**Target Case Study:** **Browse/ Front facing app**

**Here is the deliverable point:**

* **1) Use an ORM to abstract the DB layer from the server side code, and read data from the database. You can use myBatis or another framework of your choosing.**
* **[Delivered – Yes, Used myBatis]**
* **2) Write JUnit test cases to verify you can read and write from the database tables.**
* **[Delivered – Yes, Used jUnit]**
* **3) Create a service class that will return product details by ID as described above.**  **You are free to use a framework (like CXF, Axis2, etc), or just plain servlets if you**  **wish.**
* **[Delivered – Yes, Jersey]**
* **4) Write JUnit test cases to verify your service functionality.**
* **[Delivered – Yes, jUnit]**
* **5) BONUS: Add the capability to retrieve multiple products at once, or retrieve a list**  **of details by category, and write associated JUnit test cases.**
* **[Delivered – Yes, created services for all products, product by category]**
* **6) Write a short paragraph describing the structure of your code, relevant classes,**  **etc.**
* **[Delivered – Yes, Provided lots of comments on the code for easy understanding]**

**List of Deliverable items:**

1. **Complete project source code (TargetCaseStudy.zip)**
2. **Project Archive (TargetCaseStudy-0.0.1-SNAPSHOT.war)**
3. **Database Scripts (targetcasestudy.sql)**
4. **A detailed word document**

**Introduction:** This document explains Target Case Study for Browse/Front Facing app and how it was developed. I will explain in details the design, development and demo.

**Design:**

Browse/Front facing app is designed using RESTful web services and Angular JS SPA, Bootstrap CSS and HTML. First I will explain the RESTful web services and how we designed our services followed by Client/app design.

**Design RESTful web services:**

Front facing app designed as ROA (**R**esource **O**riented **A**rchitecture). First we defined our resources and then we assigned a URI to each individual resource. Each web services will return a JSON response with appropriate Response (200, 404, 403, 500).

Table 1: Resources and Descriptions

|  |  |  |
| --- | --- | --- |
| **Resource** | **Operations** | **Description** |
| ProductList | GET | To get a list of products |
| Product | GET | Return a single product by product id |
| Product | POST | A product for updating product information |
| Product Category | GET | Return a list of product by product category |

Table 2: Resource and URI mapping.

|  |  |  |
| --- | --- | --- |
| **Resource** | **Operations** | **Description** |
| ProductList | GET | http://{baseURL}/TargetCaseStudy/service/products/allproducts |
| Product | GET | http://{baseURL}/TargetCaseStudy/service/products/{productID } |
| Product Category | GET | [http://localhost:8080/TargetCaseStudy/service/products/category/{category](http://localhost:8080/TargetCaseStudy/service/products/category/%7bcategory)} |
| Product | POST | <http://localhost:8080/TargetCaseStudy/service/products/updateproduct> |

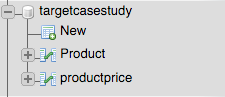
**Angular JS SPA:**

For our prototype we use angular **Single Page Architecture** (SPA). We have only one page with multiple views. Angular JS routing controls different views.

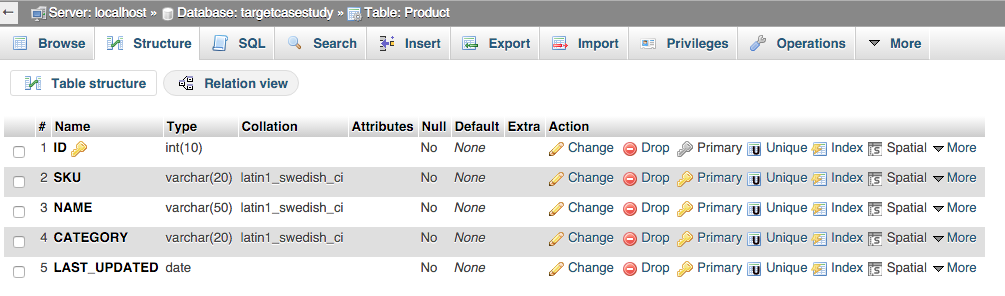
For each of the view we have different controller.

|  |  |  |
| --- | --- | --- |
| **Views/Pages** | **Controller** | **Templates** |
| ShowAllProducts | ShowProductController | show\_all\_products.html |
| ShowProductDetails/productId | ShowProductDetailsController | show\_product\_detail.html |
| ShowProductByCategory | ShowProductController | show\_products\_by\_cat.html |
| ShowProductByCategory/category | ShowProductCatController | show\_products\_category.html |
| Main Page | Single page with ng-view on it. | index.jsp |

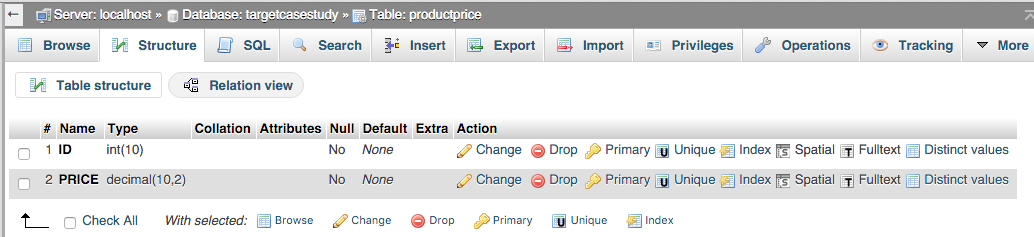
**Database design:**



**Product table:**



**Product Price table:**



**Project setup/Structure:**

**Build/deployment – Maven used to build the project.**

**Web Container – Apache Tomcat Server 7**

**Database server - MySql**

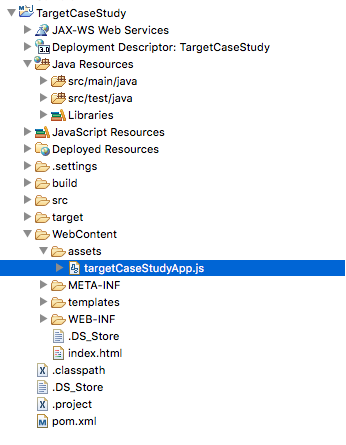
**Jersey Framework (JSR 311) – Building RESTful web services**

**myBatis - ORM**

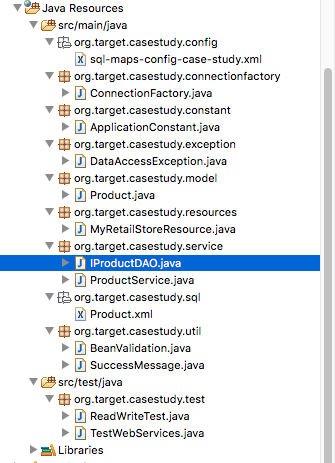
**Angular JS SPA, Angular JS routing, Bootstrap CSS, HTML**

**Optional – javax validation and hibernet validation APIs, log4j (not implemented)**

**Main Structure:**

****

**Class Structure:**

****

Above screenshot depict the package structure and relevant classes on each of the package.

