapper works.

* Basics of validation:
→ Java bean is validated with JSR 380 known as Bean validation 2.0
→ JSR 380 is specification for the Java API for bean validation. Properties of bean meet the specific criteria.
→ For validation different annotations is used like @NotNull, @min, @size etc.
→ Hibernate validator is a implementation of validation api.

* Important Annotations for validations:

@NotNull: If we don't want to keep the field as Null then we can use notNull.

@Size: length of the properties

@Max: maximum length

@Min: minimum length

@Email: validate email

@NotEmpty: For not empty field

→ In controller we add @valid annotation.

* How to use validations:

→ Spring boot provides support for Hibernate validator.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-validation
                                    </artifactId>
    <version>2.6.6</version>
</dependency>
```

* pagination:

→ page name & page size

→ we can sort also

→ ex: http://localhost:9090/post/?pageNumber=10&pagesize=5

→ If we want parameter we use @RequestPa

* Implementing Sorting with pagination:

http://localhost:9090/posts/?pageNumber=1&
pageSize=3&sortBy=id

or

http://localhost:9090/posts/?pageNumber=1&
pageSize=3&sortBy=title

* If we want in asc and desc order:

http://localhost:9090/posts/?pageNumber=1&
pageSize=3&sortBy=title&sortDir=asc or desc

* ~~Post Image or add image~~:

[codewithdurgesh.com]

* post Image or add Image: (7:40 - 8:00)

* ~~Comment~~ Add comment to the post

* Securing APIs with spring security:

→ we use JWT authentication for secure the api's

* Basic Authentication: (8:30-

* JWT implementing in project:

* Role specific access:

* challanges while registering app

* ~~* JWT token~~

* JWT authentication:

* JWTAuthenticationEntrypoint → used when user is unauthenticate.

* JWTAuthResponse → Here we write/store the token

→ JWT is stateless mechanism means which gives the security but no data will store on server.

* Swagger: (Documenting APIs)

→ It is open source & professional toolset.

→ Help to design & document APIs at scale.

→ How many url/api
→ which api's.
→ methods (Get, post)
→ Request
→ Response.

} Gives to F.E. FE for develop

→ Springfox boot starter dependency we need to add for swagger

https://localhost:8082/v3/api-docs

↓

get all docs as json.

