SPRING MVC PROJECT WITH HIBERNATE

AN ONLINE PHARMACY WEBSITE.

NEU ID: 001819034

NAME: DINESH MARIA ANTONY PACKIANATHAN JEROME

SUMMARY OF PROJECT:

The Project is an Online portal for Ordering Medicines online and delivering it to the patients, this portal has been written in a generalized way where in it can be even used as a normal shopping website as well. The functions are written on a generalized way to promote flexibility and it can be modified according to needs. This project, I have made sure I have concentrated towards Ordering medicines online. The Medicine portal helps in bringing in a network between suppliers and customers, the customers here are vulnerable patients who suffer from illness of various sorts. This website has sorting categories according to the needs and their names.

SUMMARY OF FUNCTIONALITY PERFORMED:

This website has various functionality like Shopping cart, Secure Login along with Email authentication for additional security we also have Programmatic and declarative security. The other core functionality include site categorizes the products into categories and displays it according to filters used. The Restful services provide secure session transactions & Representational State Transfer (REST). This is an API or to say an Architectural style.

We have used Email Authentication to validate the users and then save their details in USER table similarly we have a set of validations to be done, these validations are done with the products that are sold, also we have some review functionalities that can be added to individual products. These reviews help customers in buying the product. We have also included SQL filters before saving them in database. This Project uses Hibernate generates SQL queries to have all the data related management.

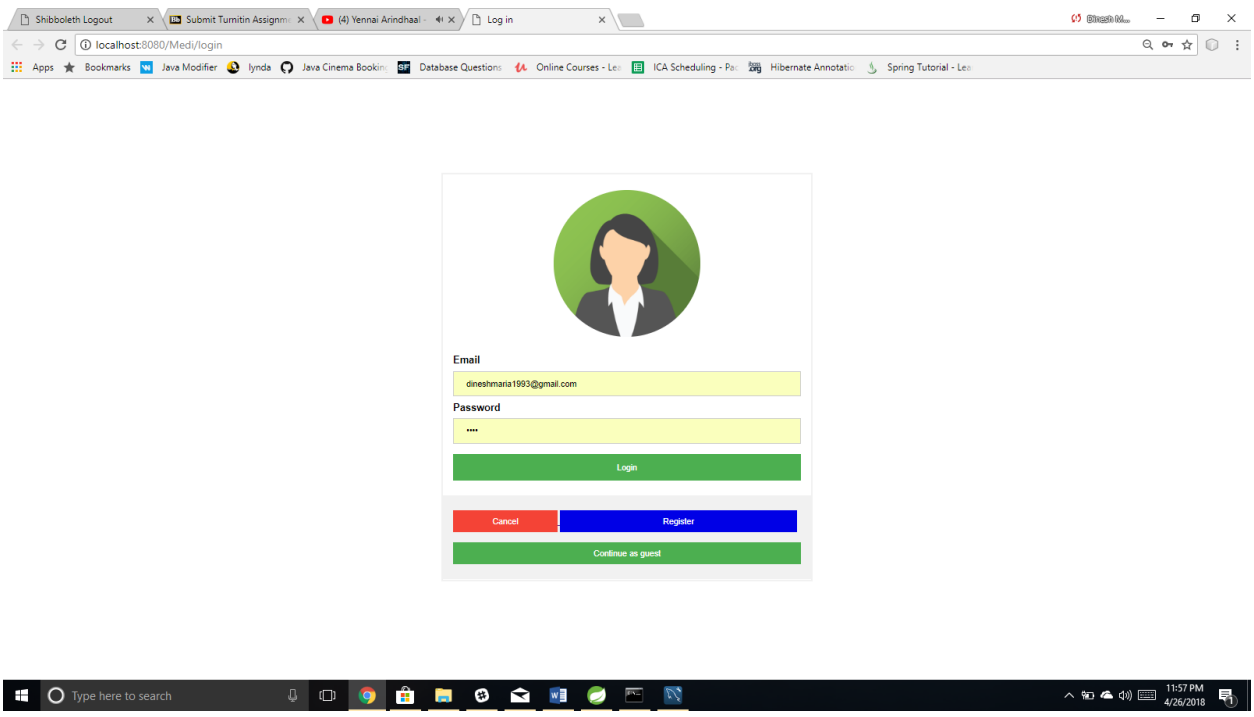
SUMMARY OF TECHNOLOGIES USED:

- Java
- Jsp
- Spring
- Hibernate
- Sql
- Restful Services.
- Front-end HTML, CSS and Javascript.