In this section I will explain very high level each of the classes and their function.

**sql-maps-config-case-study.xml** – Database configuration file.

**ConnectionFactory** – This class create an instance of sqlMapClient object using the database configuration files. A static block used to make sure that it would create only one instance.

ApplicationConstant – Holds all the constants defined through out the applications.

DataAccessException – A custom exception class used to provide more customized exceptions. This class if to give an idea we can use custom exception based on the requirement though it was not asked in the case study.

Product – This is the main model class (Plain POJO) our application, it holds all the data mapped to the database using myBatis (Object Relations Mapping). This product hold all the data and once it comes to return to the client it will automatically converted into JSON object and return to the client as a JSON object.

MyRetailStoreResource – This is the core class of our app, any request comes and mapped with the path annotation to get particular information requested from client. This class has an implementation of the following methods:

**getAllProducts() –** mapped to {baseurl}/products/all product and will return a list of product from the data base as a JSON object.

**getProductById() –** mapped to {baseurl}/products/{productid} and return a JSON formatted product

**getProductByCategory() -**  mapped to {baseurl}/category/{category} return a list of JSON formatted product.

**updateProduct() -** This is the method that is being used to post a product. Mapped to {baseurl}/products/updateproduct. If provided product is not already exist in the database then it will create a new product, else it will update the product with the provided product information using the product id.

IProductDAO – interface that contains the entire method signature that is implemented in the ProductService class. This interface has all the necessary method that requires getting data from the server.

ProductService – service class that implements all the methods defined in the previous interface and communicate with data layer to get the data. First it will open a database connection and then perform related operation to the database.

Product.xml – All the stored procedures are in Product.xml.

BeanValidation – simple validation class, as RESTful web services are a big thread of security we need to implement and do all possible validation on both front end and back end. This validation class is just to show an example on the app to do another validation in the backend. We can do lot of custom validation as well as validation using the validation, and hibernate validation api.

SuccessMessage – custom class holds status and status code of response

ReadWriteTest – jUnit test cases for read/write and other validation are defined here.

TestWebServices – jUnit test cases of service validation.

**Resources and Project Configuration file:**

**targetCaseStudyApp.js –** This is the main angular js file. Where we define our app using the angular js. We also have our entire controller. Implemented a factory for getting all the data from the server. There is different method implemented to call different webservices. We also have a custom angular js filter to filter duplicate records of product category. Angular js routing is being used to route between different views.

//Method to get all the product using http get

factory.getAllProductInfo = **function**() {

**return** $http.get(baseurl + "allproducts");

}

//Method to get a specific product based on the product id using http get

factory.getProductDetails = **function**(productId) {

**return** $http.get(baseurl + productId);

}

//Method to get all the product matched with the product cateogry using http get

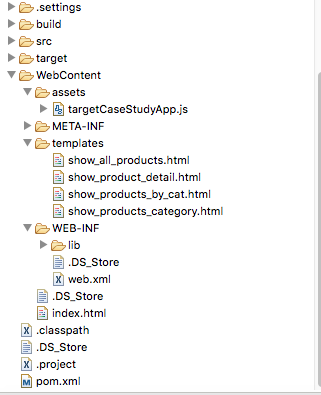
factory.getProductCategory = **function**(category) {

**return** $http.get(baseurl + "category/" +category);

}

**Following Angular js Controller is also introduced.**

1. **ShowProductCatController**
2. **ShowProductCatController**
3. **ShowProductDetailsController**