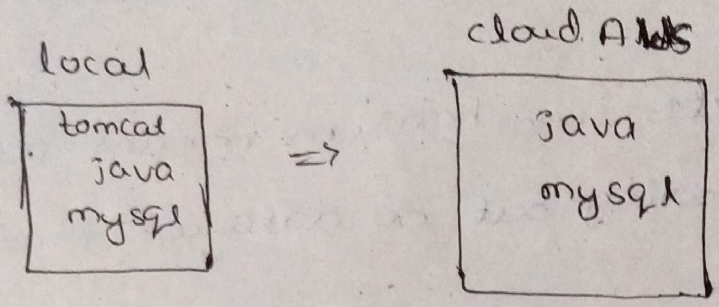
* http://localhost:8082/swagger-ui/index.html

↓

open in any browser gives all api's & all details.

* we cannot access the post, delete, create api's Because we applied security for these api's so we using swagger configeration

* Deploying Spring Boot App on AWS:



→ * AWS provides many number of services.

* comman services:

1. Amazon EC2 → manually we need to install java, mysql & configure manually.
2. AWS Elastic Beanstalk → requried to upload only code & Everything will done automatically.
3. Amazon RDS → provides Rational database like mysql.
4. Amazon Route 53 → DNS (Domain name system) means Domain name can be connected to the application through IPaddress.
5. Amazon S3 → used for retrieve any amount of data from anywhere.

* Managing different environment:

⇒ we will use profile concept for managing config-urations.

→ In Amazon RDS we will create database where give username, password.

→ Then username, password, port, hostname can be mentain on the ~~datat~~ mySql. then particular table will be created.

→ Then we can access the all tables from mySql workbench

* Package backend app in jar:

→ Select project in eclipse → Run as → maven build → Apply → Run

→ shows the Build success.

→ Select project ⇒ show In → system Explorer clicks on project - target - jar file will be shown here

→ open in cmd.

→ Run java -jar jarfilename.jar

If i geting error than mvn clean.
mvn clean install

then run again java -jar jarfilename.jar

→

* Deploying SpringBoot application on AWS:

(Elastic Beanstalk)

* Steps:

1. open Elastic Beanstalk → create application → select platform as java → upload your code → select jar → ok.
2. Configure more options → Databases → give username and password → save → create app
3. By default will run on port 5000
4. Here we get error because spring boot will run different port
5. So first change spring boot port num again rebuild & redeploy it.
6. open the link as .com and example ---- .com/swagger-ui index.html

* Spring Security without the WebSecurityConfigurerAdapter:

* Alter the autoincrement and truncate
* use student-tracker;
* Alter table student Autoincrement=3000;
* truncate student;

↓

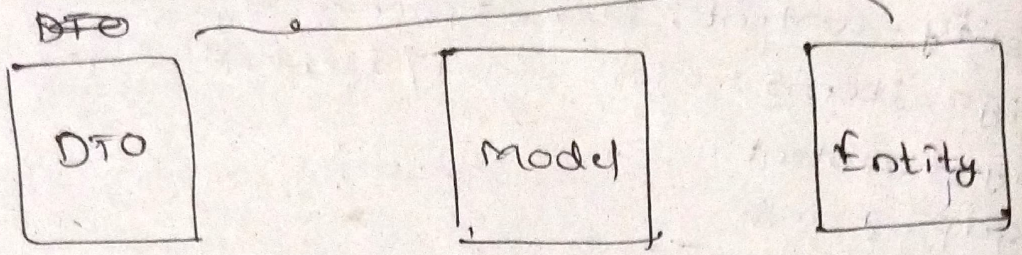
Removes all data from the database table

