SPRING BOOT USING MICRO SERVICES

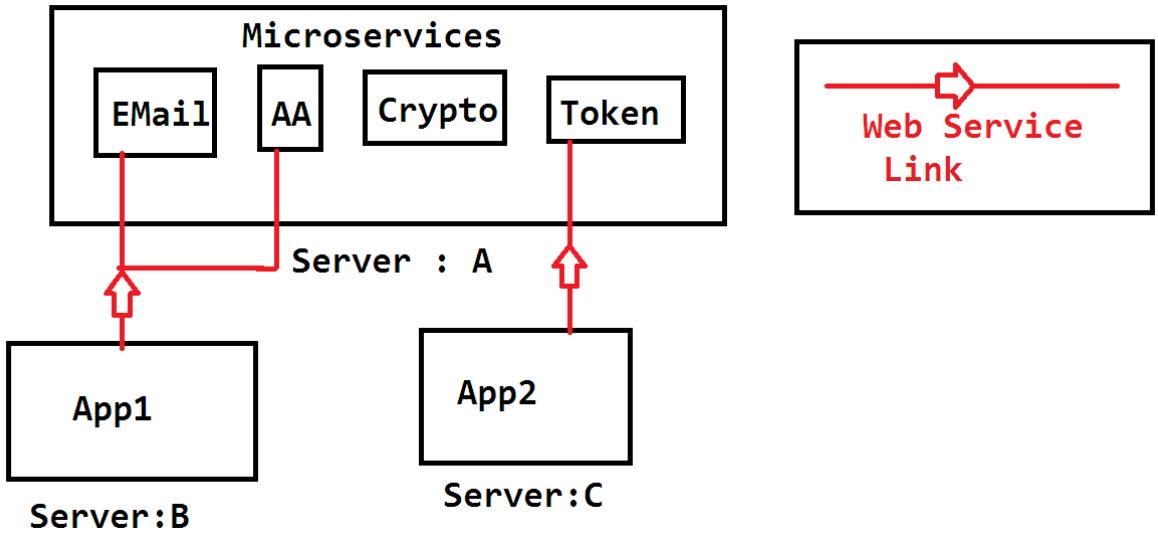
(PROJECT DOCUMENT)

Service:

A Service is a special application or code which provides extra function to existed project. Java Example Services: Java Mail, Java Securing (JAAS = Java Authentication and Authorization), CODEC(COding and DECoding), Java Design Generators(JFreeCharts), Java File Operations(Upload and Download) Java Cryptography, Java Unique Key Generators etc...

Micro services:

A Service must run independent and it should be re-usable , implemented using less coding(small part) is called as Micro Service. Every Re-usable service can be converted to micro services.



Note: A micro service should be Re-usable service, Independent (must run in diff. server), Plug-in (Link using Web-Services or parent project concept)

PROJECT DOMAIN: WAREHOUSE

A warehouse is a commercial building for storage of goods. Warehouses are used by manufacturers, importers, exporters, wholesalers, transport businesses, customs, etc. They are usually large plain buildings in industrial areas of cities, towns and villages.

They usually have **loading docks** to load and unload goods from trucks. Sometimes warehouses are designed for the loading and unloading of goods directly from railways, airports, or seaports. They often have **cranes** and **forklifts** for moving goods, which are usually placed on ISO standard pallets loaded into **pallet racks**. Stored goods can include any raw materials, packing materials, spare parts, components, or finished goods associated with agriculture, manufacturing and production. In Indian English a warehouse may be referred to as a godown different types of locations

Pallet

loading docs

Pallet Racks

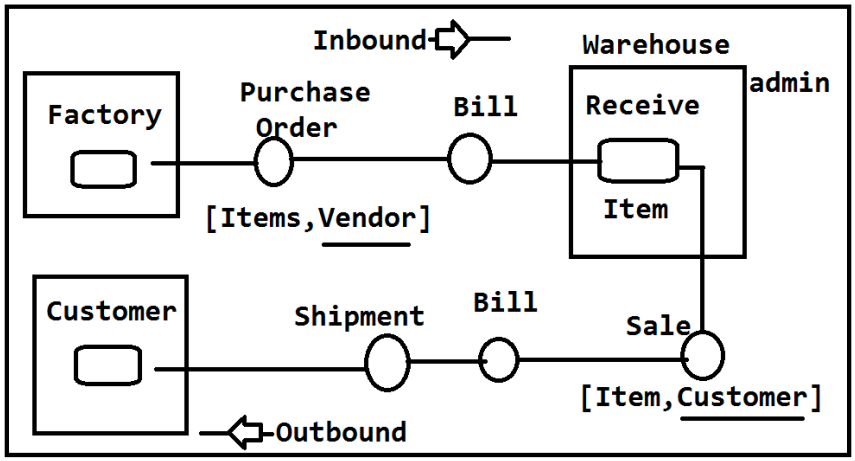
Mobile Crane

forklifts

Storage Types in warehouses are :

- Cold storage
- Dry storage
- Rack Storage
- Open Storage

Sending an item/part into warehouse is called as inbound operation. Purchase order comes under Inbound operation. Sending an item/part out from warehouse is called as outbound operation. Sales Order and Shipment Orders come under outbound operations.



Warehouse Application:-

Warehouse is a storage location where different items are stored location wise connected to vendor. We are developing an Automation application which handles Warehouse operations for Item management for any kind of users.

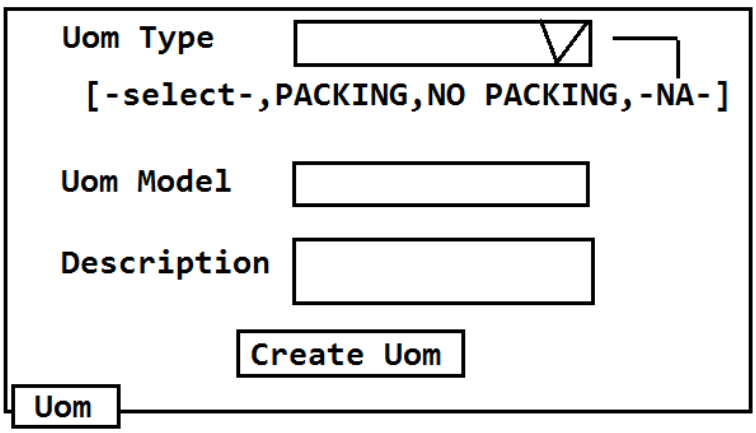
These are mainly divided into 3 types

1. Basic Modules
2. Inbound Operations
3. Outbound Operations

1. Basic Modules : Also called as Master Data Management, which handles data related to basic information of an item like

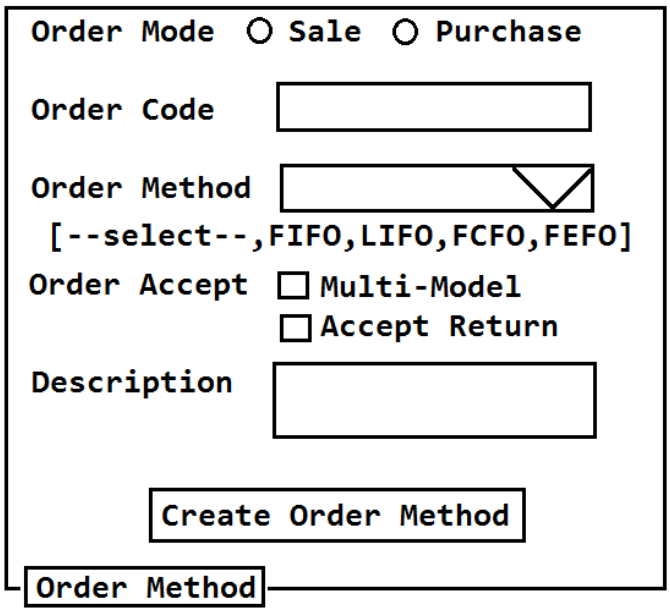
UOM : Unit of Measurement Which contains Packing details with model and description. It can be calculated in Models like BOX Small, Medium, Large, Pallet , Plate, Rack, Ultra Box, Semi closed etc..