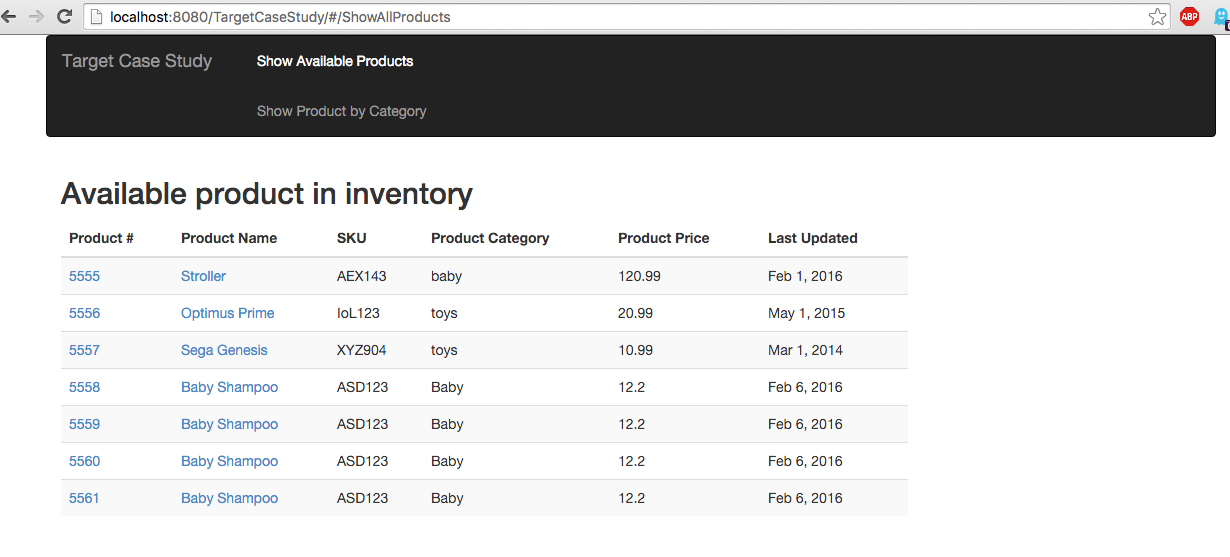


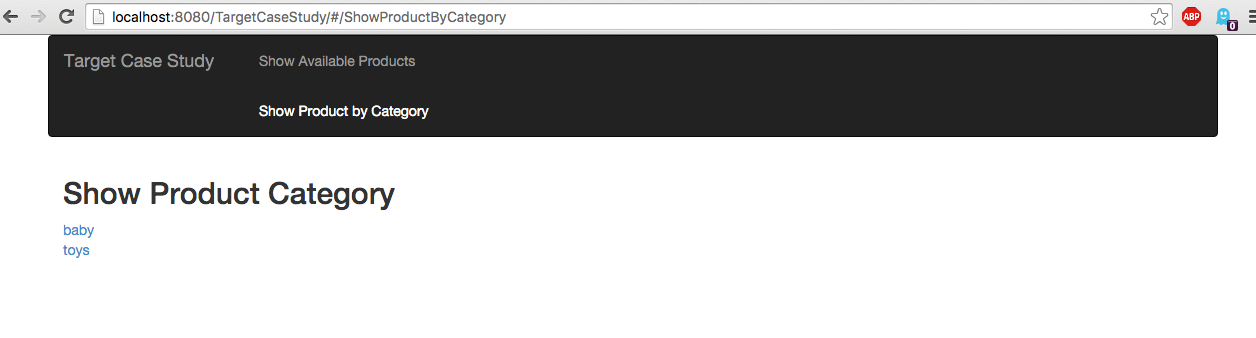
**Deployment:**

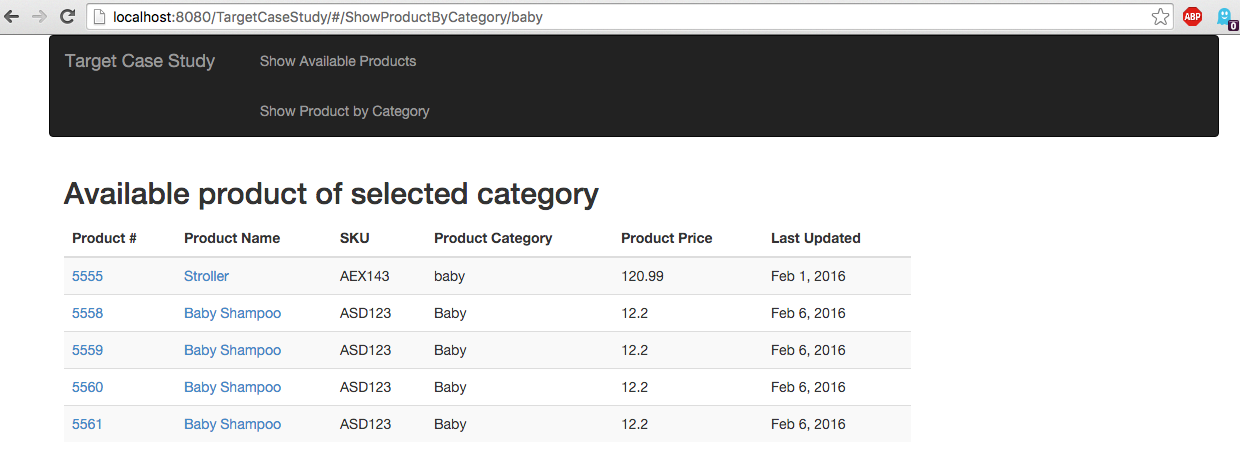
War file can be deployed in any application server. In order to deploy it in tomcat, please create a database using the provided script or manually as described above. Then deploy the war file on tomcat and start the server. Rename any javascript file from example.js.txt to example.js

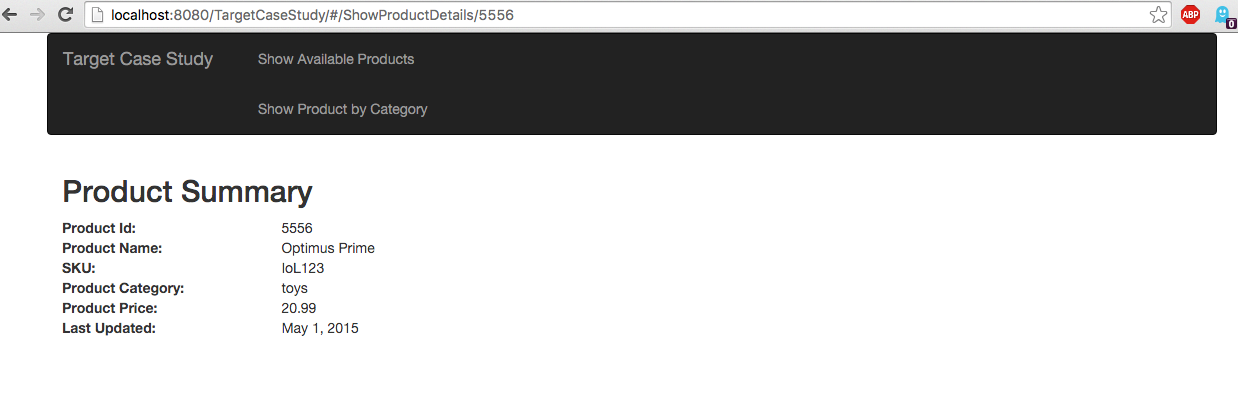
If database server (*jdbc:mysql://localhost:3306/targetcasestudy*) is different and then following file need to be modified to point to the correct data base location, “**sql-maps-config-case-study.xml**”

Demo:









**pom.xml**

<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/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<groupId>TargetCaseStudy</groupId>

<artifactId>TargetCaseStudy</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>war</packaging>

<build>

<sourceDirectory>src</sourceDirectory>

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.1</version>

<configuration>

<source>1.7</source>

<target>1.7</target>

</configuration>

</plugin>

<plugin>

<artifactId>maven-war-plugin</artifactId>

<version>2.4</version>

<configuration>

<warSourceDirectory>WebContent</warSourceDirectory>

<failOnMissingWebXml>false</failOnMissingWebXml>

</configuration>

</plugin>

</plugins>

</build>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.glassfish.jersey</groupId>

<artifactId>jersey-bom</artifactId>

<version>${jersey.version}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<dependencies>

<dependency>

<groupId>org.glassfish.jersey.containers</groupId>

<artifactId>jersey-container-servlet-core</artifactId>

</dependency>

<dependency>

<groupId>org.glassfish.jersey.media</groupId>

<artifactId>jersey-media-json-jackson</artifactId>

</dependency>

<dependency>

<groupId>org.apache.ibatis</groupId>

<artifactId>ibatis-sqlmap</artifactId>

<version>2.3.4.726</version>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.9</version>

</dependency>

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>1.2.17</version>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.11</version>

</dependency>

<dependency>

<groupId>javax.validation</groupId>

<artifactId>validation-api</artifactId>

<version>1.0.0.GA</version>

</dependency>

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-validator</artifactId>

<version>4.0.2.GA</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>1.7.14</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.14</version>

</dependency>

</dependencies>

<properties>

<jersey.version>2.3.1</jersey.version>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

</project>

**web.xml**

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns=*"http://java.sun.com/xml/ns/javaee"* xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"* id=*"WebApp\_ID"* version=*"3.0"*>