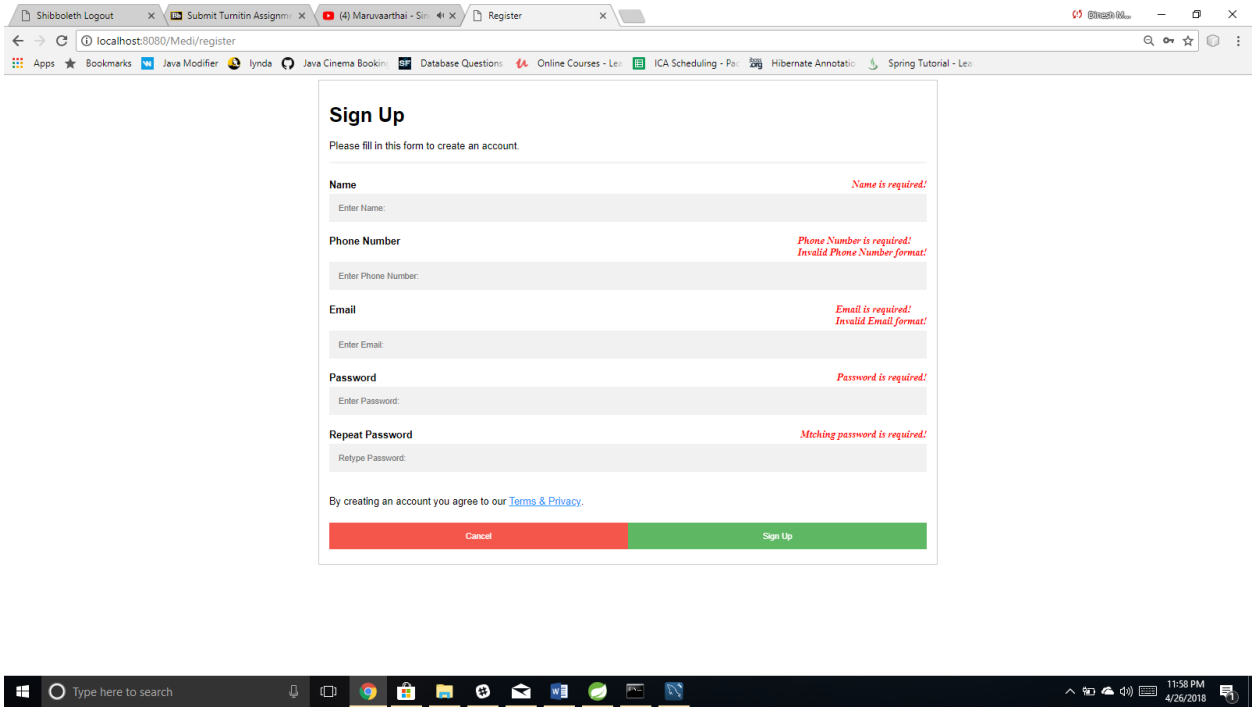
Brief list and description of the roles, and tasks each role could perform

- **Supplier:** The supplier will be able to upload products and images related to products, He or she can upload excel files and export them this will enable them to upload a set of big product list in case of multiple set of products.
- **Customer:** The customer can order products and checkout, he or she can also give reviews accordingly to individual products.