To create and manage these we are creating one Module in application screen looks as below,

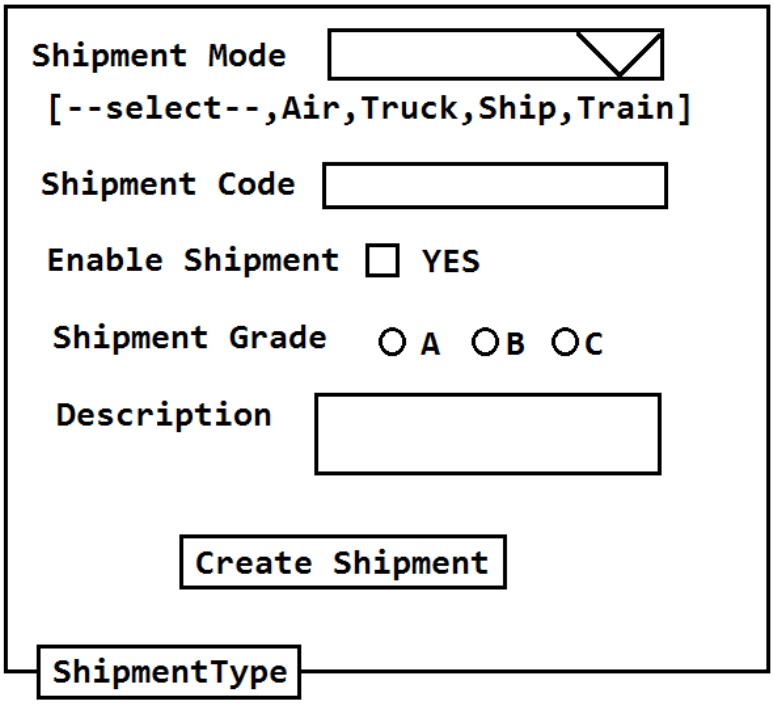


Order Method : Every Order can be classified into 2 types One is Purchase Order and second one is purchase Order. For Every Order should have one flow method. And every item should connect to

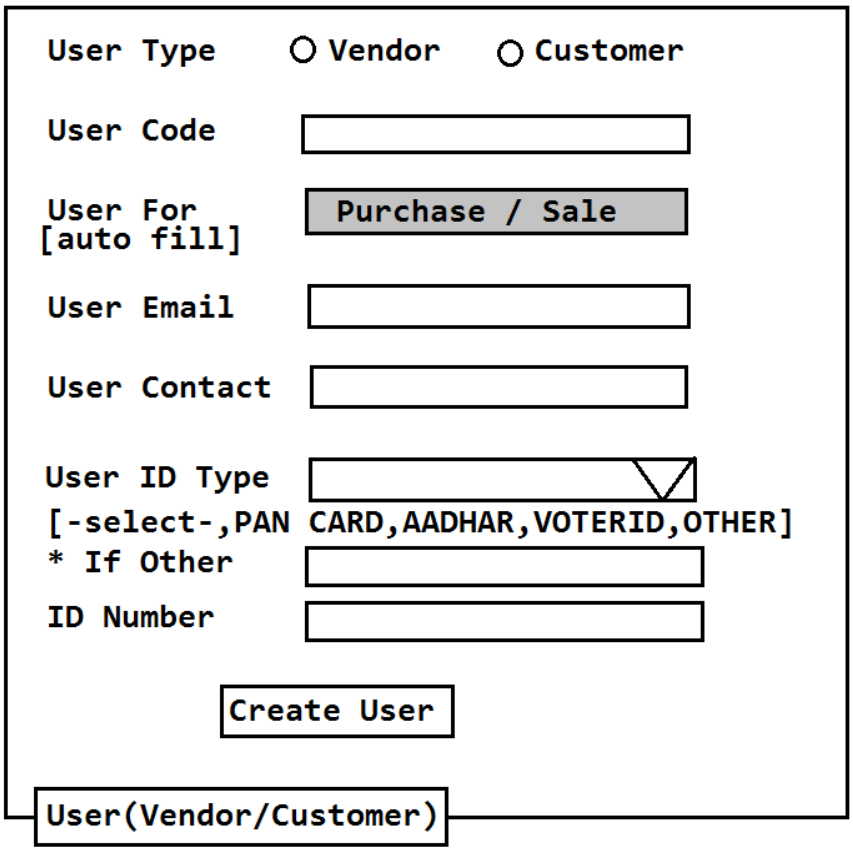
one sale type and one purchase type. For that we need to create unique order Codes with unique Mode combinations. It can extra details like multiple models like different colors and different models are accepted or not. If damaged return is accepted or not, like this all details can be provided. Basic description can also be provided along every record in application, screen looks as



Shipment Type: Shipment process should be connected to one shipment type, application Item can be shipped using different ways like Road, Rail, Air, Sea etc.. All these types must be entered into application, one of this can be selected before creating one shipping order in project. It can be enabled and disabled if the available shipment comes under un-availability. Every Shipment mode will be identified with one unique code in project with basic description. Screen is given below,

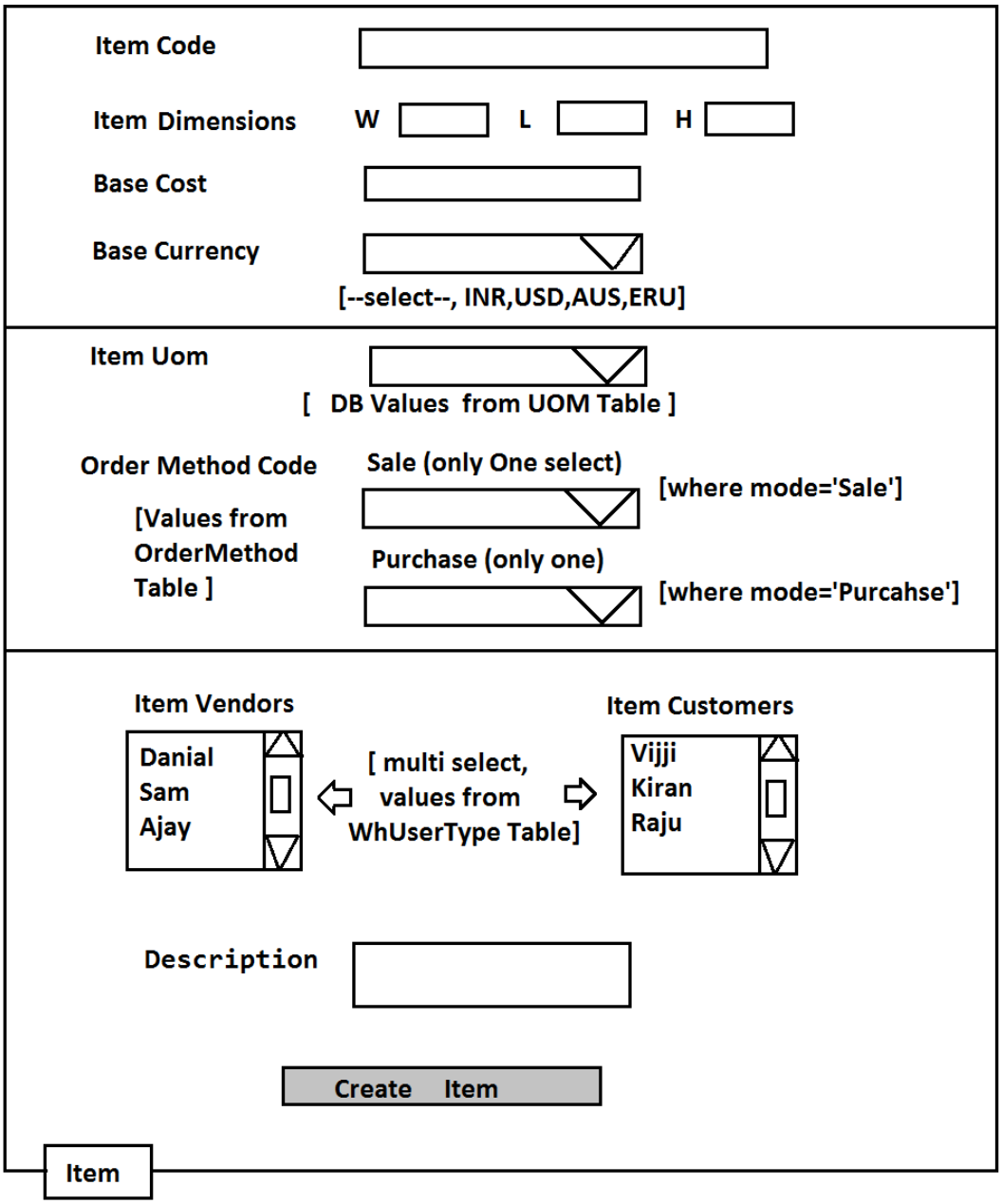


Vendor and Customer : To handle an item related operation, we must provide details of item like it's owner and quantity related to Item owner. For those details we have provider owner details in User of Item details, which are also called as Ware House User Types (WHUserTypes). Every user must have identification details with complete communication information in application. Vendor or Customer one Type should be selected. Vendor is used in Inbound Operations and where Customer is used in outbound operations.



Item: Warehouse application main usage is storing of items. So we must create every item details in project like name, code, cost, currency, Uom details, Sale and Purchase Order Codes. Every Item should connect with multiple Vendors and Multiple customers also.

Only Vendor related Items are shown in Purchase order and Customer connected Items are shown in Sale Order. So at least one Item Should connect with one vendor and customer to use in operation.



Operations Modules: 1. Inbound Operations 2. Outbound Operations

1. Inbound Operations: These operations indicates "Getting an Item into Warehouse, from factory". These operations are:

I. Purchase Order
II. Vendor Invoice
III. GRN(Goods Receive Note)

Once this operation is done quantity of items will be increased (+Quantity)

Purchase Order(PO): Screen#1 Main Details Screen#2 Item Details

Step#1 Go to screen#1 Enter Details Code, Mode, Vendor etc.. PO will be created with status "OPEN" (Ready to order-zero items in Order).

