Server 🡪 tomcat application server

Building the application 🡪 maven build tool

Framework 🡪 springboot

DB 🡪 mysql

Db conneting ORM tool 🡪 hibernate JPA repository

Postman 🡪 API testing

Unit testing 🡪 Junit and Mockito framework

Plugins Lombok

HTTPServletRequest 🡪 request send by User from frontend

HTTPServletresponse 🡪 after processing request the end response will be send back to UI and that response will be displayed.

Here request will be in Json format from UI

Registration form:

Name:

Id:

Age:

Backened🡪 json

Student:

{

“Name”: “data”,

“id” : 1,

“age” : 12

}

**HTTP methos:**

POST 🡪 whenever your adding the data 🡪 will use HTTP POST

GET 🡪 for fetching records

PUT 🡪 updating data

PATCH-> partial update

DELETE 🡪 deletion data

**Http status codes:**

Along with response we will pass status code

Clientside error codes 🡪 400

Server side 🡪 500

Success codes 🡪 200

200 🡪 success response

201 🡪 created new records

204 🡪 no - content

400 🡪 bad request

401 🡪 unauthorized

403 🡪 forbidden no access

404 -🡪 not found

500 🡪 internal server error

502 🡪 bad gateway

503 🡪 service unavaible

Restfull webservices:

REST 🡪 representational state transfer 🡪 it is architecture designing application and coding it

Front end -🡪 display

Db: name, version, date, price

JSON 🡪

Student json request

{

“name”: “data”,

“id” : 1,

“age” : 12

}

Student model:

Private String name;

Id

Age

Setter

Getter methods

A 🡪 B 🡪 C 🡪 D 🡪 cascading calls 🡪 tightly coupled 🡺 new key to create object

Handled to make it loosely couple by **dependency injection.**

**We wont create object**, we will define but IOC container will create object for us with the help DI.

Public class Student {

@Autowired

Address address; 🡪 no qualifying bean type found

address.fnd();

}

**IOC** 🡪 beans 🡪 configures 🡪 manages that beans till its usage

Beans 🡪 object 🡪 by default it is singleton bean

@Bean

Public Address getAddress() {

return new Address(); // 1 singleton object

@Component

public class Address {

}

public class Student {

@Qualifier(name=”name of bean impl class”)

@Autowired

Address address;

}

Public void College {

@Autowired

Address address;

}

.java

.class

.Jar 🡪 group of .class files combines into file 🡪 jar file

.War 🡪 group of .jar files combines into single file 🡪 .war file

.Ear 🡪 group of .war files combining into single file -🡪 .ear file

Maven build tool 🡪 snapshot-applicationname .jar

Deploying jar file into server(tomcat) 🡪 extract the jar and it will run the all the files code 🡪 port number 🡪 8080

POM.XML 🡪 maven application 🡪 dependencies (jars)