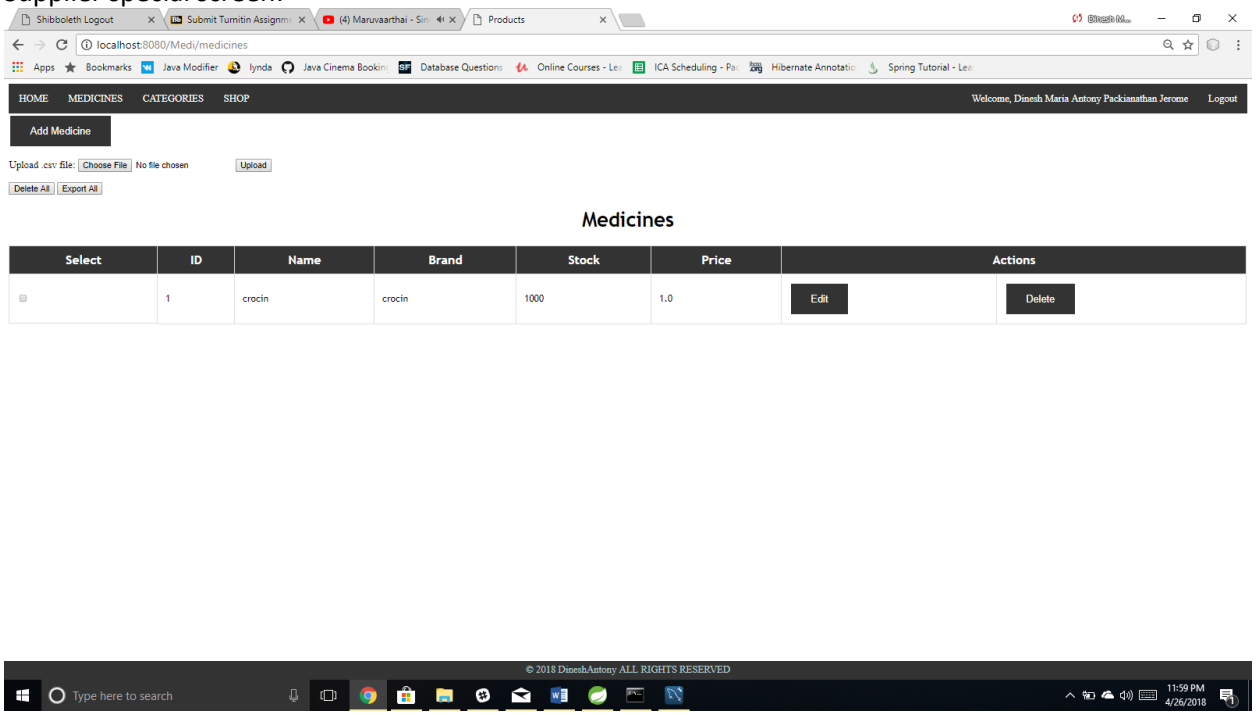
SCREENSHOTS:Supplier

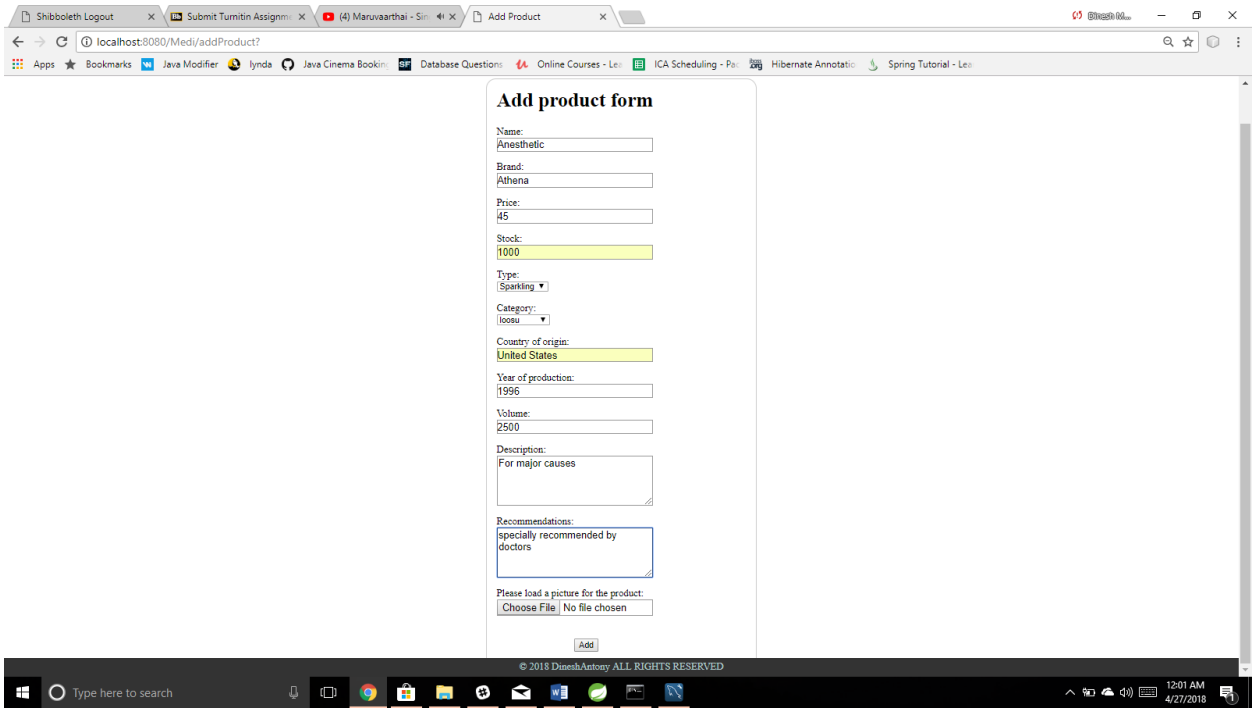


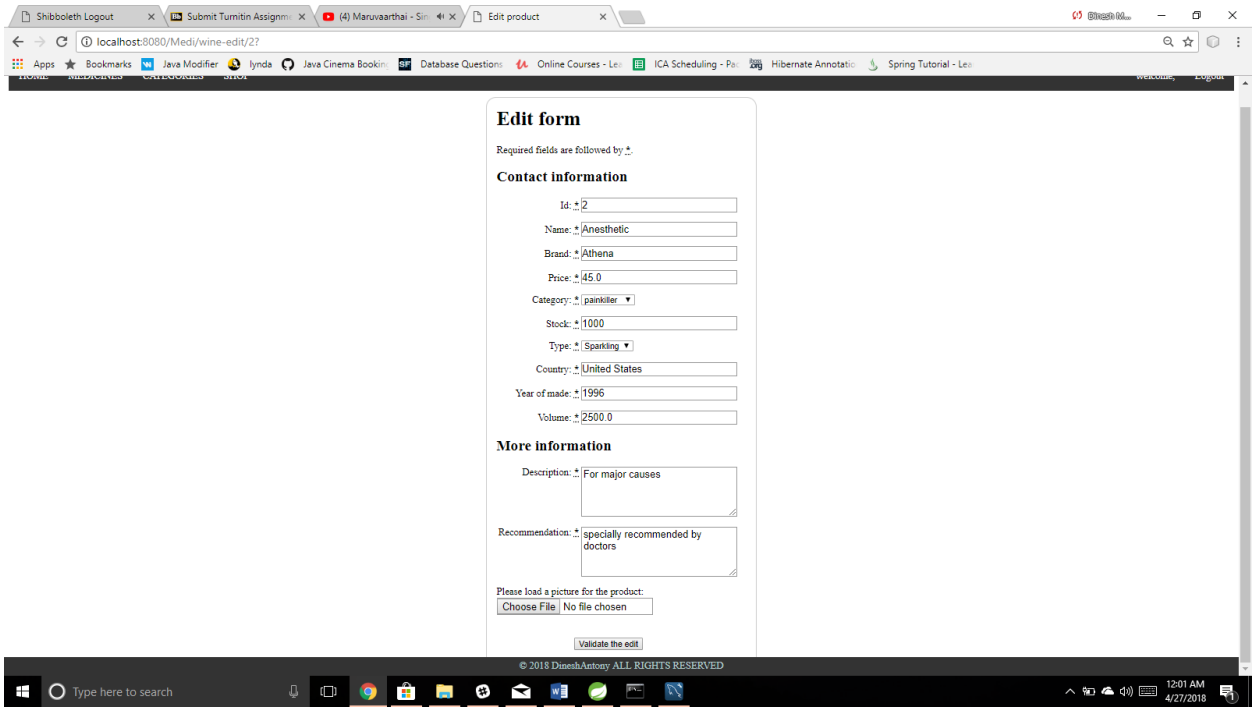
Validations:



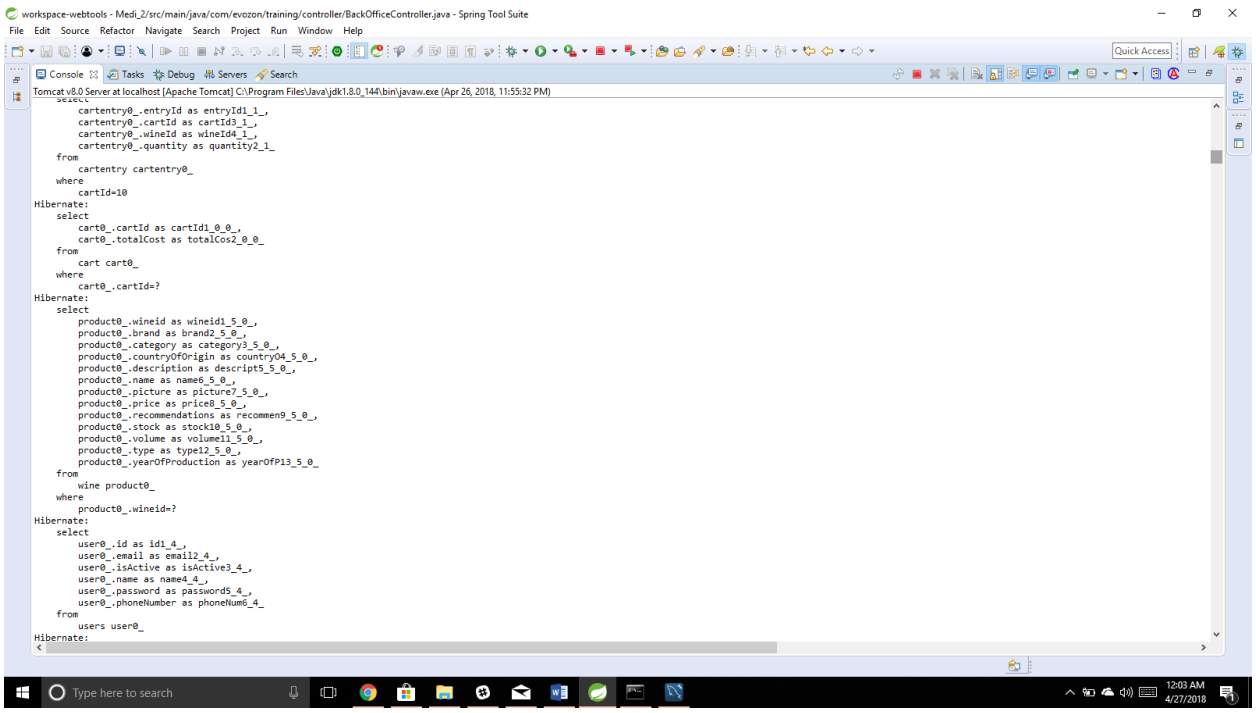
Supplier special screen:



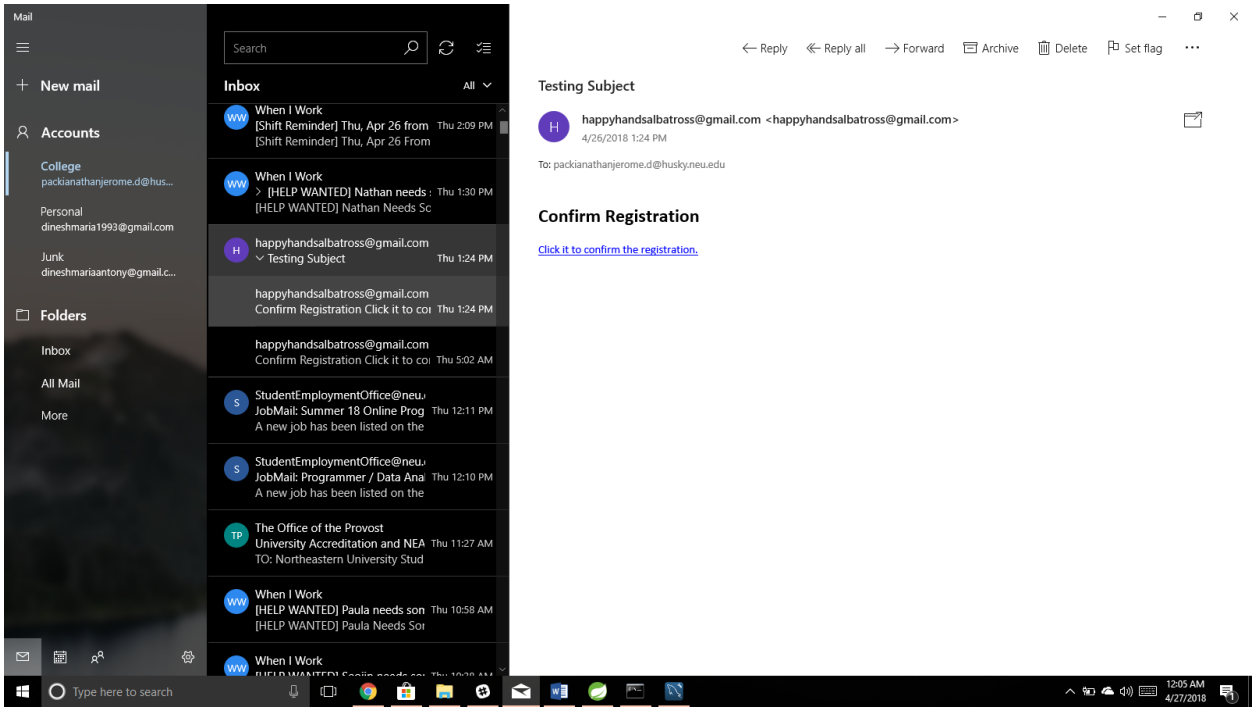



Shibboleth Logout
Submit Turnitin Assignme
(4) Maruvaarthai - Sin
Edit product
localhost:8080/Medi/wine-edit/2?
Edit form
Required fields are followed by *.
Contact information
Id: * 2
Name: * Anesthetic
Brand: * Athena
Price: * 45.0
Category: * painkiller
Stock: * 1000
Type: * Sparkling
Country: * United States
Year of made: * 1996
Volume: * 2500.0
More information
Description: * For major causes
Recommendation: * specially recommended by doctors
Please load a picture for the product:
Choose File No file chosen
Validate the edit
© 2018 DineshAntony ALL RIGHTS RESERVED
Type here to search
12:01 AM
4/27/2018