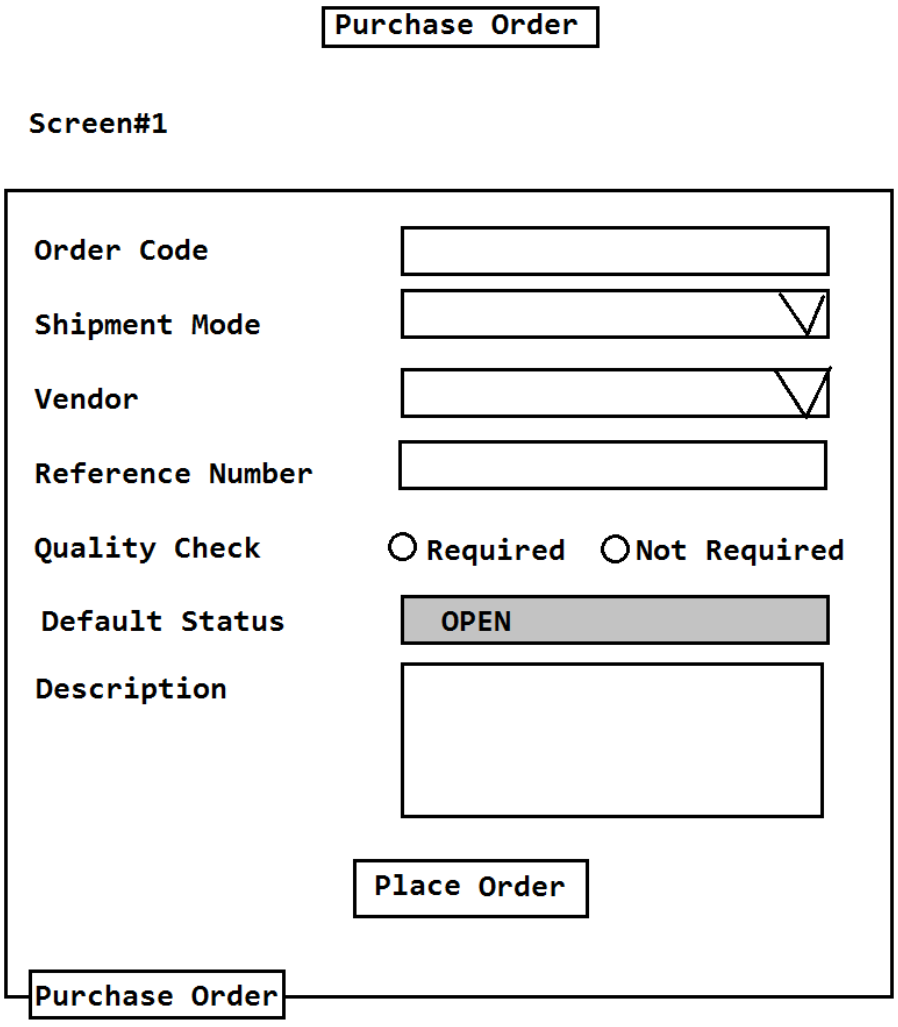
Step#2 Display All Items in Data Page. if Status is OPEN -You can add items using Screen#2.But only selected vendor items will be displayed.

Step#3 Select Item and quantity, click on add Item. Items should be added to PO and status will be "PICKING" ie PO having >=1 item. If status is PICKING then PO Edit should not allow Vendor modify.

** If we remove all Items again status should be "OPEN".

Step#4 If Order is selected >=1 item then display "save and continue" button. On click this No More Edit & No More Select items (is equals to ORDERED/CONFIRM).

Step#5 From any status we can cancel Order. PO status will be "CANCELLED". CANCELLED Order cannot be used for adding items or Process Order



Screen#2 : It is represented by Purchase Order Detail model class

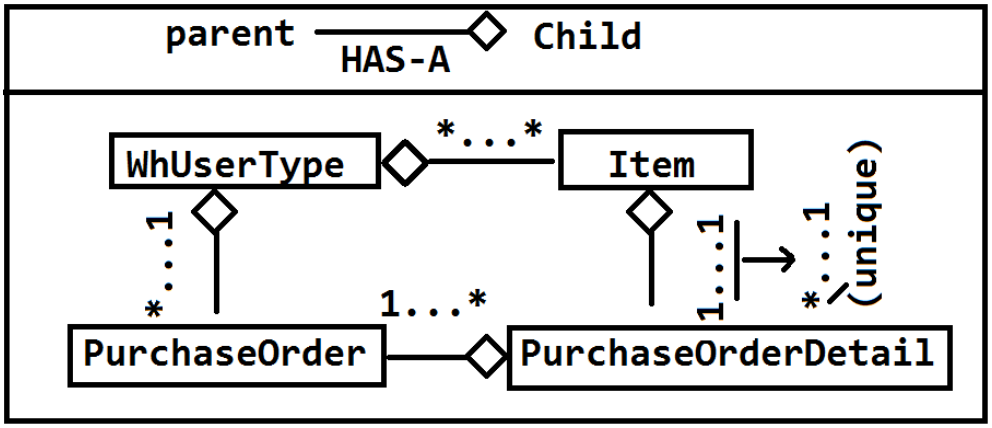
1) One Order can contain multiple Details ie 1...*

PurchaseOrder----<>PurchaseOrderDetail

2)One Detail object contains one - slno one - item and its base Cost & quantity

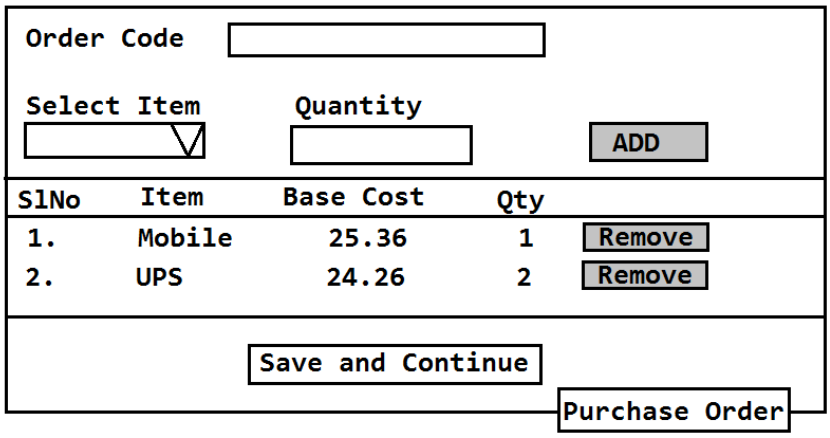
so 1...1

PurchaseOrderDetail-----<> Item

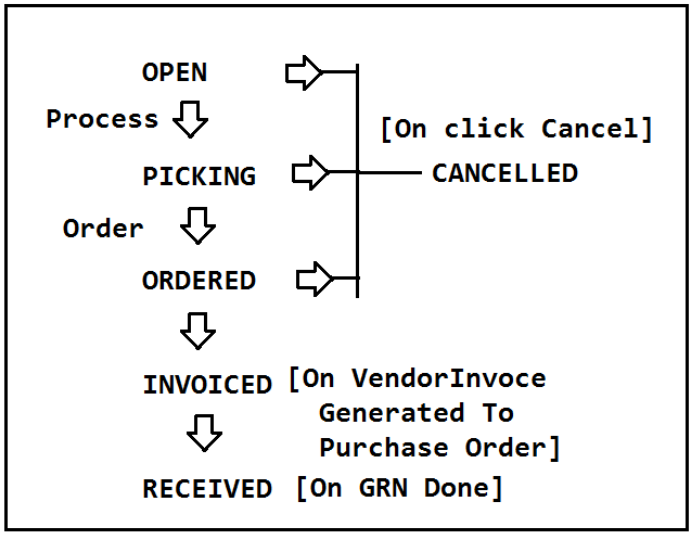


If we observe Screen#2 every line (item details) is one Purchase Order Detail object.