<display-name>TargetCaseStudy</display-name>

<!-- Welcome page, main page once the app loaded -->

<welcome-file-list>

<welcome-file>index.html</welcome-file>

</welcome-file-list>

<servlet>

<servlet-name>MyRetailTargetCaseStudy</servlet-name>

<servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>

<init-param>

<param-name>jersey.config.server.provider.packages</param-name>

<param-value>org.target.casestudy;org.codehaus.jackson.jaxrs</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>MyRetailTargetCaseStudy</servlet-name>

<url-pattern>/service/\*</url-pattern>

</servlet-mapping>

</web-app>

**sql-maps-config-case-study.xml**

<?xml version=*"1.0"* encoding=*"UTF-8"* ?>

<!DOCTYPE sqlMapConfig

PUBLIC "-//ibatis.apache.org//DTD SQL Map Config 2.0//EN"

"http://ibatis.apache.org/dtd/sql-map-config-2.dtd">

<sqlMapConfig>

<settings useStatementNamespaces=*"true"*/>

<!-- Database Connection information -->

<transactionManager type=*"JDBC"*>

<dataSource type=*"SIMPLE"*>

<property name=*"JDBC.Driver"* value=*"com.mysql.jdbc.Driver"*/>

<property name=*"JDBC.ConnectionURL"* value=*"jdbc:mysql://localhost:3306/targetcasestudy"*/>

<property name=*"JDBC.Username"* value=*"root"*/>

<property name=*"JDBC.Password"* value=*""*/>

</dataSource>

</transactionManager>

<!-- Product Mapper -->

<sqlMap resource=*"org/target/casestudy/sql/Product.xml"*/>

</sqlMapConfig>

**ProductMapper.xml**

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE sqlMap PUBLIC "-//ibatis.apache.org//DTD SQL Map 2.0//EN" "http://ibatis.apache.org/dtd/sql-map-2.dtd">

<sqlMap namespace=*"Product"*>

<typeAlias alias=*"PRODUCT"* type=*"org.target.casestudy.model.Product"* />

<resultMap id=*"userResultMap"* class=*"PRODUCT"*>

<result property=*"productId"* column=*"ID"* />

<result property=*"productName"* column=*"NAME"* />

<result property=*"sku"* column=*"SKU"* />

<result property=*"productCategory"* column=*"CATEGORY"* />

<result property=*"lastUpdatedDate"* column=*"LAST\_UPDATED"* />

<result property=*"price"* column=*"PRICE"* />

</resultMap>

<!-- End: Mapper.select all product -->

<select id=*"getProducts"* resultMap=*"userResultMap"*>

SELECT p.id, p.sku, p.name, p.category, p.last\_updated, price.price

FROM product p, PRODUCTPRICE price

WHERE p.id = price.id

</select>

<!-- End: Mapper.select all product -->

<!-- Start: Mapper.select product by product id -->

<select id=*"getProductById"* parameterClass=*"int"* resultMap=*"userResultMap"*>

SELECT p.id, p.sku, p.name, p.category, p.last\_updated, price.price

FROM product p, PRODUCTPRICE price

WHERE p.id = price.id

AND p.id = #value#

</select>

<!-- End: Mapper.select product by product id -->

<!-- Start: Mapper.insert into product -->

<insert id=*"insertIntoProduct"* parameterClass=*"org.target.casestudy.model.Product"*>

INSERT INTO PRODUCT

(ID, SKU, NAME, CATEGORY, LAST\_UPDATED)

VALUES ( #productId#, #sku#, #productName#, #productCategory#, #lastUpdatedDate#)

</insert>

<!-- End: Mapper.insert into product -->

<!-- Start: Mapper.insert into product Price -->

<insert id=*"insertIntoProductPrice"* parameterClass=*"org.target.casestudy.model.Product"*>

INSERT INTO PRODUCTPRICE

(ID, PRICE)

VALUES ( #productId#, #price#)

</insert>

<!-- End: Mapper.insert into product Price -->

<!-- Start: Mapper.select product by category id -->

<select id=*"getProductByCategory"* parameterClass=*"java.lang.String"* resultMap=*"userResultMap"*>

SELECT p.id, p.sku, p.name, p.category, p.last\_updated, price.price

FROM product p, PRODUCTPRICE price

WHERE p.id = price.id

AND p.category = #value#

</select>

<!-- End: Mapper.select product by category -->

<!-- Start: Mapper.update product by category -->

<update id=*"updateProduct"* parameterClass=*"org.target.casestudy.model.Product"*>

UPDATE PRODUCT

SET

SKU = #sku#,

NAME = #productName#,

CATEGORY = #productCategory#,

LAST\_UPDATED = #lastUpdatedDate#

WHERE ID = #productId#

</update>

<!-- End: Mapper.update product by category -->

<!-- Start: Mapper.update product by category -->

<update id=*"updateProductPrice"* parameterClass=*"org.target.casestudy.model.Product"*>

UPDATE PRODUCTPRICE

SET

PRICE = #price#

WHERE ID = #productId#

</update>

<!-- End: Mapper.update product by category -->

</sqlMap>

/\*\*

\*

\*/

package org.target.casestudy.connectionfactory;

import java.io.FileNotFoundException;

import java.io.IOException;

import java.io.Reader;

import org.target.casestudy.constant.ApplicationConstant;

import com.ibatis.common.resources.Resources;

import com.ibatis.sqlmap.client.SqlMapClient;

import com.ibatis.sqlmap.client.SqlMapClientBuilder;

/\*\*

\* @author Asad Islam

\* Last Updated: Feb 7, 2016

\*

\* This class is a connection manager, this will create a new sqlMapClient it not created.

\*/

public class ConnectionFactory {

private static SqlMapClient sqlMapClient;

// Loaded only when this class will be loaded first time.

static {

try {

String resource = ApplicationConstant.CONFIG\_FILE;

Reader reader = Resources.getResourceAsReader(resource);

// if sqlMapClient is null then build a client using the reader.

if (sqlMapClient == null) {

sqlMapClient = SqlMapClientBuilder.buildSqlMapClient (reader);

}

}

catch (FileNotFoundException fileNotFoundException) {

fileNotFoundException.printStackTrace();

}

catch (IOException iOException) {

iOException.printStackTrace();

}

}

/\*\*

\* This static method will return an instance of sqlMapClient,

\* and accessible directly with the class name.

\* @return sqlMapClient

\*/

public static SqlMapClient getSqlMapFactory() {

return sqlMapClient;

}

}

/\*\*

\*

\*/

package org.target.casestudy.constant;

/\*\*

\* @author Asad Islam

\* Last Updated: Feb 7, 2016

\*

\* Holds all the constant defined throughout the application.