Console:


workspace-webtools - Medi_2/src/main/java/com/evozon/training/controller/BackOfficeController.java - Spring Tool Suite
File Edit Source Refactor Navigate Search Project Run Window Help
Console
Tasks
Debug
Servers
Search
Tomcat v8.0 Server at localhost [Apache Tomcat] C:\Program Files\Java\jdk1.8.0_144\bin\javaw.exe (Apr 26, 2018, 11:55:32 PM)
cartentry0_.entryId as entryId1_1_,
cartentry0_.cartId as cartId3_1_,
cartentry0_.wineId as wineId4_1_,
cartentry0_.quantity as quantity2_1_
from
cartentry cartentry0_
where
cartId=10
Hibernate:
select
cart0_.cartId as cartId1_0_0_,
cart0_.totalCost as totalCos2_0_0_
from
cart cart0_
where
cart0_.cartId=?
Hibernate:
select
product0_.wineid as wineid1_5_0_,
product0_.brand as brand2_5_0_,
product0_.category as category3_5_0_,
product0_.countryOfOrigin as countryO4_5_0_,
product0_.description as descript5_5_0_,
product0_.name as name6_5_0_,
product0_.picture as picture7_5_0_,
product0_.price as price8_5_0_,
product0_.recommendations as recommen9_5_0_,
product0_.stock as stock10_5_0_,
product0_.volume as volume11_5_0_,
product0_.type as type12_5_0_,
product0_.yearOfProduction as yearOfP13_5_0_
from
wine product0_
where
product0_.wineid=?
Hibernate:
select
user0_.id as id1_4_,
user0_.email as email2_4_,
user0_.isActive as isActive3_4_,
user0_.name as name4_4_,
user0_.password as password5_4_,
user0_.phoneNumber as phoneNum6_4_
from
users user0_
Hibernate:
Type here to search
12:03 AM
4/27/2018

Email Authentication:



APPENDIX

Controllers:

```
package com.me.project.controller;

import com.me.project.model.Category;

import com.me.project.model.Product;

import com.me.project.model.ProductDAO;

import com.me.project.model.users.UserDAO;

import com.me.project.model.users.UserFormValidators;

import com.me.project.service.CategoriesService;

import com.me.project.service.FileSystemStorageService;

import com.me.project.service.Prod_services;

import com.me.project.service.Medi_user_service;

import com.me.project.service.convertors.DAOConverter;
```

```
import com.me.project.service.validatorss.ProducTotallidators;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.core.io.FileSystemResource;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.validation.BindingResult;

import org.springframework.validation.annotation.Validated;

import org.springframework.web.bind.WebDataBinder;

import org.springframework.web.bind.annotation.*;

import org.springframework.web.multipart.MultipartFile;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import java.io.ByteArrayInputStream;

import java.io.File;

import java.io.IOException;

import java.net.URISyntaxException;

import java.util.HashMap;

import java.util.List;

import java.util.Locale;

import java.util.Optional;

@Controller

public class BackOfficeController extends UserController {
```

```
    private static final Logger logger = LoggerFactory.getLogger(BackOfficeController.class);

    @Autowired

    private Prod_services product_Service;

    @Autowired

    private Medi_user_service medi_user_service;

    @Autowired

    private CategoriesService categoriesService;

    @Autowired

    private FileSystemStorageService fileStorage;

    @Autowired

    private ProducTotallidators producTotallidators;

    @Autowired

    private DAOConverter converter;

    @Autowired

    private UserFormValidators userFormValidators;


    @InitBinder(value = "productDAO")

    private void initBinder(WebDataBinder binder) {

        binder.seTotallidators(producTotallidators);

    }
@RequestMapping(value = "/home", method = RequestMethod.GET)

    public String home(Locale locale, Model model) {

        logger.info("Home page. The client locale is {}", locale);

        String authenticatedUserName = medi_user_service.getAuthenticatedUser().getName();

        model.addAttribute("userName", authenticatedUserName);

        return "home";
```

```
}

@RequestMapping(value = "/addProduct", method = RequestMethod.GET)
public String showAddView(Locale locale, Model model) {
    List<Category> categories = categoriesService.getAllTheCategories();
    model.addAttribute("categories", categories);
    logger.info("Add product. The client locale is {}", locale);
    model.addAttribute("productDAO", new ProductDAO());

    String authenticatedUserName = medi_user_service.getAuthenticatedUser().getName();
    model.addAttribute("userName", authenticatedUserName);

    return "add-product";
}

@RequestMapping(value = "/addProduct", method = RequestMethod.POST)
public String addProduct(Locale locale, Model model,
                    @ModelAttribute("productDAO") @Validated ProductDAO productDAO,
                    BindingResult bindingResult,
                    @RequestParam String category
                    //@RequestParam(value = "picture", required = false) MultipartFile file
) {
    if (bindingResult.hasErrors()) {
        logger.info("Input validation failed!");
        List<Category> categories = categoriesService.getAllTheCategories();
```

```
        model.addAttribute("categories", categories);

        String authenticatedUserName = medi_user_service.getAuthenticatedUser().getName();
        model.addAttribute("userName", authenticatedUserName);

        return "add-product";
    } else {
        List<Category> categories = categoriesService.getAllTheCategories();
        for (Category c : categories) {
            if (c.getName().equals(category))
                productDAO.setCategoryID(c.getID());
        }
        Product product = converter.convertDAOToProduct(productDAO);
        boolean wasAdded = true;
        product_Service.addProduct(product);
        logger.info("Product added. The client locale is {}", locale);

        return "redirect:medicines";
    }
}

@RequestMapping(value = "/medicines", method = RequestMethod.GET)
public String getProducts(Locale locale, Model model,HttpServletRequest request) {
    logger.info("List of medis. The client locale is {}", locale);
    model.addAttribute("mediList", product_Service.getAllProducts());
```

```
    String authenticatedUserName = medi_user_service.getAuthenticatedUser().getName();


    model.addAttribute("userName", authenticatedUserName);
    return "medicines";
}


@RequestMapping(value = "/importProducts", method = RequestMethod.POST)
public String importProducts(@RequestParam("file") MultipartFile file, Model model) {
    int nb = 0;
    try {
        ByteArrayInputStream stream = new ByteArrayInputStream(file.getBytes());
        String myString = org.apache.commons.io.IOUtils.toString(stream, "UTF-8");
        nb = product_Service.manageImports(myString);
    } catch (IOException e) {
        logger.error("GET BYTES ERROR", e);
    }
    model.addAttribute("nbOfProducts", nb);
    return "import-message";
}



@RequestMapping(value = "/medi-edit", method = RequestMethod.POST)
public String productHome(@ModelAttribute("productDAO") @Validated ProductDAO productDAO,
                          BindingResult bindingResult,
                          Model model,
```

```
                    @RequestParam(value = "picture", required = false) MultipartFile file) {
    if (bindingResult.hasErrors()) {
        logger.info("Input validation failed!");
        return "medi-edit";
    } else {
        Product product = converter.convertDAOToProduct(productDAO);
        product_Service.updateProduct(product);
        logger.info("You edit the dates successfully!");
        return "redirect:medicines";
    }
}

@RequestMapping(value = "/medi-edit/{id}", method = RequestMethod.GET)
public String productEdit(Model model, @PathVariable Integer id) {
    logger.info("You are finally in edit mode!");
    ProductDAO productDAO = converter.convertProductToDAO(product_Service.getProduct(id));
    model.addAttribute("productDAO", productDAO);

    List<Category>categories = categoriesService.getAllTheCategories();
    model.addAttribute("categories", categories);

    return "medi-edit";
}
```

```
@RequestMapping(value = "/delete", method = RequestMethod.POST)
public String deleteProduct(@RequestParam("deleteId") String id) {
    logger.info(id);
    product_Service.deleteProduct(id);
    return "redirect:medicines";
}

@RequestMapping(value = "/categories", method = RequestMethod.GET)
public String categories(Model model){
    HashMap<String, Integer> categoryQuantityMap = categoriesService.getAllTheCategoriesMap();
    model.addAttribute("Categories", categoryQuantityMap);

    String authenticatedUserName = medi_user_service.getAuthenticatedUser().getName();
    model.addAttribute("userName", authenticatedUserName);
    return "categories";
}

@RequestMapping(value = "/categories", method = RequestMethod.POST)
public String categories(Model model, @RequestParam String categoryName) {
    HashMap<String, Integer> categoryQuantityMap;
    Category category = new Category();

    category.setName(categoryName);

    if (categoriesService.addCategory(category))
```

```