Screen#2



Purchase Order Status Flow:



Vendor Invoce

Vendor Invoce Code

Vendor Code | Order Code

Final Cost | Shipment Type

Item Details

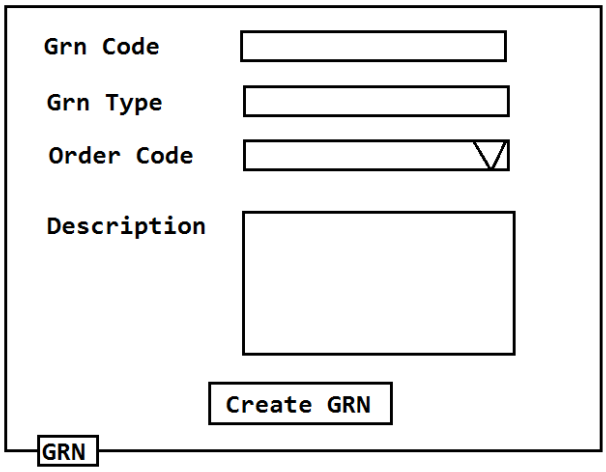
Item Code	Base Cost	Quantity	Item Value

Generated On: DD/MM/YYY HH:mm:ss

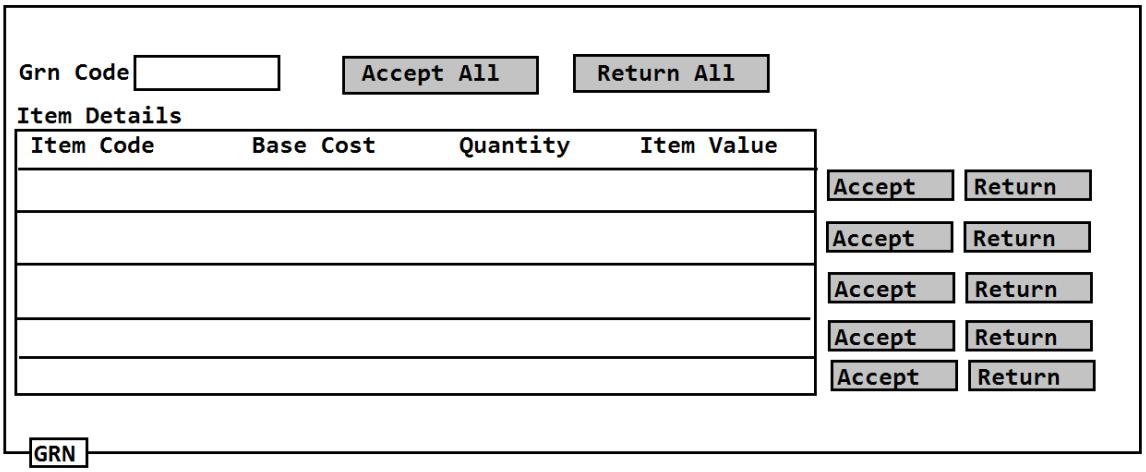
Vendor Invoice PDF

Good Recevice Note(GRN)

Screen#1



Screen#2



2. Outbound Operations: These operations indicates "Sending an Item out from Warehouse to customer/end customer".

I. Sale Order
II. Customer(Sale) Invoice
III. Shipment Items

On performing Outbound operation quantity of item will be reduced (-Quantity)

Module Implementation Using Spring Boot and Layers Design:-

To implement one module we need to design

1. Model class(Entity Class/DTO) DTO= Data Transfer Object.

2.Design one Interface in DataAccess Layer that extends another Spring JPA interface

"JpaRepository (I)" (org.springframework.data.jpa.repository)

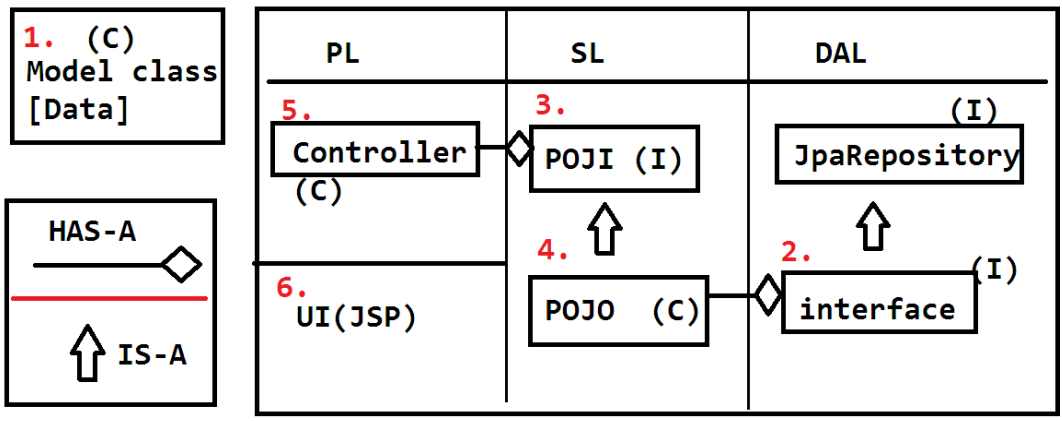
3. Design Service Layer using POJI-POJO design pattern which is used to connect(link) layer using Loosely coupled Designs.

4. Define Presentation Layer using Controller class which takes request to process it.

5. Define UI pages for Data Register, Edit,Show,Search,Delete etc (JSP+JSTL+Bootstrap)

** Between layers Apply HAS-A Relation to link one layer with another. Current Layer class connects to next layer interface.

Design:-

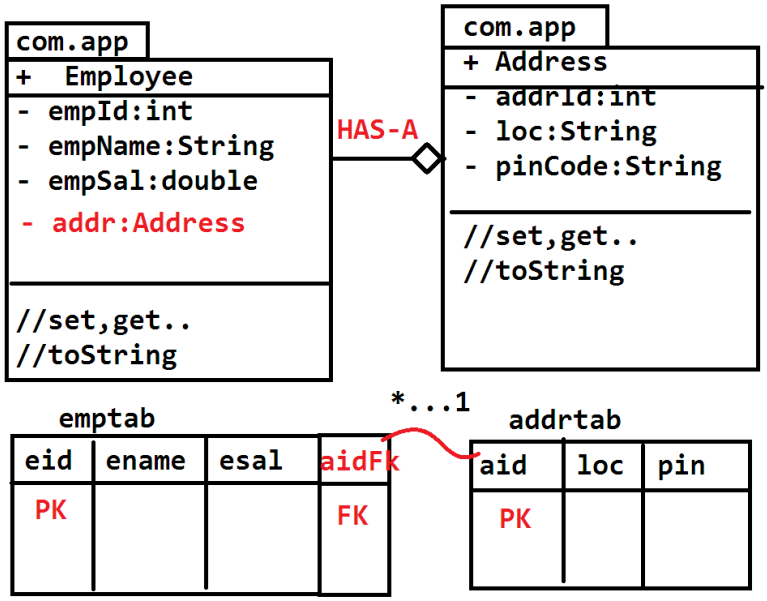


1.Model class/Entity class/DTO/VO: DTO=Data Transfer Object VO = Values Object =means DTO.

Model classes are used to create DTOs by Spring Container or Hibernate(ORM). These objects are used to transfer data from UI to DB and DB to UI.

HTML Form = Object = DB Table Row

Design:-

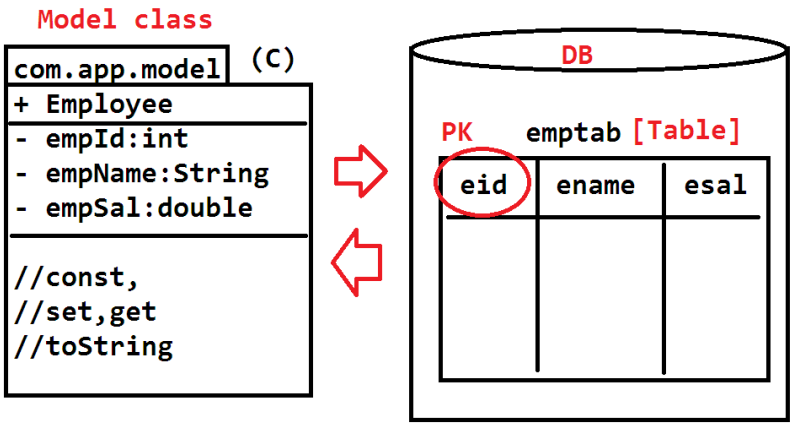


Programmer should write one model class that must be mapped with DB table using JPA Annotations. Mapping must be done as class Name must be mapped with Table Name and variable Name must be mapped with Column

Example:-

```
package com.app.model;
//ctrl+shift+O (imports)
javax.persistence.*;
@Entity
public class Employee {
 @Id
  private int empId;
  private String empName;
  private double empSal;
  //set,get..toString..
}
```

Here @Entity maps class with table and variable with columns. @Id indicates Primary key column. Here Table Name is (className) EMPLOYEE and columns are(variableNames) empId, empName, empSal .



Example#2:

```
package com.app.model;
//ctrl+shitf+O (imports)

@Entity
@Table(name="emptab")
public class Employee {
 @Id
 @GeneratedValue //GENERATE PK value
 @Column(name="eid")
 private int empId;

 @Column(name="ename")
 private String empName;

 @Column(name="esal")
 private double empSal;

 @Transient //ignore as Table col
 private String data;

 @Column(name="c_dat")
 @Temporal(TemporalType.TIMESTAMP)
 private Date createDate;

 @Column(name="c_dat_m")
 @Temporal(TemporalType.DATE)
 private Date lastModified;

 @Lob  //byte[]+Lob= BLOB (file data)
 @Column(name="emp_img")
 private byte[] fileData;
```

```
 @Lob  // char[]+ Lob = CLOB
 @Column(name="emp_txt_resume")
 private char[] txtFileDate;
  ..//set,get..toString..
}
```

- All annotations are from javax.persistence package. BLOB=Binary Large OBject and CLOB=Character Large OBject
- TemporalType is a enum, it has possible values DATE,TIME,TIMESTAMP.
- TemporalType.DATE ex: 13/09/2017
- TemporalType.TIME ex: 10:33:56 895
- TemporalType.TIMESTAMP ex:13/09/2017 10:33:56 854

Layers Design :- To implement one module we need to define model class that is mapped with DB table using JPA Annotation.