Reset auto_increment starting with 1.

* ~~Diff~~ DTO, model & Entity:

→ Job is same but at different layer

Data transfer object



weblayer → Servicelayer → Database layer

(controller) (Service) or (Repository)

(client will need)

{ FullName, age }

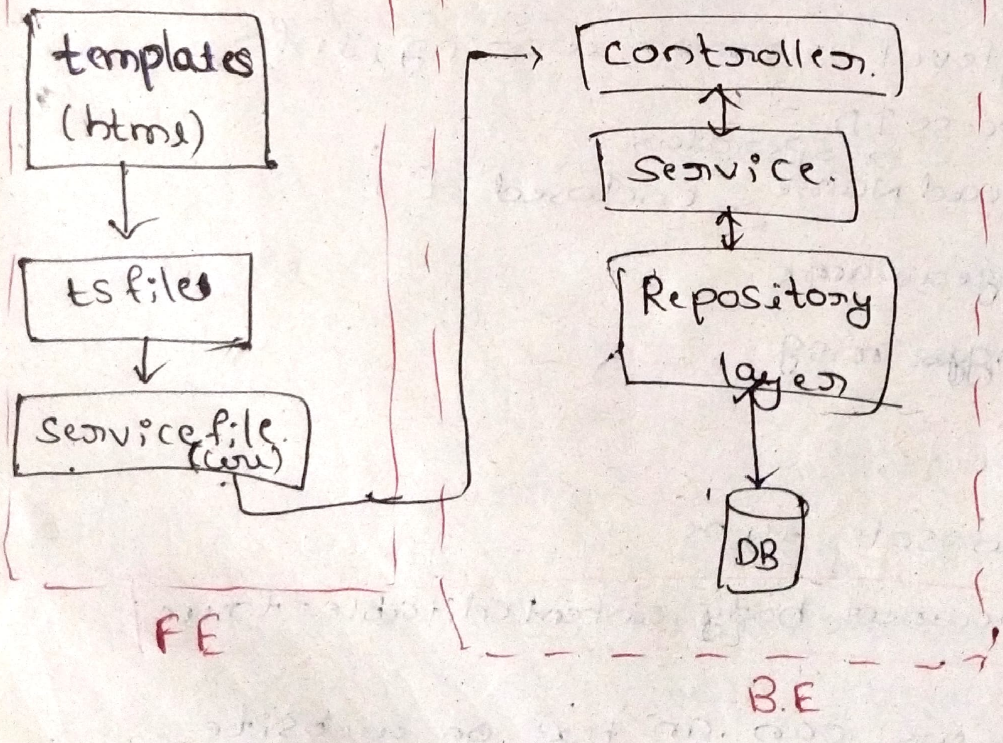
(Business layer)

fN, LN, fullName, age

firstname, lastname, DOB

* POJO vs Bean vs Entity vs DTO/VO vs model/Domain:

* FE & BE connections:



* Redis: (Remote dictionary Serve)

→ 'kubernets' is one platform to run microser

→ Redis is in-memory data

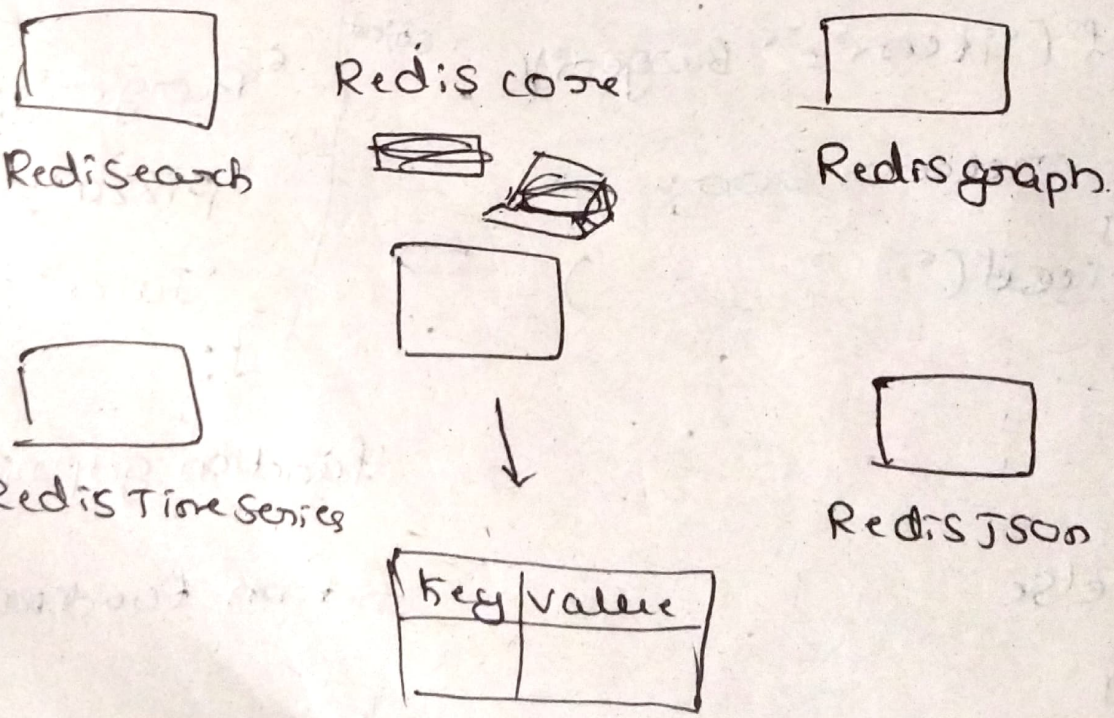
→ Redis is fully-fledged primary database

→ persist multiple data formats

→ Redis is used for multi-model database

→ Redis is supports for multiple data~~base~~ format in ~~an~~ single database

→ Redis is NOSql like mangoDB, ...
(Different than RealtionDB like mysql, oracle)



* Note:

→ Collections.sort(list); // ascending order
Collections.reverse(list); // descending order

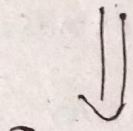
(or)

list.stream().sorted().forEach(----) // ascending

list.stream().sorted(Comparator.reverseOrder())
forEach(----) // descending

* Data Transformation & Flattening:

Stream.of('a', 'b', 'c', 'd');

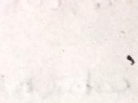


[A, B, C, D]

*Converting lowercase to uppercase and stores in another stream called as.
'Data transformation'

* Flattening

[[1, 2], [3, 4], [5, 6]]



[1, 2, 3, 4, 5, 6]

Convert streamof stream into single stream is called as 'flattening'.

* optional: (If we have null values then we can use it)

public Optional<String> email { return Optional.ofNullable(email)};

→ avoid the NullPointerException.

* Caching:

@EnableCaching:

@Cacheable(cacheName="books", key="#isbn")

↓

First time used it is @annoted on get resource

@CachePut(cacheName="books", key="#isbn")

↓

For update annoated on update method

@CacheEvict(cacheName="", allEntries=true)

↓

used for delete the information.

@Caching → used to specify multiple annotations of the same type as @CacheEvict or @CachePut

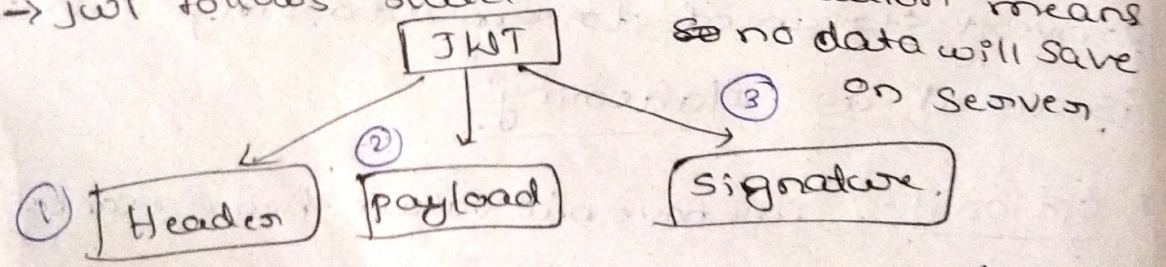
@CacheConfig.

(io.jsonwebtoken)

* JWT: (Json web token) [authorization: token]

→ It is token based. authenciation & which is used to secure api's.

→ Jwt follows stateless authenticaion means no data will save on server.



x.y.z → Token