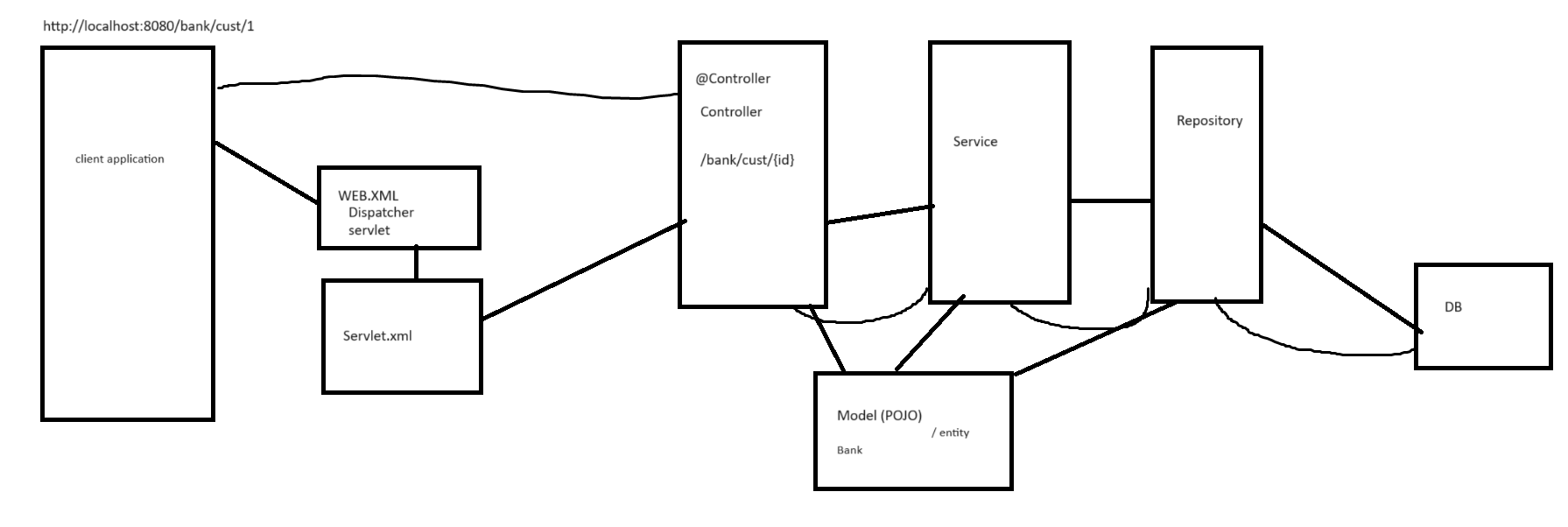
Stereotype annotation

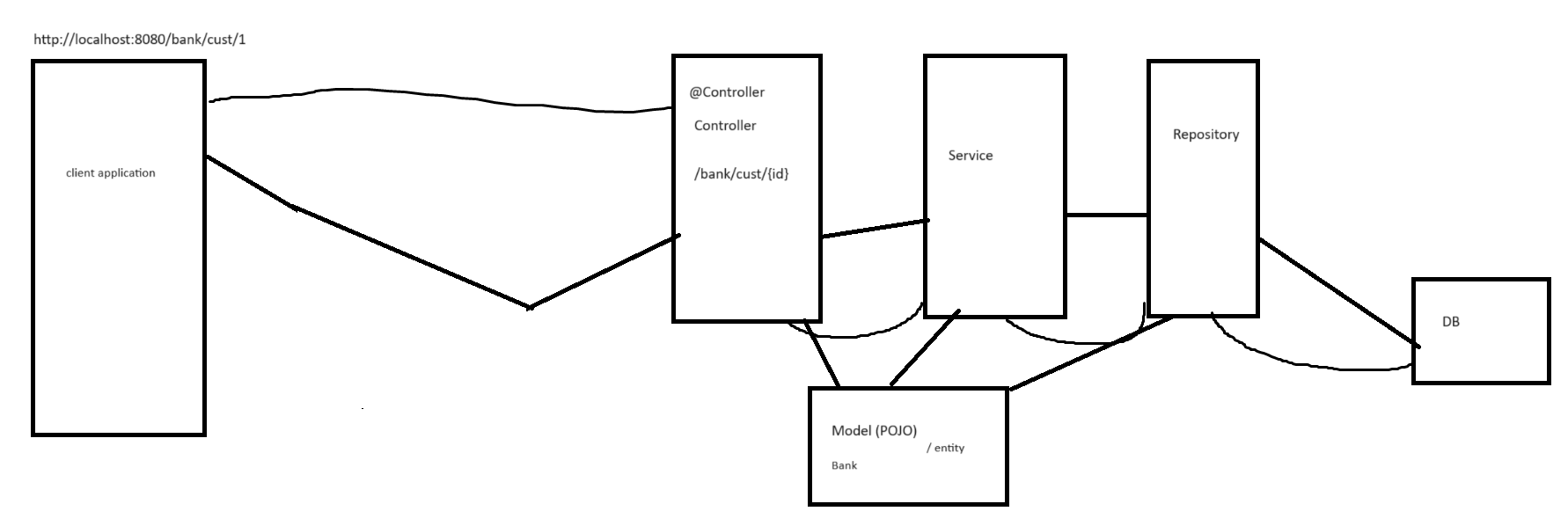
@Component 🡪 bean 🡪

@Controller 🡪 all urls related class on top of it

@Service 🡪 all business logic related

@Repository 🡪 all the database related 🡪 JPA hibernate queries





Drawbacks:

Dependency issues

Tomcat and maven should be downloaded and configured

Spring related configuration and application related configuration we need write

Springboot:

Autoconfiguration

Embedded tomcat and maven

Dependency issues will be resolved 🡪 starter POM.xml

@SprinbootApplication 🡪 main method class , which consists of EnableAutoConfiguration, ComponenScan, SpringbootConfiguration

@EnableAutoConfiguration 🡪 autoconfiguration will does by

@ComponenScan 🡪 all the components will be load to our classpath

@SpringbootConfiguration 🡪

@Bean 🡪 object

@Configuration 🡪 class as configuration

@Component 🡪 class as entire single bean

@Service 🡪 business related

@Repository 🡪 database classes

@Controller 🡪 URLs related

@Qualifier 🡪 to which bean to call

@Primary 🡪 to specify which bean to call in impl

@ResponseBody 🡪 java object to Json object and send to HTTPresponse

@RequestBody 🡪 json body HTTPrequest will be converted to Java objects

@RestController 🡪 @Controller + @ResponseBody

@RequestMapping 🡪 URL mappings

@Pathvariable 🡪

@Requestparam -> key value pair

@ControllerAdvice

@RestControllerAdvice 🡺 @ControllerAdvice + @RespondeBody

@ExceptionalHandler

@Value(key)

@PostMapping 🡪 Adding records

@GetMapping 🡪 Fetching records

@PutMapping 🡪 updating records

@DeleteMapping 🡪 deleting records

@PatchMapping🡪 partial update records

Animal

Hi();

DogIMPL

Hi(){

}

CatImpl

HI(){

}

JPA -> Java persistence API 🡪 its specification 🡪 ORM tool 🡪 Hibernate

JDBC 🡪 queries 🡪 configuration details

Db table columns with java objects

Name 🡪 studentName

Hibernate 🡪 ORM tool

Object relation mapping

Object 🡪 table columns

Cache 🡪 temporary memory storage

Sessions 🡪 1st level cache 🡪 query executions

Session factory 🡪 2nd level cache 🡪 multiple session 🡪 datasource 🡪 database related inform

**Get(), persist(), load(), save(), saveOrUpdate(), fetch Type EAGER vs LAZY**

Hibernate mappings

OnetoOne

OneTOmany

ManyTOOne

ManyTOMany