Once Model class is designed then start Layers Design. To implement one module we are taking

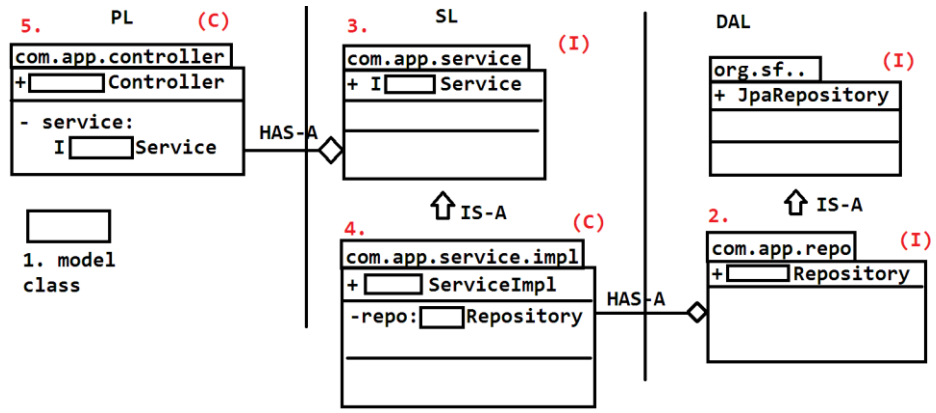
3 layers design.

1. Presentation Layer (PL)

2. Service Layer (SL)

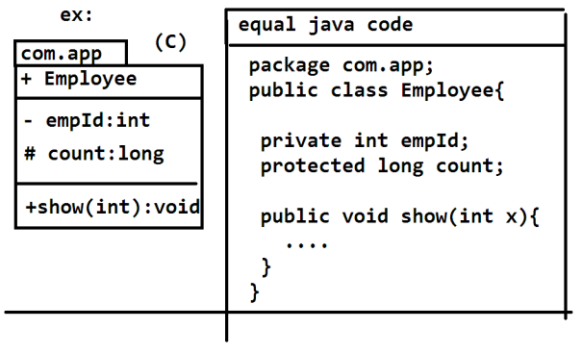
3. Data Access Layer (DAL)

4. Integration Layer (IL)

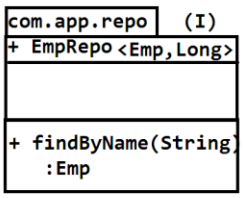


Coding order must be DAL->SL->PL/IL.

UML Designs: To define one class or interface we use notations of UML (Unified Modeling Language) which is applicable for OOP languages (like Java,.net, php etc). Java Relations between classes and interfaces

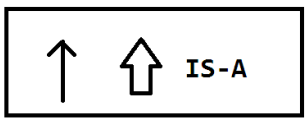


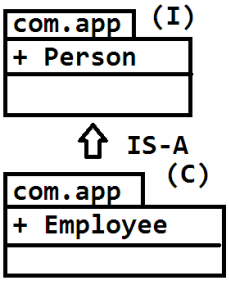
equal java code:



1. IS-A (Inheritance) use extends or implements key word
2. HAS-A (Association) Use child as DataType in parent class to create child reference variable.

i)IS-A Example

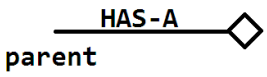


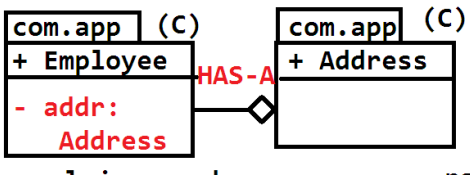


equal java code

```
package com.app;
public interface Person{ }

package com.app;
public class Employee
 implements  Person{   }
```

ii)HAS-A Example:

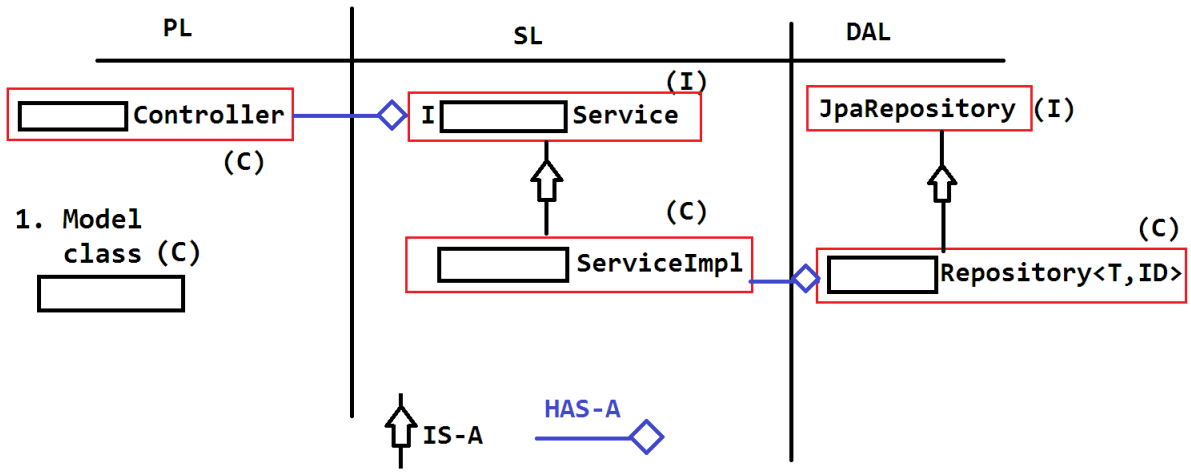




equal java code:

```
package com.app;
public class Address{ }
```

```
package com.app ;

public class Employee{
 private Address addr;
   //HAS-A
}
```



Module (Implementation) as a Service (MaaS)

We must implement one module which supports operations using

1. UI Forms and Tables in PL

2. Web Service calls for different clients and Integration Apps(Ex: Android, 3rd party Apps, service engines etc)

3. Bulk Operations (Data Upload and Download also called as Export and Import using Excel sheets).

For all this programming logic should be written only one time (in common layer ie) Service Layer. This Service layer can connect to Data Access and even Utils(Support classes). Use Server side (Service side) validator for all Request processing.

MaaS has a Standard Design for Module implementations which is re-usable for all types of Applications, changes will be at PL(Presentation Layer) or IL (Integration Layer). Calculations and common logics are written in SL and it's Utils. Common validations are written in Validator class level.

Design of MaaS:-

Common Code files in MaaS:-

1. Model class (C)

2. Repository (I) [DAL]

3. IService (I) and ServiceImpl (C) [SL]

4. Util (C)

5. Validator (C)

6. Controller(UI/Multipart) (C) [PL]

7. RestController/ServiceProvider (C) [IL]

MaaS

End To End Module Design & Coding

JpaRepository (All Operations)

Service and Controller Design

PostMan Tool

Bulk Operations (Multipart)

Pagination

Sorting and Search using Specification

Validator(PL & IL)

[Annotations & Validator]

Validator(Multipart)

Swagger UI

application.properties

Bootstrap (twitter API)

Modules Next Level Design

Services(Email,Security(AA))

Jpa Repository: Data Access Layer Design and coding:-

To perform DB operations like select and non-select (insert,update,delete) operations, Spring Boot DataAccess has provided one basic interface ie. CrudRepository (I) (org.springframework.data.repository) used mainly for JDBC type operations. To perform ORM (Hibernate)operations one extended interface is provided ie. JpaRepository(I) (org.springframework.data.jpa.repository)

Operations using Repository:-

1. findAll() : List<T>

2. getOne(id):T (or) findOne(id):T

3. save(obj):T

4. a) delete(id) (or) delete(obj)

   b) deleteAll()

5. exists(id):boolean

Consider Example Module "EMPLOYEE"

1. Model class

```
package com.app.model;
@Entity
@Table(name="emptab")
public class Employee {
 @Id
 @GeneratedValue
 @Column(name="eid")
 private int empId;
 @Column(name="ename")
 private String empName;
 @Column(name="esal")
 private double empSal
//set,get..toString.. }
```

then Data Access Layer code is:

```
package com.app.repository;
public interface EmployeeRepository extends JpaRepository<Employee,Integer>{ }
```

then, cosider

T = Model class Name = Employee

ID = Primary Key data Type=int(Integer)

id = Primary key value (ex: 25).

Operations:

1. findAll():List<T>

On executing this , it generates select SQL which selects all the rows from DB. Each row converted to one object, all Objects are stored in List format.

Design:-

equal SQL generated by Dialect

select * from emptab

2. getOne(id):T / findOne(id):T

To select one row based on primary key use this method.

Design:-

getOne(id):T is from JPA (ORM) findOne(id):T is from JDBC but final output is same.