        model.addAttribute("success", Boolean.TRUE);
    else
        model.addAttribute("success", Boolean.FALSE);

    categoryQuantityMap = categoriesService.getAllTheCategoriesMap();
    model.addAttribute("Categories", categoryQuantityMap);

    String authenticatedUserName = medi_user_service.getAuthenticatedUser().getName();
    model.addAttribute("userName", authenticatedUserName);
    return "categories";
}
@RequestMapping(value = "/login", method = RequestMethod.GET)
public String login() {
    return "login-form";
}
@RequestMapping(value = "/deleteAll", method = RequestMethod.POST)
public String deleteAll(@RequestParam(value = "getAll", defaulTotallue = "null") String id) {
    if (!"null".equals(id)) {
        product_Service.deleteAll(id);
    }
    return "redirect:medicines";
}

@RequestMapping(value = "/exportAllSuccess", method = RequestMethod.GET)
@ResponseBody
```

```
public FileSystemResource exportProductsSuccess(HttpServletRequest request) throws URISyntaxException
   {

    String ids=(String)request.getSession().getAttribute("ids");

    HttpServletResponse response=(HttpServletResponse) request.getSession().getAttribute("response");

    File file=product_Service.exportProducts(ids);

    response.setContentLength((int)file.getName().length());

    response.setContentType("application/force-download");

    response.setHeader("Content-Disposition", "attachment; filename=\"" + file.getName() + "\"");

    return new FileSystemResource(file);

}

  @RequestMapping(value = "/register", method = RequestMethod.GET)

public String register(Model model) {

    logger.info("You arrived into the register formular");

    UserDAO userDAO = new UserDAO();

    model.addAttribute("user", userDAO);

    return "registration";

}



@RequestMapping(value = "/exportAll", method = RequestMethod.POST )

public String exportProducts(@RequestParam(value = "getAll",defaulTotallue = "null") String ids,
   HttpServletRequest request,

                        HttpServletResponse response ) {

    if (ids.equals("null")){

        return "redirect:medicines";

    }

    else{

        request.getSession().setAttribute("ids",ids);
```

```
            request.getSession().setAttribute("response",response);

            return "redirect:exportAllSuccess";

        }

    }

    @RequestMapping(value="/register/{id}", method = RequestMethod.GET)

    public String userRegister(Model model, @PathVariable Integer id){

        logger.info("You can now confirm your registration");

        medi_user_service.activatingUserAccount(id);

        model.addAttribute("success",Boolean.TRUE);

        return "redirect:/login";

    }

    @RequestMapping(value = "/register", method = RequestMethod.POST)

    public String registerUserAccount(Model model, @ModelAttribute("user") UserDAO userDAO,
      BindingResult bindingResult) {

logger.info("Now it's time for registration validation");

        userFormValidators.validate(userDAO, bindingResult);

        if (!bindingResult.hasErrors()) {

            medi_user_service.registerNewUserAccount(userDAO);

            return "redirect:medicines";

        } else {

            model.addAttribute("user", userDAO);

            return "registration";
}

    }
  }
```

```
package com.me.project.controller;

import javax.servlet.http.HttpSession;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.bind.annotation.RequestParam;

import com.me.project.facade.CartFacade;

import com.me.project.model.CartDAO;

@Controller

public class CartController extends UserController {

    @Autowired

    CartFacade cartFacade;

    private static final Logger logger = LoggerFactory.getLogger(CartController.class);

    @RequestMapping(value = "/adDAOCart", method = RequestMethod.POST)

    public String adDAOCart(

                    @RequestParam(value="quantity", defaulTotallue = "1" ) Integer quantity,

                    @RequestParam( value="productId") Integer productId,

                    HttpSession session){

        logger.info(String.valueOf(productId));

        logger.info(String.valueOf(quantity));
```

```
            Integer cartId = (Integer) session.getAttribute(CART_SESSION_ATTRIBUTE_NAME);

            cartFacade.adDAOCart(cartId, productId, quantity);

            return "redirect:home";

    }

    @RequestMapping(value = "/cart-page", method = RequestMethod.GET)

    public String cartPage(HttpSession session, Model model) {

            Integer idCart = (Integer) session.getAttribute(CART_SESSION_ATTRIBUTE_NAME);

            CartDAO cart = cartFacade.getCartById(idCart);

            model.addAttribute("cart", cart);

            return "cart-page";

    }

    @RequestMapping(params = "updateId", value = "/cart-page/update", method = RequestMethod.POST)

    public String cartPageAmount(@RequestParam("amount") Integer amount, @RequestParam("entryId")
    Integer entryId){

            cartFacade.setAmount(amount, entryId);

            return  "redirect:/cart-page";

    }

    @RequestMapping(params = "deleteId", value = "/cart-page/delete", method = RequestMethod.POST)

    public String deleteEntry(@RequestParam("deleteId") Integer id){

            cartFacade.deleteEntry(id);

            return "redirect:/cart-page";

    }}
```

```
package com.me.project.controller;

import com.me.project.model.Category;

import com.me.project.model.CheckBox;

import com.me.project.model.FilterOptions;
```

```
import com.me.project.model.Product;

import com.me.project.rest.RestService;

import com.me.project.rest.Review;

import com.me.project.service.CategoriesService;

import com.me.project.service.Filt_serv;

import com.me.project.service.Prod_services;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.*;

import java.util.AbstractMap;

import java.util.ArrayList;

import java.util.List;

import java.util.Map;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.bind.annotation.RequestParam;

import javax.servlet.http.HttpServletRequest;

@Controller

public class ClientController extends UserController {

    @Autowired

    Prod_services prod_services;
```

```
@Autowired
CategoriesService categoriesService;
@Autowired
Filt_serv filt_serv;
private static final Logger logger = LoggerFactory.getLogger(ClientController.class);
@RequestMapping(value = "/product/{id}", method = RequestMethod.GET)
public String displayProductDetails(Model model, @PathVariable Integer id) {
    Product medi = prod_services.getProduct(id);
    AbstractMap.SimpleEntry<Double ,List<Review>> reviewsPair  = prod_services.getReviews(id);
    model.addAttribute("reviewsList", reviewsPair.geTotallue());
    model.addAttribute("average", reviewsPair.getKey());
    model.addAttribute("medi", medi);
    return "product-details";
}
 @RequestMapping(value = "/shop/{page}", method = RequestMethod.GET)
public String cViewProducts(HttpServletRequest request, Model model, @PathVariable Integer page,
                        @RequestParam(value = "argSort", required = false, defaulTotallue = "idasc") String
  argSort,
                        @RequestParam(value = "category", required = false, defaulTotallue = "null") String
  category,
                        @RequestParam(value = "year", required = false, defaulTotallue = "0") String year,
                        @RequestParam(value = "type", required = false, defaulTotallue = "null") String type) {
    List<Product> productList=new ArrayList<Product>();
    Integer startIndex = (page - 1) * 9;
