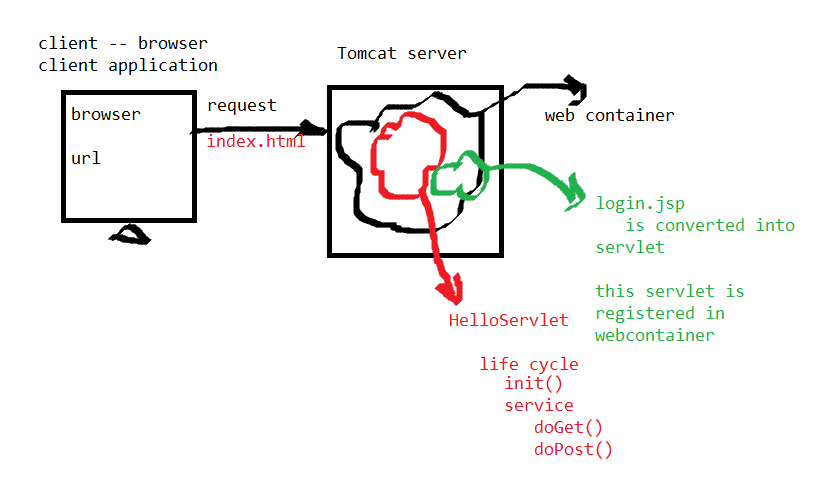
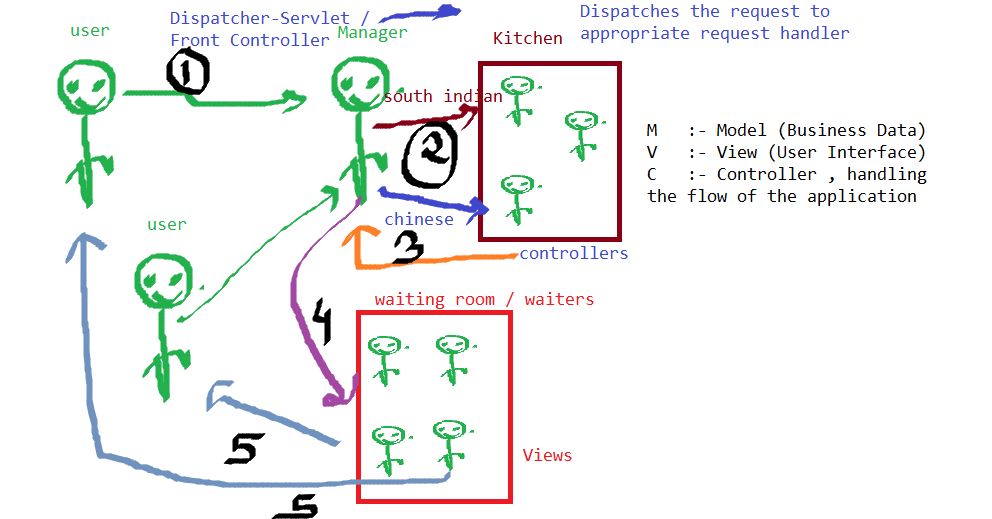
**SPRING MVC**





**Creating spring mvc project:**

1. Create a maven project

Archetype-webapp

Groupid :- com.spring.mvc

Artifact :- springmvc

1. Build path-> configure build path-> targeted runtime environment-> enable tomcat server-> finish.
2. Configure web.xml and pom.xml

(creating a new java folder gives src/main/java. And create a views folder in webINF)

1.     What is the difference between an application server and a Web server?

A Web server exclusively handles HTTP requests, whereas an application server serves business logic to application programs through any number of protocols.

**The Web server**

A Web server handles the HTTP protocol. When the Web server receives an HTTP request, it responds with an HTTP response, such as sending back an HTML page. To process a request, a Web server may respond with a static HTML page or image, send a redirect, or delegate the dynamic response generation to some other program such as CGI scripts, JSPs (JavaServer Pages), servlets, ASPs (Active Server Pages), server-side JavaScripts, or some other server-side technology. Whatever their purpose, such server-side programs generate a response, most often in HTML, for viewing in a Web browser.

**The application server**

As for the application server, according to our definition, an application server exposes business logic to client applications through various protocols, possibly including HTTP. While a Web server mainly deals with sending HTML for display in a Web browser, an application server provides access to business logic for use by client application programs. The application program can use this logic just as it would call a method on an object (or a function in the procedural world).

Spring Boot project

4 different ways for creating Spring Boot project  
1. Create a maven based project and add starter dependency  
2. Spring boot command line interface  
3. STS (IDE like Eclipse)  
    Spring Tool Suite  
4. Using Spring Initializr tool \*\*\*\*\*\*  
        (Chrome)

=========================  
As a developer our actual job is to focus only n only on business logic

 not on objects creation, maintainance, configurations,...