If record exist returns Object else null Data selected only based on primary key.

3. save(obj):T

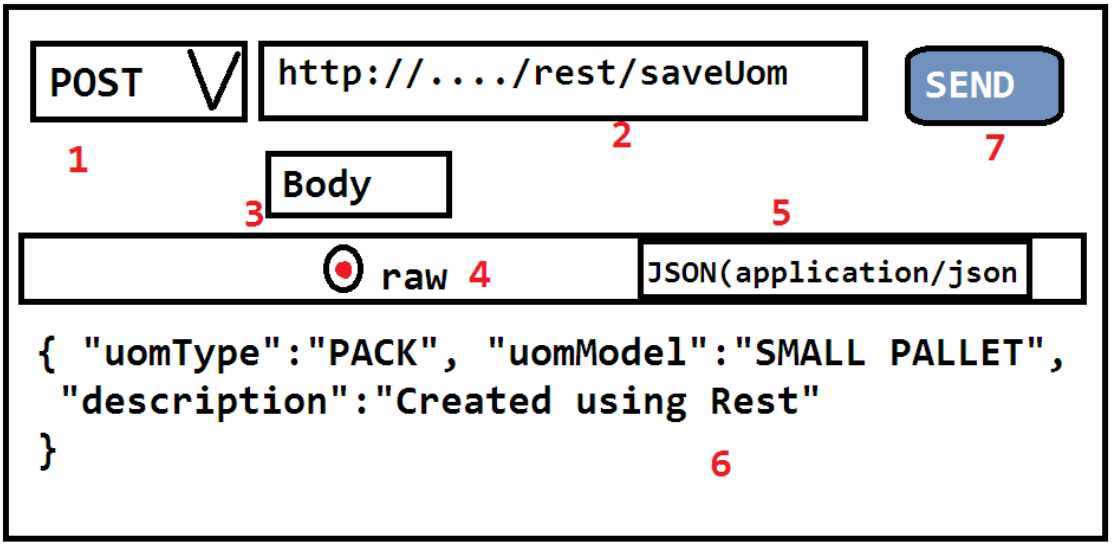
This method is used to perform save or update operation. Based on primary key column it will take decision.

It executes select count(*) SQL query which returns either zero or one if zero - insert record (save) if one - update all columns based on primary key column (update).

Design:-

POSTMAN SCREENS:

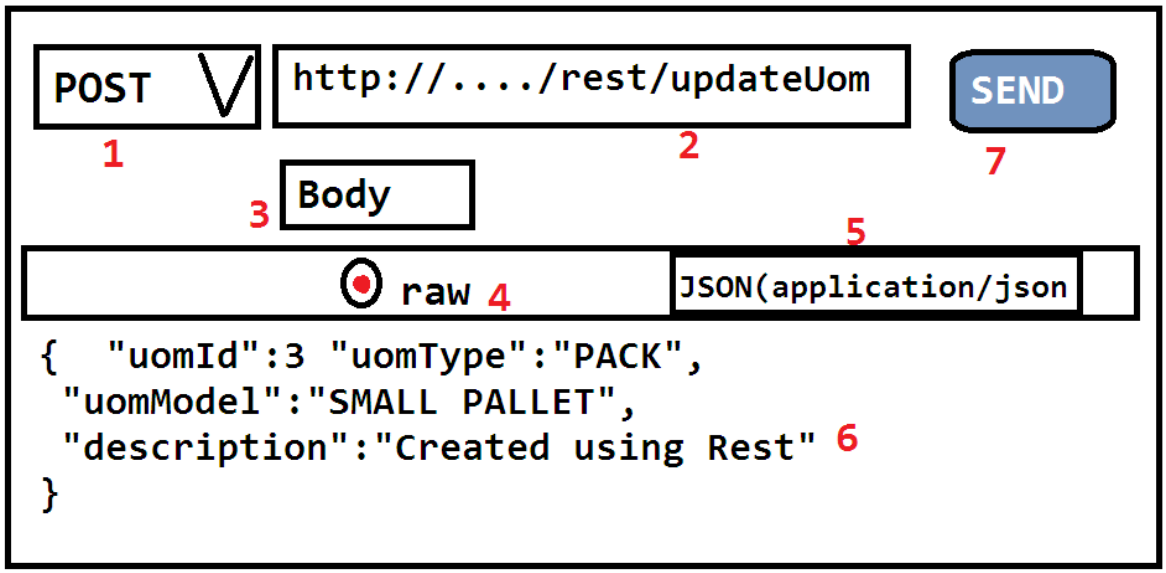
1. save UOM



http://localhost:2018/rest/saveUom

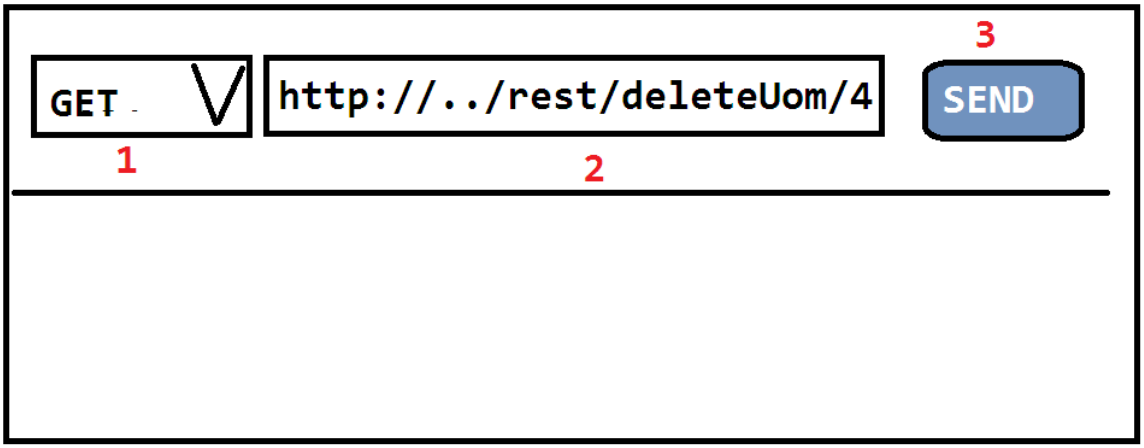
---

2. Update UOM



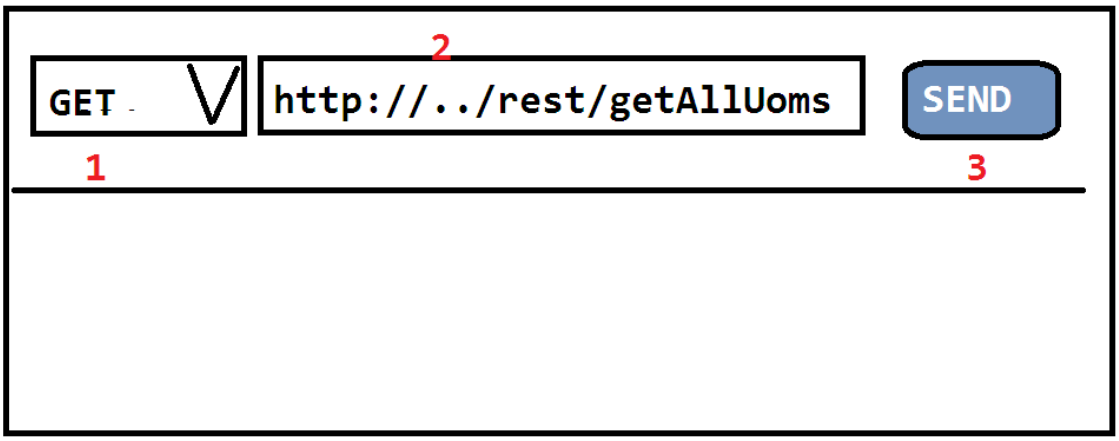
http://localhost:2018/rest/updateUom

3. Delete UOM



http://localhost:2018/rest/deleteUom/4

4. Get All Uoms

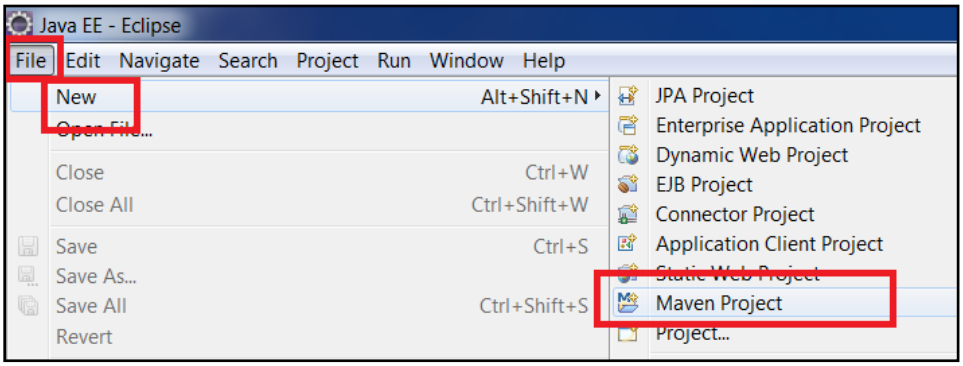


http://localhost:2018/rest/getAllUoms

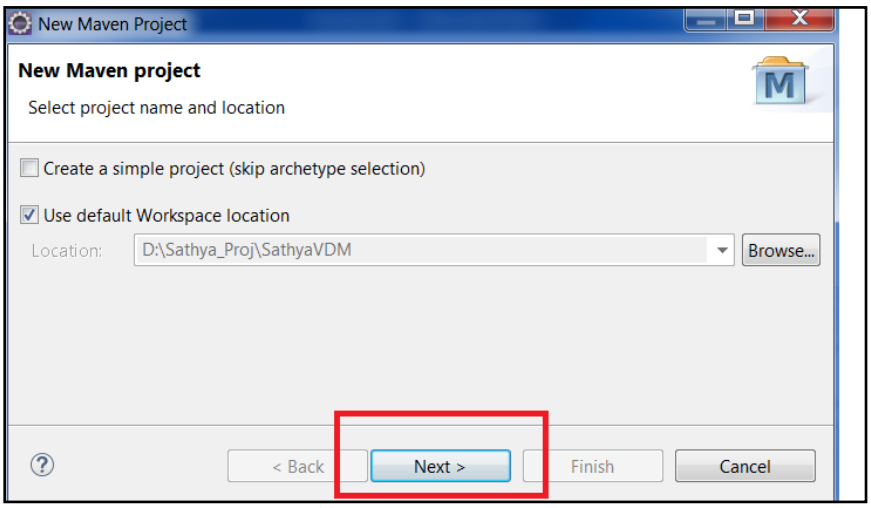
PROJECT SETUP

Create one new maven web project:

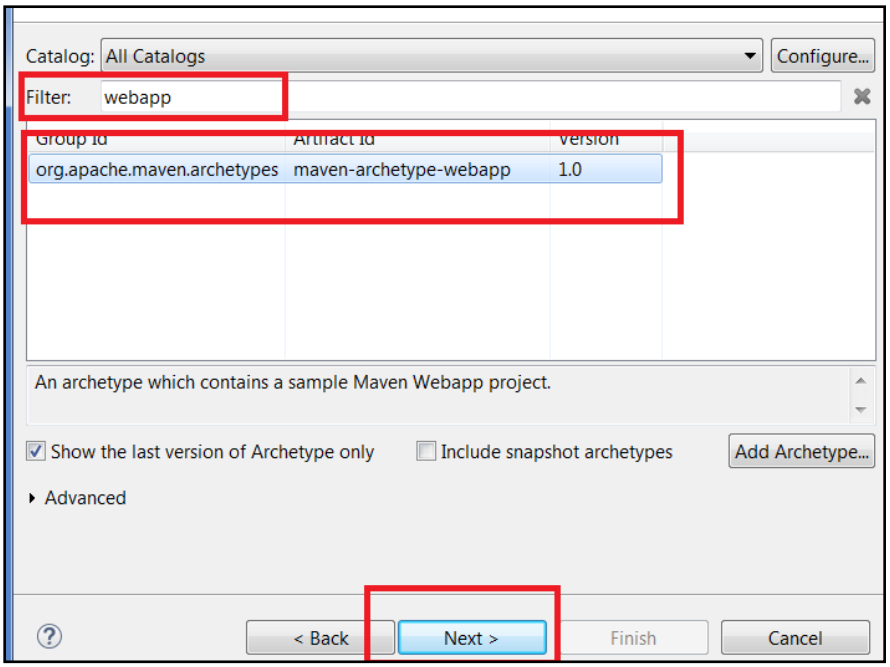
File => new=> Maven Project



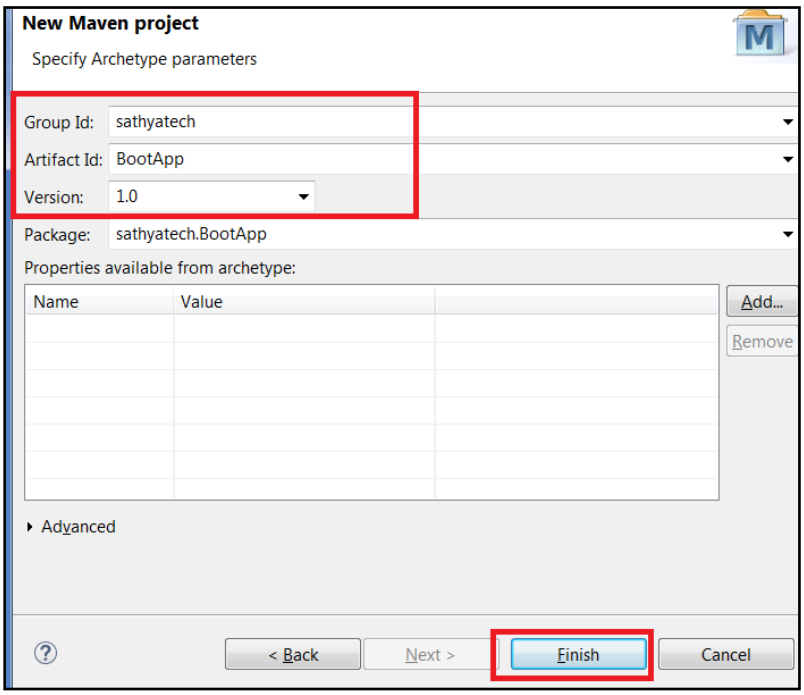
click on next button



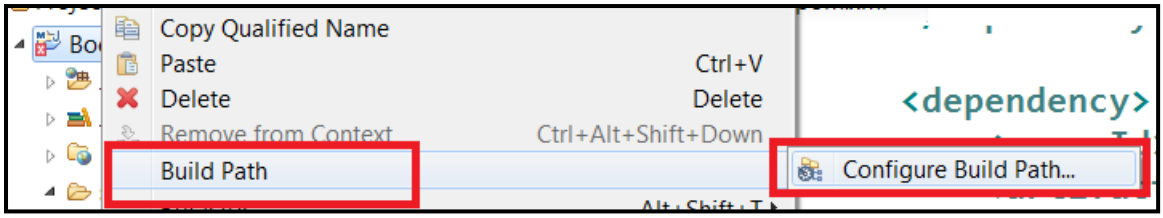
search with webapp and choose "maven-archtype-webapp" , click next



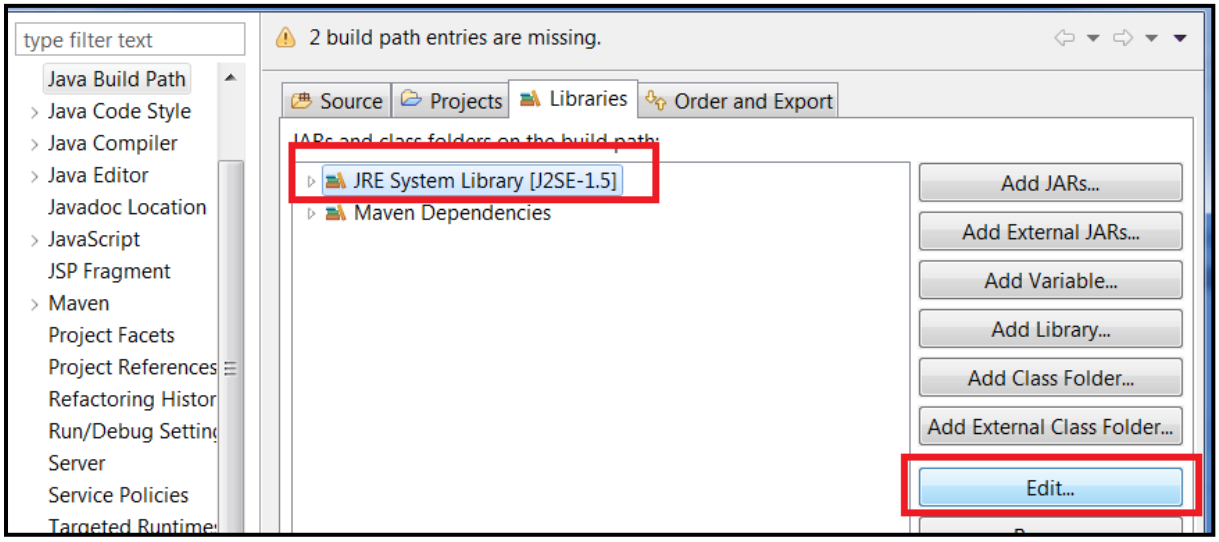
Enter Details in group Id, Artifact Id and version then click finish.