\*/

public class ApplicationConstant {

/\*\* The constant INSERT\_INTO\_PRODUCT \*/

public static final String INSERT\_INTO\_PRODUCT = "Product.insertIntoProduct";

/\*\* The constant INSERT\_INTO\_PRODUCT\_PRICE \*/

public static final String INSERT\_INTO\_PRODUCT\_PRICE = "Product.insertIntoProductPrice";

/\*\* The constant GET\_PRODUCT\_BY\_ID \*/

public static final String GET\_PRODUCT\_BY\_ID = "Product.getProductById";

/\*\* The constant GET\_ALL\_PRODUCT \*/

public static final String GET\_ALL\_PRODUCT = "Product.getProducts";

/\*\* The constant GET\_PRODUCT\_BY\_CATEGORY \*/

public static final String GET\_PRODUCT\_BY\_CATEGORY = "Product.getProductByCategory";

/\*\* The constant UPDATE\_PRODUCT\_TABLE \*/

public static final String UPDATE\_PRODUCT\_TABLE = "Product.updateProduct";

/\*\* The constant UPDATE\_PRODUCT\_PRICE\_TABLE \*/

public static final String UPDATE\_PRODUCT\_PRICE\_TABLE = "Product.updateProductPrice";

/\*\* The constant CONFIG\_FILE \*/

public static final String CONFIG\_FILE = "org/target/casestudy/config/sql-maps-config-case-study.xml";

}

**package** org.target.casestudy.exception;

/\*\*

\* This class represents the data related unchecked exceptions.

\*

\* **@author** Asad Islam

\* Last Updated: Feb 7, 2016

\*/

**public** **class** DataAccessException **extends** RuntimeException {

/\*\*

\* Holds the value of serial version number.

\*/

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

/\*\*

\* Instantiates a new data access exception.

\*

\* **@param** code the code

\*/

**public** DataAccessException(String code) {

**super**(code);

}

/\*\*

\* Instantiates a new data access exception.

\*

\*/

**public** DataAccessException() {

**super**();

}

/\*\*

\* Instantiates a new data access exception.

\*

\* **@param** exception - Throwable Object

\*/

**public** DataAccessException(Throwable exception) {

**super**(exception);

}

/\*\*

\* Make Object non cloneable

\* **@return** Object

\* **@throws** CloneNotSupportedException

\*/

@Override

**public** **final** Object clone() **throws** java.lang.CloneNotSupportedException {

**throw** **new** java.lang.CloneNotSupportedException();

}

}

/\*\*

\*

\*/

package org.target.casestudy.model;

import java.util.Date;

import javax.validation.constraints.NotNull;

import javax.xml.bind.annotation.XmlAccessType;

import javax.xml.bind.annotation.XmlAccessorType;

import javax.xml.bind.annotation.XmlRootElement;

/\*\*

\* @author Asad Islam

\* Last Updated: Feb 7, 2016

\*/

@XmlRootElement(name="Products")

@XmlAccessorType(XmlAccessType.FIELD)

public class Product {

@NotNull

public int productId;

public String sku;

public String productName;

public String productCategory;

public Date lastUpdatedDate;

public double price;

public Product() {

}

public Product(int prodId, String prodName, String prodCat, Date ProdLastUpd, double price, String sku) {

this.productId = prodId;

this.productName = prodName;

this.productCategory = prodCat;

this.lastUpdatedDate = ProdLastUpd;

this.price = price;

this.sku = sku;

}

/\*\*

\* @return the productId

\*/

public int getProductId() {

return productId;

}

/\*\*

\* @param productId the productId to set

\*/

public void setProductId(int productId) {

this.productId = productId;

}

/\*\*

\* @return the sku

\*/

public String getSku() {

return sku;

}

/\*\*

\* @param sku the sku to set

\*/

public void setSku(String sku) {

this.sku = sku;

}

/\*\*

\* @return the productName

\*/

public String getProductName() {

return productName;

}

/\*\*

\* @param productName the productName to set

\*/

public void setProductName(String productName) {

this.productName = productName;

}

/\*\*

\* @return the productCategory

\*/

public String getProductCategory() {

return productCategory;

}

/\*\*

\* @param productCategory the productCategory to set

\*/

public void setProductCategory(String productCategory) {

this.productCategory = productCategory;

}

/\*\*

\* @return the lastUpdatedDate

\*/

public Date getLastUpdatedDate() {

return lastUpdatedDate;

}

/\*\*

\* @param lastUpdatedDate the lastUpdatedDate to set

\*/

public void setLastUpdatedDate(Date lastUpdatedDate) {

this.lastUpdatedDate = lastUpdatedDate;

}

/\*\*

\* @return the price

\*/

public double getPrice() {

return price;

}

/\*\*

\* @param price the price to set

\*/

public void setPrice(double price) {

this.price = price;

}

}

package org.target.casestudy.resources;

/\*\*

\* @author Asad Islam

\* Last Updated: Feb 7, 2016

\*

\* Four web services are defined in this class, and mapped to four different url.

\* URL: http://localhost:8080/TargetCaseStudy/service/products/allproduct mapped to allproduct resuorces

\* URL: http://localhost:8080/TargetCaseStudy/service/products/{productId}, mapped to only one product resource

\* URL: http://localhost:8080/TargetCaseStudy/service/products/category/{category}, mapped to a product list with selected category

\* URL: http://localhost:8080/TargetCaseStudy/service/products/updateproduct

\* This service is mapped to create a new resource or update an existing resource, if a resource if already present

\*/

import java.io.IOException;

import java.util.List;

import javax.validation.ConstraintViolationException;

import javax.validation.ValidationException;

import javax.ws.rs.Consumes;

import javax.ws.rs.GET;

import javax.ws.rs.POST;

import javax.ws.rs.Path;

import javax.ws.rs.PathParam;

import javax.ws.rs.Produces;

import javax.ws.rs.core.MediaType;

import javax.ws.rs.core.Response;

import javax.ws.rs.core.Response.Status;

import org.target.casestudy.model.Product;

import org.target.casestudy.service.ProductService;

import org.target.casestudy.util.BeanValidation;

//import org.target.casestudy.util.BeanValidation;

import org.target.casestudy.util.SuccessMessage;

@Path("/products")

public class MyRetailStoreResource {

// Product service

ProductService productService = new ProductService();

/\*\*

\* Mapped to path allproducts

\* @return all the product from the data base

\*/

@GET

@Path("/allproducts")

@Produces(MediaType.APPLICATION\_JSON)

public List<Product> getAllProducts()

{

return productService.getAllProduct();

}

/\*\*

\* This will return a product matched with the provided product id.

\* @param productId, path parameter as a product id

\* @return a Product

\*/

@GET

@Path("{productId}")

@Produces(MediaType.APPLICATION\_JSON)

public Product getProductById(@PathParam("productId") int productId)

{

return productService.getProductByID(productId);

}

/\*\*

\* This will return a list of product of selected category.

\* @param category, which the list should be returned.