Java developer         100%    efforts (Everything )  
Spring Java developer    40%        60% -- spring  
            .xml  
            .java  
            annotations  
            BL  
Spring Mvc        configuration

SpringBoot developer    20%        80%    --spring  
            Conventions    Configurations  
            Business Logic    Identify beans, registering                     beans, life cycle, autowiring,                     dependency injections,

No .xml file  
No java configuration file

**SPRING BOOT**

4 different ways for creating Spring Boot project  
1. Create a maven based project and add starter dependency  
2. Spring boot command line interface  
3. STS (IDE like Eclipse)  
    Spring Tool Suite  
4. Using Spring Initializr tool \*\*\*\*\*\*  
       (Chrome)

**Creating SpringBoot Project:**

1.Go to chrome and search for <https://start.spring.io/> which is **spring initializr**

2. Select the type of project as Maven and language java, spring boot version 3.0.2

3. In project metadata,

Group is com.capg(any package)

artifact is FirstSpringBootProject(any name)

name is same as artifact

package name is same as group

packaging as jar and java version is 17

4. Add dependency 🡪basic dependency is added ie., Spring Web

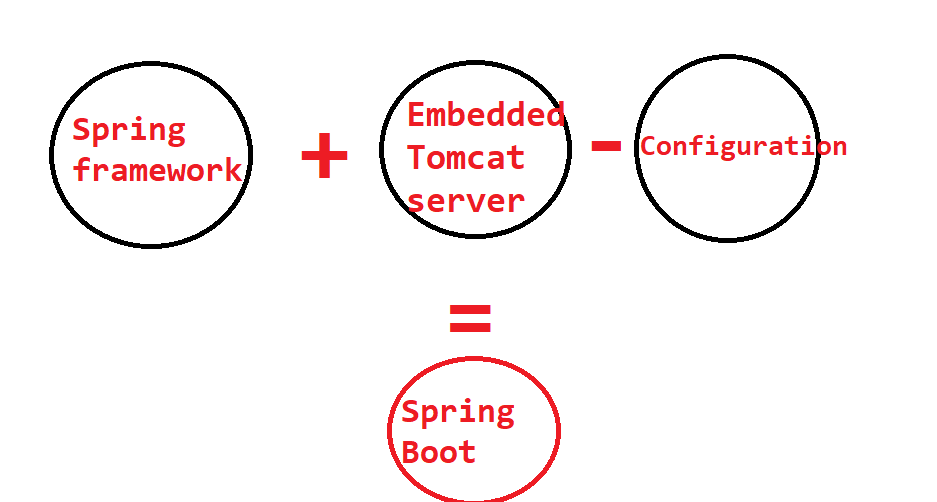
Based on the requirements can also add progresql jdbc driver, spring data JPAand so on..

Add them only when required or can add after starting the project also.

5. Click on generate.

6. Extract the generated zip folder of project and add it in the eclipse environment.

7. In eclipse, **import -> maven->existing maven projects->browse->select the firstSpringBootProject folder and check if the pom.xml is checked and finish**.

****

SpringBoot has a special file called as application.properties  
                    application.yaml

Says focus on Conventions , not on configuration

@SpringBootApplication has all these 3 annotations in it.

    @Configuration  
    @ComponentScan  
    @EnableAutoConfiguration

=========================

1. CoC software design style  
Convention over Configuration

2. Opinionated Default   
Automatically configured

3. Scans the component, identifies the dependencies , auto injections..

**RESTFUL SPRINGBOOT**

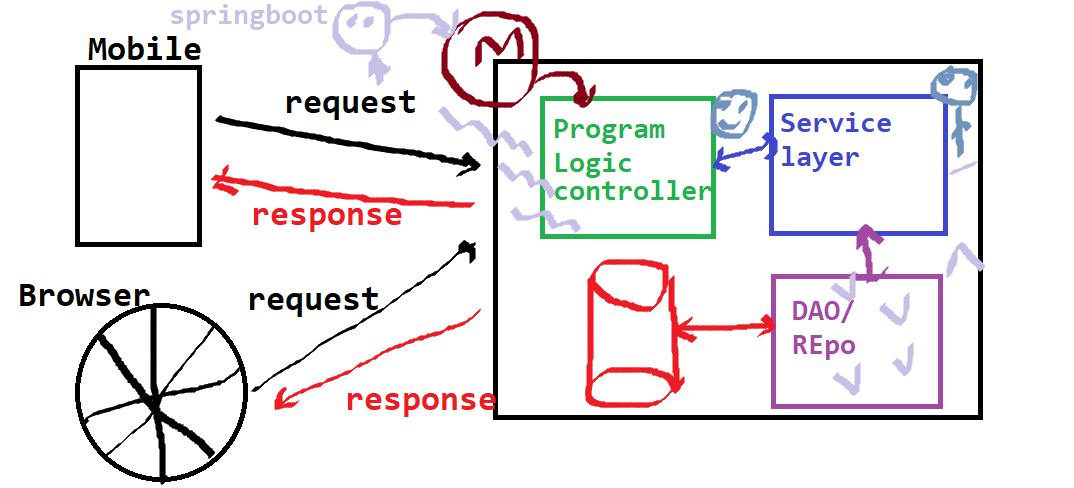
Product  
Customer  
Users  
============================  
Service Layered Architecture (Factory Design Pattern)  
Layers  DAO layer    Service        Program Logic

  entity        dao        controllers    Mapping Handlers

User    User.java    UserDao(i)    UserController    login ("/login")  
            UserDaoImpl                register("/register")