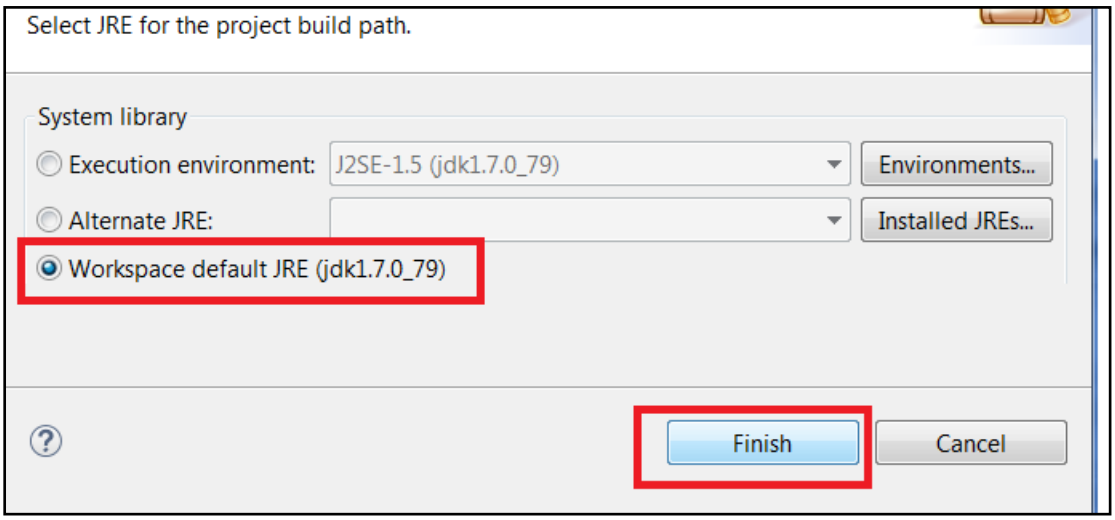
Update JDK Details to project: Right click on project > build path > configure Build path



choose JRE System Library and click on edit



choose work space or any other updated JRE click on finish

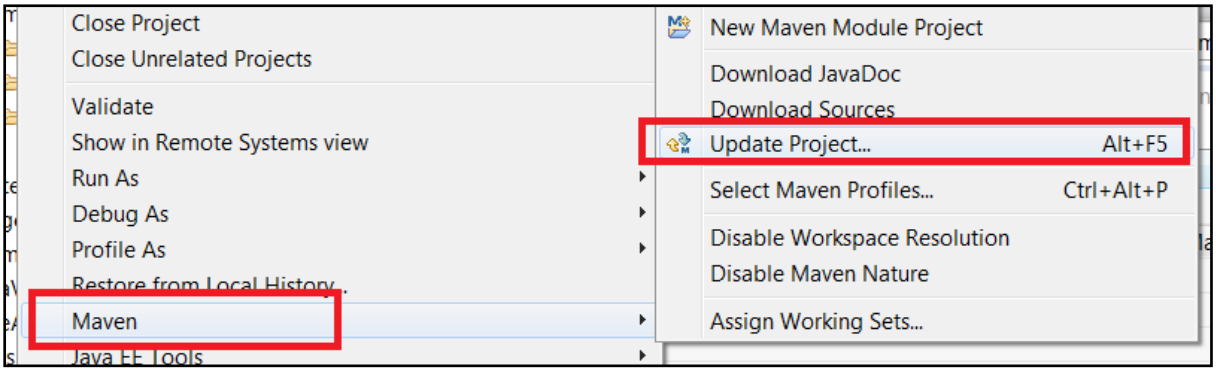


add details in pom.xml under <project> tag (complete code)

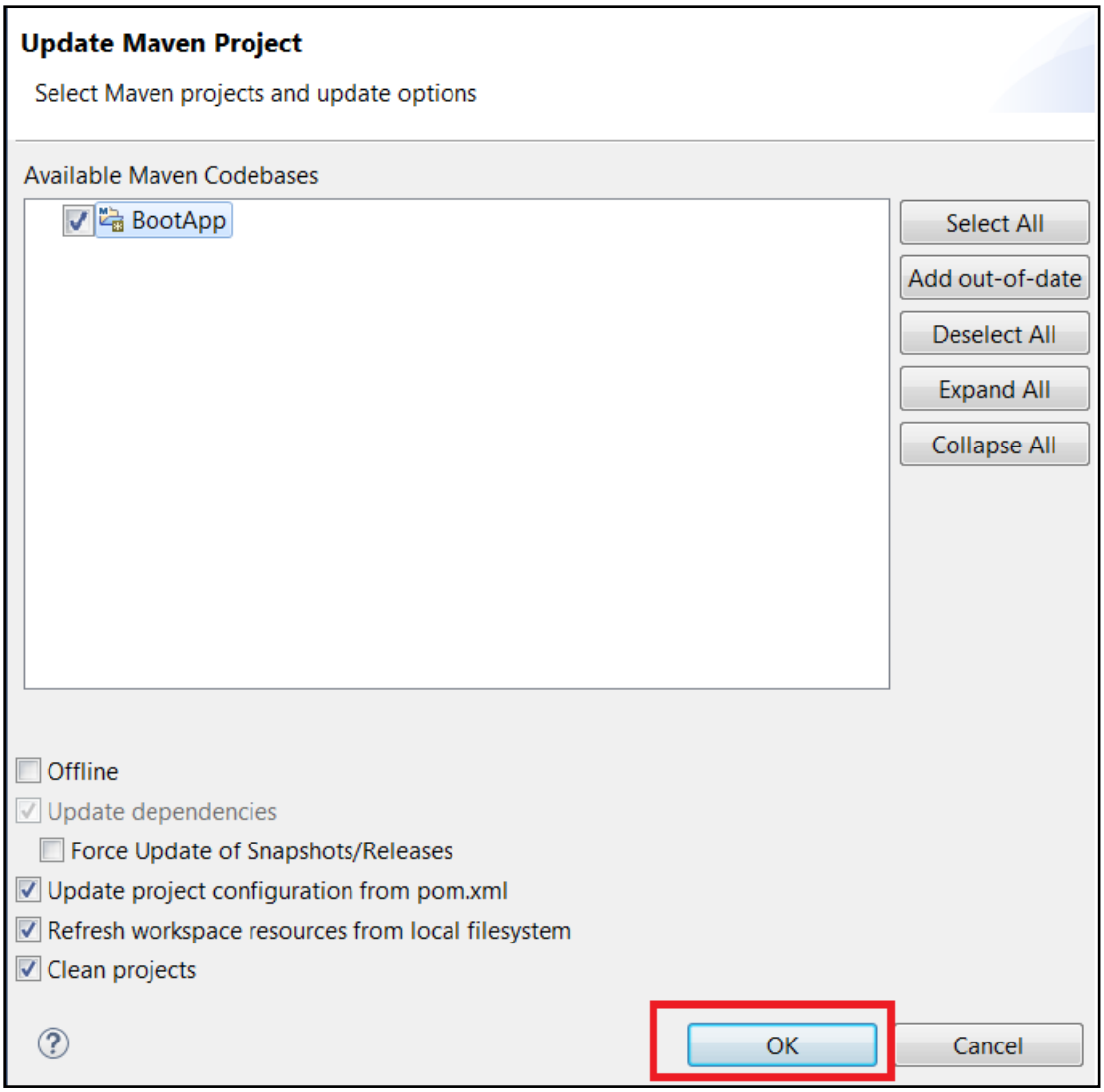
```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>sathyatech</groupId>
  <artifactId>BootApp</artifactId>
  <packaging>war</packaging>
  <version>1.0</version>
  <name>BootApp Maven Webapp</name>
  <url>http://maven.apache.org</url>
  <parent>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-parent</artifactId>
            <version>1.5.4.RELEASE</version>
            <relativePath/> <!-- lookup parent from repository -->
      </parent>

      <properties>
            <java.version>1.7</java.version>
      </properties>

      <dependencies>
            <dependency>
                  <groupId>org.springframework.boot</groupId>
                  <artifactId>spring-boot-starter-web</artifactId>
            </dependency>

            <dependency>
                  <groupId>org.springframework.boot</groupId>
                  <artifactId>spring-boot-starter-test</artifactId>
                  <scope>test</scope>
            </dependency>


            <dependency>
             <groupId>org.springframework.boot</groupId>
             <artifactId>spring-boot-starter-tomcat</artifactId>
        </dependency>
        <!-- JSTL -->
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>jstl</artifactId>
        </dependency>
        <!-- To compile JSP files -->
        <dependency>
            <groupId>org.apache.tomcat.embed</groupId>
```

```
            <artifactId>tomcat-embed-jasper</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>

    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-
plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

update maven project: Right click on Project > maven > Update Project



Click on OK



create application.propeties under "src/main/resources".

```
#Server config
server.port=2018

##View Resolver
spring.mvc.view.prefix=/WEB-INF/views/
spring.mvc.view.suffix=.jsp

## Db Details
spring.datasource.driver-class=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe
spring.datasource.username=system
spring.datasource.password=manager

##ORM Details
```

```
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.OracleDialect
spring.jpa.properties.hibernate.hbm2ddl.auto=update
```

Create one startup class under src/main/java

```
package com.app;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class DemoApplication {

	public static void main(String[] args) {
		SpringApplication.run(DemoApplication.class, args);
	}
}
```