\* @return a list of product with matching provided category.

\*/

@GET

@Path("/category/{category}")

@Produces(MediaType.APPLICATION\_JSON)

public List<Product> getProductByCategory(@PathParam("category") String category)

{

return productService.getProductByCategory(category);

}

/\*\*

\* This method will update a product based on the product id. A JSON will be posted

\* from the client and after the validation it will be updated into the data base

\* of provided product id.

\* @param product, a product as a JSON format

\* @return, SuccessMesssage with 200 status code, and OK

\* @throws ConstraintViolationException, if there is any constraint

\* @throws ValidationException, if there is any violation

\* @throws IOException, optional just in case if there is any error related to IO

\*/

@POST

@Path("updateproduct")

@Consumes(MediaType.APPLICATION\_JSON)

@Produces(MediaType.APPLICATION\_JSON)

public Response updateProduct(Product product) throws ConstraintViolationException, ValidationException, IOException

{

Response response = null;

BeanValidation beanValidation = new BeanValidation();

if(beanValidation.validateBean(product)) {

boolean update = productService.updateProductDetails(product);

if(update) {

SuccessMessage successMessage = new SuccessMessage();

successMessage.setSuccessCode(Status.OK.getStatusCode());

successMessage.setSuccessMessage(Status.OK.toString());

response = Response.status(200).entity(successMessage).build();

}

}

return response;

}

}

/\*\*

\*

\*/

package org.target.casestudy.service;

import java.util.List;

import org.target.casestudy.model.Product;

/\*\*

\* @author Asad Islam

\* Last Updated: Feb 7, 2016

\*/

public interface IProductDAO {

public List<Product> getAllProduct(); // Get a list of products

public Product getProductByID(int productId); // get a product by id

public List<Product> getProductByCategory(String Category); //get a list of product by category

public boolean insertIntoProduct(Product product); // insert a product into product and productprice table

public boolean updateProductDetails(Product product); // update a product based on the provided product id.

}

package org.target.casestudy.service;

import java.sql.SQLException;

import java.util.List;

import org.target.casestudy.connectionfactory.ConnectionFactory;

import org.target.casestudy.constant.ApplicationConstant;

import org.target.casestudy.exception.DataAccessException;

import org.target.casestudy.model.Product;

import com.ibatis.sqlmap.client.SqlMapClient;

/\*\*

\*

\* @author Asad Islam

\* Last Updated: Feb 7, 2016

\* TODO: Logger

\*/

@SuppressWarnings("unchecked")

public class ProductService implements IProductDAO {

SqlMapClient sqlMapClient = ConnectionFactory.getSqlMapFactory(); // Create db connection

/\*\*

\* This method will get all the product from database.

\* and return a list of products from the database.

\*/

public List<Product> getAllProduct() {

List<Product> products = null;

try {

products = sqlMapClient.queryForList(ApplicationConstant.GET\_ALL\_PRODUCT);

} catch (SQLException exception) {

System.out.println(this.getClass().getPackage() + " " + this.getClass().getName() + " " + exception);

throw new DataAccessException(exception.toString());

}

return products;

}

/\*\*

\* This method will return a product match by provided product id.

\*/

@Override

public Product getProductByID(int productId) {

Product product = null;

try {

product = (Product) sqlMapClient.queryForObject(ApplicationConstant.GET\_PRODUCT\_BY\_ID, productId);

} catch (SQLException exception) {

System.out.println(this.getClass().getPackage() + " " + this.getClass().getName() + " " + exception);

throw new DataAccessException(exception.toString());

}

return product;

}

/\*\*

\* This method return a list of product matched by category.

\*/

@Override

public List<Product> getProductByCategory(String Category) {

List<Product> products = null;

try {

products = sqlMapClient.queryForList(ApplicationConstant.GET\_PRODUCT\_BY\_CATEGORY, Category);

} catch (SQLException exception) {

System.out.println(this.getClass().getPackage() + " " + this.getClass().getName() + " " + exception);

throw new DataAccessException(exception.toString());

}

return products;

}

/\*\*

\* This method will insert a new product into database (Product and productprice table)

\*/

@Override

public boolean insertIntoProduct(Product product) {

boolean inserted = true;

try {

sqlMapClient.startTransaction();

sqlMapClient.insert(ApplicationConstant.INSERT\_INTO\_PRODUCT, product);

sqlMapClient.insert(ApplicationConstant.INSERT\_INTO\_PRODUCT\_PRICE, product);

sqlMapClient.commitTransaction();

} catch (SQLException sqlException) {

inserted = false;

throw new DataAccessException(sqlException);

} finally {

try {

sqlMapClient.endTransaction();

} catch (SQLException exception) {

inserted = false;

}

}

return inserted;

}

/\*\*

\* This will update a product matched by product id.

\*/

@Override

public boolean updateProductDetails(Product product) {

boolean updated = true;

try {

sqlMapClient.startTransaction();

sqlMapClient.update(ApplicationConstant.UPDATE\_PRODUCT\_TABLE, product);

sqlMapClient.update(ApplicationConstant.UPDATE\_PRODUCT\_PRICE\_TABLE, product);

sqlMapClient.commitTransaction();

} catch (SQLException sqlException) {

updated = false;

throw new DataAccessException(sqlException);

} finally {

try {

sqlMapClient.endTransaction();

} catch (SQLException exception) {

updated = false;

}

}

return updated;

}

}

/\*\*

\*

\*/

package org.target.casestudy.util;

import java.util.HashSet;

import java.util.Set;

import javax.validation.ConstraintViolation;

import javax.validation.ConstraintViolationException;

import javax.validation.Validation;

import javax.validation.ValidationException;

import javax.validation.ValidatorFactory;

import javax.validation.Validator;

import org.target.casestudy.model.Product;

/\*\*

\* @author Asad Islam

\* Last Updated: Feb 7, 2016

\*/

public class BeanValidation {

boolean isValid = false;

/\*\*

\* This method will create a validatorFactory using validation.builddefaultvalidatorfactory.

\* and check if then provided bean class has any exceptions. If there is any exceptions present

\* then put into set and then retun as inValid.

\* @param product

\* @return either true or false.