Product    Product.java    ProductDao(i)    ProductController  
            ProductDaoImpl              
            create()            ("/product")  
            update()            ("/product")  
            delete()            ("/product")  
            display()            ("/product")

Customer Customer.java    CustomerDao(i)  
            CustomerDaoImpl    CustomerController  
            create()            ("/customer")  
            update()            ("/customer")  
            delete()            ("/customer")  
            display()            ("/customer")



From SpringBoot to RESTful SpringBoot

 Two annotations:                        Replace with  
@Controller          
@ResponseBody                        @RestController

**SERVICE LAYERED ARCHITECTURE:**

Model

DAO

Interface

Implemented class

Controllers

Handlers ->methods

Rest API mapping based on crud functionality

JPA SERVICE LAYERED: (control flow of code)

* Project starts from the controller and dispatcher servlet dispatches the requests based on the mapping.
* based on the mapping, the handler is invoked. handler is your controller and this controller handler is requesting to the service layer ( employee Dao) layer to get all the employees.

//return this.employeeDao.getEmployees();

* So employee Dao impl is autowired as a service And it is injecting our dependency of a repository because you don't want to write any piece of code you want JPA repository to do everything for us.

//@Service

//Class EmolyeeDaoImpl

//{

//@Autowired

//Private EmployeeRepository repository;

//}

JPA repository has built in methods like findAll(), findBy(), countBy(), getBy()..

* Using JPA just create your own method if you want & don't write a query. JPa will write a query for you… Only follow the standard conventions and the method name starts with findBy(), countBy(), getBy().

https://stackabuse.com/spring-data-jpa-guide-to-the-query-annotation/

======================RESTful API ========================  
@RequestMapping(value="/" , method=RequestMethod.GET)    @GetMapping      
@RequestMapping(value="/" , method=RequestMethod.POST)   @PostMapping  
@RequestMapping(value="/" , method=RequestMethod.DELETE) @DeleteMapping  
@RequestMapping(value="/" , method=RequestMethod.PUT)    @PUTMapping

This above is CRUD implementation

 Create        :- PostMapping  
Read        :- GetMapping  
Update        :- PutMapping  
Delete        :- DeleteMapping

RESTful web services are executed on third party platform   
Swagger  
Postman

\*\*\*(Mainly we focus on Model-Controller)(no view)

Controller->handler

View-> .jsp(on a web page using html)

Wt exactly is the use of postman?

Is list static? Cant we delete a record from list?

@RequestBody in handler parameter to see what the postman is adding into the list.

Spring boot restful annotations:

@requestController

@GetMapping

@PostMapping

@PutMapping

@DeleteMapping

@Autowired

@Service

@Pathvariable

@RequestBody

When m2/maven repository is corrupted, then

* Close all projects in eclipse
* Delete m2 repository in file manager
* Open eclipse and select project->maven->update project

JPQL

Global exception handler

Spring boot(user defined) repository interface extends JPA repository.

JPA repository has built-in methods to

* Find a record by id
* Save an entity
* Delete an entity
* Update existing
* Display all records of a particular entity

\*\*Behind this hibernate framework generates queries based on the mapping called.

\*\*JPA implements JPQL behind..

JPQL project creation(Maven project ->quickstart archetype):

1. Create maven project with com.hibernate and **jpqlhibernateproject**
2. Take persistence.xml and paste in src/main/resources/meta-inf(create folders)
3. Add dependencies in pom.xml -> hibernate core,jpa,postgresql
4. Create city.java in com.hibernate
5. Go to Java persistence api for hibernate in maven repository and add the dependency.

<https://stackabuse.com/spring-data-jpa-guide-to-the-query-annotation/>

1. Write query in the form of string  
2. createQuery using entitymanager  
3. Execute it.

JPQL works with classes and instances. Retrieves data through entities not by sql tables.

Query methods:

* getResultList

global exception handling:

<https://springframework.guru/exception-handling-in-spring-boot-rest-api/>

MOCKITO FRAMEWORK

Stub: incomplete object which can be done by us./ fake service

When aggregating the interface in a class where the class cant access its methods without initialization but interface cant be initialized, we create a constructor in the class and initialize the interface using parameter given in the constructor and the parameter to the constructor is passed from test case call of class

Create a stub(anonymous inner class for interface) for interface in test class so that we can run the class and test it.

\*\*Mockito can mock non static and non-final services.