\* @throws ConstraintViolationException

\* @throws ValidationException

\*/

public boolean validateBean(Product product) throws ConstraintViolationException, ValidationException {

ValidatorFactory factory = Validation.buildDefaultValidatorFactory();

Validator validator = factory.getValidator();

Set<ConstraintViolation<Product>> violations = validator.validate(product);

if(!violations.isEmpty()) {

throw new ConstraintViolationException(new HashSet<ConstraintViolation<?>>(violations));

} else {

isValid = true;

}

return isValid;

}

}

/\*\*

\*

\*/

package org.target.casestudy.util;

/\*\*

\* @author Asad Islam

\* last Updated: Feb 7, 2016

\*/

public class SuccessMessage {

private int successCode;

private String successMessage;

/\*\*

\* @return the successCode

\*/

public int getSuccessCode() {

return successCode;

}

/\*\*

\* @param successCode the successCode to set

\*/

public void setSuccessCode(int successCode) {

this.successCode = successCode;

}

/\*\*

\* @return the successMessage

\*/

public String getSuccessMessage() {

return successMessage;

}

/\*\*

\* @param successMessage the successMessage to set

\*/

public void setSuccessMessage(String successMessage) {

this.successMessage = successMessage;

}

/\*\*

\* @param successCode

\* @param successMessage

\*/

public SuccessMessage(int successCode, String successMessage) {

this.successCode = successCode;

this.successMessage = successMessage;

}

/\*\*

\*

\*/

public SuccessMessage() {

// TODO Auto-generated constructor stub

}

}

package org.target.casestudy.test;

import java.sql.SQLException;

import java.util.Date;

import java.util.List;

import junit.framework.TestCase;

import org.junit.Test;

import org.target.casestudy.model.Product;

import org.target.casestudy.service.ProductService;

public class ReadWriteTest extends TestCase{

ProductService productService = new ProductService();

Product product = null;

@Test

public void readAllProducts() {

List<Product> products = productService.getAllProduct();

assertNotNull("Failure - Product is null", products);

assertTrue("Failure - More than 3 products are expected",

products.size() == 4);

assertEquals(Double.MIN\_VALUE, products.get(0).getPrice());

}

@Test

public void readProductById() {

product = productService.getProductByID(5555);

assertNotNull("Failure - Product is null", product);

assertTrue("Failure - Product id 5555 expected",

product.getProductId() == 5555);

assertTrue("Failure - Product name is not correct",

product.getProductName() == "Strolle");

}

@Test

public void writeIntoProductsTable() throws SQLException {

product = new Product(5561, "Baby Shampoo", "Baby", new Date(),

12.20, "ASD123");

boolean inserted = productService.insertIntoProduct(product);

assertTrue("Failure - Product was not inserted", inserted == true);

product = productService.getProductByID(5561);

assertNotNull("Failure - Product is null", product);

assertTrue("Failure - Product name is not correct",

product.getProductName() == "Baby Shampoo");

}

@Test

public void validProduct() {

product = productService.getProductByID(5555);

assertTrue("Failure - Product id does not match: ", product.productId == 12323);

}

@Test

public void getNullResponseWithCategory() {

String category = null;

List<Product> products = productService.getProductByCategory(category);

assertNull("Failure - Invalid product:", products.get(0));

}

@Test

public void invalidProduct() {

//Test invalid product

assertTrue("Failure - Invalid product: ",productService.getProductByID(5555) == null);

}

}

package org.target.casestudy.test;

import static org.junit.Assert.assertEquals;

import java.net.URI;

import javax.ws.rs.NotFoundException;

import javax.ws.rs.client.Client;

import javax.ws.rs.client.ClientBuilder;

import javax.ws.rs.client.WebTarget;

import javax.ws.rs.core.MediaType;

import javax.ws.rs.core.Response;

import javax.ws.rs.core.UriBuilder;

import org.glassfish.jersey.client.ClientConfig;

import org.junit.Test;

public class TestWebServices {

@Test

public void readAllproduct() {

ClientConfig config = new ClientConfig();

Client client = ClientBuilder.newClient(config);

WebTarget target = client.target(getBaseURI());

String response = target.path("products").path("allproduct").request().accept(MediaType.APPLICATION\_JSON).get(Response.class).toString();

System.out.println(response);

}

@Test

public void findAllProducts() {

ClientConfig config = new ClientConfig();

Client client = ClientBuilder.newClient(config);

WebTarget target = client.target(getBaseURI());

Response response = target.path("products").path("allproduct").request().accept(MediaType.APPLICATION\_JSON).get(Response.class);

System.out.println("JSON Results: " + response);

String jsonResponseAsString = response.readEntity(String.class);

assertEquals(200, response.getStatus());

assertEquals("{\"id\":\"5555\",\"productname\":\"Stroller\"}", jsonResponseAsString);

}

@Test

public void findProductById() throws NotFoundException {

ClientConfig config = new ClientConfig();

Client client = ClientBuilder.newClient(config);

WebTarget target = client.target(getBaseURI());

Response response = target.path("products").path("5555").request().accept(MediaType.APPLICATION\_JSON).get(Response.class);

System.out.println("JSON Results: " + response);

String jsonResponseAsString = response.readEntity(String.class);

assertEquals(200, response.getStatus());

assertEquals("{\"id\":\"5555\",\"productname\":\"Stroller\"}", jsonResponseAsString);

}

@Test

public void findProductByCategory() throws NotFoundException {

ClientConfig config = new ClientConfig();

Client client = ClientBuilder.newClient(config);

WebTarget target = client.target(getBaseURI());

Response response = target.path("products").path("category").path("baby").request().accept(MediaType.APPLICATION\_JSON).get(Response.class);

System.out.println("JSON Results: " + response);

String jsonResponseAsString = response.readEntity(String.class);

assertEquals(200, response.getStatus());

assertEquals("{\"id\":\"5555\",\"productname\":\"Stroller\"}", jsonResponseAsString);

}

private static URI getBaseURI() {

return UriBuilder.fromUri("http://localhost:8081/TargetCaseStudy/").build();

}

}

client sde code:

//Define an angular module for our app

**var** targetCaseStudy = angular.module('targetCaseStudy', []);

targetCaseStudy.config(['$routeProvider',

**function**($routeProvider) {

$routeProvider.

when('/ShowProductDetails/:productId', {

templateUrl: 'templates/show\_product\_detail.html',

controller: 'ShowProductDetailsController'

}).

when('/ShowAllProducts', {

templateUrl: 'templates/show\_all\_products.html',

controller: 'ShowProductController'

}).

when('/ShowProductByCategory', {

templateUrl: 'templates/show\_products\_by\_cat.html',

controller: 'ShowProductController'

}).

when('/ShowProductByCategory/:cat', {

templateUrl: 'templates/show\_products\_category.html',

controller: 'ShowProductCatController'

}).

otherwise({

redirectTo: '/ShowAllProducts'

});

}]);

/\*\*

\* This controller again call the factory method getProductDetails, where we this controller

\* provide the product id and call webservices to get related data for this product.

\*

\*/

targetCaseStudy.controller('ShowProductCatController', **function**($scope, $routeParams, productFactory) {

$scope.productCategoryList = {};

$scope.category = $routeParams.cat;

getProductCategory($scope.category);

**function** getProductCategory(category) {

productFactory.getProductCategory(category).success(**function**(response) {

$scope.productCategoryList = response;

}).error(**function**(error){

$scope.status = "Error Retriving data";

});

}

});

/\*\*

\* This controller again call the factory method getProductDetails, where we this controller

\* provide the product id and call webservices to get related data for this product.

\*

\*/

targetCaseStudy.controller('ShowProductDetailsController', **function**($scope, $routeParams, productFactory) {

$scope.product = {};

$scope.product\_Id = $routeParams.productId;

getProductDetails($scope.product\_Id);

**function** getProductDetails(product\_Id) {

productFactory.getProductDetails(product\_Id).success(**function**(response) {

$scope.product = response;

}).error(**function**(error){

$scope.status = "Error Retriving data";

});

}

});

/\*\*

\* ShowProductController is used to get all the product from the server using factory

\* method getAllProductInfo and then push into the productlist array, later we use this array

\* to display all the product in a table.

\*/

targetCaseStudy.controller('ShowProductController', **function**($scope, productFactory) {

$scope.productList = {};

getAllProduct();

**function** getAllProduct() {

productFactory.getAllProductInfo().success(**function**(response) {

$scope.productList = response;

}).error(**function**(error){

$scope.status = "Error Retriving data";

});

}

});

/\*\*

\* This is a factory where we produce all the data for our case study.

\* We use angular js http methods to get data using all the web services

\* available.

\* <Example>

\* http://localhost:8080/TargetCaseStudy/service/products/allproduct

\* create a factory with the above url and feed data to the appropriate controller.

\*/

targetCaseStudy.factory('productFactory', ['$http', **function**($http) {

**var** factory = {};

**var** baseurl = 'http://localhost:8080/TargetCaseStudy/service/products/';

factory.getAllProductInfo = **function**() {

**return** $http.get(baseurl + "allproducts");

}

factory.getProductDetails = **function**(productId) {

**return** $http.get(baseurl + productId);

}

factory.getProductCategory = **function**(category) {

**return** $http.get(baseurl + "category/" +category);

}

**return** factory;

}])

targetCaseStudy.filter('unique', **function**() {

**return** **function**(collection, keyname) {

**var** output = [],

keys = [];

angular.forEach(collection, **function**(item) {

**var** key = item[keyname].toLowerCase();

**if**(keys.indexOf(key) === -1) {

keys.push(key);

output.push(item);

}

});

**return** output;

};

});

<h2>Available product in inventory</h2>

<table class=*"table table-striped"*>

<thead>

<tr>

<th>Product #</th>

<th>Product Name</th>

<th>SKU</th>

<th>Product Category</th>

<th>Product Price</th>

<th>Last Updated</th>

</tr>

</thead>

<tbody>

<tr data-ng-repeat=*"product in productList"*>

<td><a href=*"#ShowProductDetails/{{product.productId}}"*>{{product.productId}}</a></td>

<td><a href=*"#ShowProductDetails/{{product.productId}}"*>{{product.productName}}</a></td>

<td>{{product.sku}}</td>

<td>{{product.productCategory}}</td>

<td>{{product.price}}</td>

<td>{{product.lastUpdatedDate | date}}</td>

</tr>

</tbody>

</table>

<h2>Product Summary</h2>

<div class=*"row"*>

<div class=*"col-md-3"*>

<strong>Product Id: </strong>

</div>

<div class=*"col-md-3"*>{{product.productId}}</div>

</div>

<div class=*"row"*>

<div class=*"col-md-3"*>

<strong>Product Name: </strong>

</div>

<div class=*"col-md-3"*>{{product.productName}}</div>

</div>

<div class=*"row"*>

<div class=*"col-md-3"*>

<strong>SKU: </strong>

</div>

<div class=*"col-md-3"*>{{product.sku}}</div>

</div>

<div class=*"row"*>

<div class=*"col-md-3"*>

<strong>Product Category: </strong>

</div>

<div class=*"col-md-3"*>{{product.productCategory}}</div>

</div>

<div class=*"row"*>

<div class=*"col-md-3"*>

<strong>Product Price: </strong>

</div>

<div class=*"col-md-3"*>{{product.price}}</div>

</div>

<div class=*"row"*>

<div class=*"col-md-3"*>

<strong>Last Updated: </strong>

</div>

<div class=*"col-md-3"*>{{product.lastUpdatedDate | date}}</div>

</div>

<h2>Show Product Category</h2>

<div data-ng-repeat=*"product in productList | unique:'productCategory'"*>

<div><a href=*"#ShowProductByCategory/{{product.productCategory}}"*>{{product.productCategory}}</a></div>

</div>

<h2>Available product of selected category</h2>

<table class=*"table table-striped"*>

<thead>

<tr>

<th>Product #</th>

<th>Product Name</th>

<th>SKU</th>

<th>Product Category</th>

<th>Product Price</th>

<th>Last Updated</th>

</tr>

</thead>

<tbody>

<tr data-ng-repeat=*"product in productCategoryList"*>

<td><a href=*"#ShowProductDetails/{{product.productId}}"*>{{product.productId}}</a></td>

<td><a href=*"#ShowProductDetails/{{product.productId}}"*>{{product.productName}}</a></td>

<td>{{product.sku}}</td>

<td>{{product.productCategory}}</td>

<td>{{product.price}}</td>

<td>{{product.lastUpdatedDate | date}}</td>

</tr>

</tbody>

</table>

<!DOCTYPE html>

<html lang=*"en"*>

<head>

<title>Target Case Study</title>

<meta name=*"viewport"* content=*"width=device-width, initial-scale=1"*>

<link

href=*"http://netdna.bootstrapcdn.com/bootstrap/3.0.0/css/bootstrap.min.css"*

rel=*"stylesheet"*>

</head>

<body ng-app=*"targetCaseStudy"*>

<div class=*"container"*>

<div class=*"row"*>

<nav class=*"navbar navbar-inverse"*>

<div class=*"container-fluid"*>

<div class=*"navbar-header"*>

<a class=*"navbar-brand"* href=*"#"*>Target Case Study</a>

</div>

<div class=*"col-md-3"*>

<ul class=*"nav navbar-nav"*>

<li><a href=*"#ShowAllProducts"*> Show Available Products

</a></li>

<li><a href=*"#ShowProductByCategory"*> Show Product by

Category </a></li>

</ul>

</div>

</div>

</nav>

<div class=*"col-md-9"*>

<div ng-view></div>

</div>

</div>

</div>

<script

src=*"http://ajax.googleapis.com/ajax/libs/angularjs/1.0.7/angular.min.js"*></script>

<script src=*"assets/targetCaseStudyApp.js"*></script>

</body>

</html>