    List<CheckBox> categories = new ArrayList<CheckBox>();
    List<CheckBox> years = new ArrayList<CheckBox>();
    List<CheckBox> types = new ArrayList<CheckBox>();
```

```
Map<String, Integer> numberOfProductsPerType;

Map<String, Integer> numberOfProductsPerYear;

Map<String, Integer> numberOfProductsPerCategory;

if (category.equals("null") & year.equals("0") & type.equals("null")){    // put the checkboxes cleared

    for (Category c : filt_serv.getCategories())

        categories.add(new CheckBox(c.getName(), false));

    for (String y : filt_serv.getYears())

        years.add(new CheckBox(y, false));



    for (String t : filt_serv.getTypes())

        types.add(new CheckBox(t, false));



    productList= prod_services.sortProducts(argSort, startIndex, 10);  //get with +1 to see if  there will be a
next page

}



else {  //any checkbox was checked

    FilterOptions filterOptions=new FilterOptions();

    if (!category.equals("none"))

        filterOptions.setCategory(category);

    if (!year.equals("none"))

        filterOptions.setYear(year);

    if (!type.equals("none"))

        filterOptions.setType(type);
```

```
    productList = filt_serv.applyFilters(filterOptions, startIndex, 10, argSort);

    List<Category> allCategories = categoriesService.getAllTheCategories();

    categories = filt_serv.getCategoriesFilteredProducts(productList, filterOptions, allCategories);

    years = filt_serv.getYearsFilteredProducts(productList, filterOptions);

    types = filt_serv.getTypesFilteredProducts(productList, filterOptions);

    numberOfProductsPerType = filt_serv.getNumberOfProductsPerType(productList);

    numberOfProductsPerYear = filt_serv.getNumberOfProductsPerYear(productList);

    numberOfProductsPerCategory = filt_serv.getNumberOfProductsPerCategory(productList,
allCategories);

    model.addAttribute("numberOfProductsPerYear", numberOfProductsPerYear);

    model.addAttribute("numberOfProductsPerType", numberOfProductsPerType);

    model.addAttribute("numberOfProductsPerCategory", numberOfProductsPerCategory);

 }

 boolean nextArrow = false;

 boolean noPageNb = false;

 if (productList.size() > 9) {

    nextArrow = true;

    productList.remove(9);

 }

 if (page == 1 & productList.size() < 9) {

    logger.info("size=" + productList.size());

    noPageNb = true;

 }

 model.addAttribute("category", category);

 model.addAttribute("type", type);

 model.addAttribute("year", year);
```

```
        model.addAttribute("noPageNb", noPageNb);
        model.addAttribute("nextArrow", nextArrow);
        model.addAttribute("argSort", argSort);
        model.addAttribute("currentPage", page);
        model.addAttribute("mediList", productList);
        model.addAttribute("categories", categories);
        model.addAttribute("years", years);
        model.addAttribute("types", types);
        model.addAttribute("number", productList.size());
        return "shop";
    }
}

package com.me.project.controller;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
@Controller
public class ErrorController {
    @RequestMapping(value = "/error", method = RequestMethod.GET)
    public String getErrorPage() {
        return "error";
    }
}
```

```
package com.me.project.controller;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.transaction.annotation.Transactional;

import org.springframework.web.bind.annotation.ModelAttribute;

import com.me.project.facade.CartFacade;

import javax.servlet.http.HttpSession;

public class UserController {

  protected static final String CART_SESSION_ATTRIBUTE_NAME = "sessionCartId";

   @Autowired

   CartFacade cartFacade;

   @ModelAttribute("nrProductsFromCart")

   @Transactional

   public Integer getNrProductsFromCart(HttpSession session) {

      Integer cartId = (Integer) session.getAttribute(CART_SESSION_ATTRIBUTE_NAME);

      Integer nrProductsFromCart = cartFacade.getNrProductsFromCart(cartId);

      return nrProductsFromCart;

   }

}
```

```
package com.me.project.controller.errorhandling;

import org.springframework.stereotype.Controller;

import org.springframework.web.multipart.MaxUploadSizeExceededException;

import org.springframework.web.servlet.HandlerExceptionResolver;

import org.springframework.web.servlet.ModelAndView;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;
```

```
@Controller
public class GlobalMethodHandlerExeptionResolver implements HandlerExceptionResolver {
   @Override
   public ModelAndView resolveException(HttpServletRequest httpServletRequest, HttpServletResponse
     httpServletResponse, Object o, Exception exception) {
      if (exception instanceof MaxUploadSizeExceededException) {
         ModelAndView modelAndView = new ModelAndView("picture-size-exception");
         modelAndView.getModel().put("pictureException", "Picture size exceeds limit of 2 MB!");
         return modelAndView;
      }
      return null;
   }
}

package com.me.project.controller.interceptors;
import org.springframework.web.servlet.handler.HandlerInterceptorAdapter;
import com.me.project.service.CartService;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
public class CartRestorationInterceptor extends HandlerInterceptorAdapter {
   private CartService cartService;
   private static final String CART_COOKIE_NAME = "cartId";
   private static final String CART_SESSION_ATTRIBUTE_NAME = "sessionCartId";
```

```
@Override
public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) {
    Cookie cartCookie = null;
    Cookie cookies[] = request.getCookies();
    if (cookies != null) {
        for (Cookie cookie : cookies) {
            if (cookie.getName().equals(CART_COOKIE_NAME)) {
                cartCookie = cookie;
                break;
            }
        }
    }
    Integer cookieValue = null;
    if (cartCookie != null) {
        cookieValue = Integer.parseInt(cartCookie.geTotallue());
    }
    HttpSession session = request.getSession();
    Integer sessionCartId = (Integer) session.getAttribute(CART_SESSION_ATTRIBUTE_NAME);
    if(cookieValue == null){
        if (sessionCartId == null) {
            sessionCartId = cartService.createNewCart();
        }
    } else {
        if (sessionCartId == null) {
            sessionCartId = cookieValue;
```

```
            }
        }
        session.setAttribute(CART_SESSION_ATTRIBUTE_NAME, sessionCartId);
        cartCookie = new Cookie(CART_COOKIE_NAME, sessionCartId.toString());
        response.addCookie(cartCookie);
        return true;
    }
    public void setCartService(CartService cartService) {
        this.cartService = cartService;
    }
}

package com.me.project.facade;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;
import com.me.project.model.Cart;
import com.me.project.model.CartDAO;
import com.me.project.service.CartService;
import com.me.project.service.convertors.DAOConverter;
@Component
public class CartFacade {
        @Autowired
        CartService cartService;
        @Autowired
        DAOConverter converter;
```

```
public void adDAOCart(Integer cartId, Integer productId, Integer quantitiy){
        cartService.adDAOCart(cartId, productId, quantitiy);
}
public Cart getCart(int cartId){
        return cartService.getCart(cartId);
}
public void updateTotalPrice(Cart cart){
        cartService.updateTotalPrice(cart);
}
public Integer getNrProductsFromCart(Integer cartId){
        return cartService.getNrProductsFromCart(cartId);
}
@Transactional
public CartDAO getCartById(Integer idCart){
        Cart cart = cartService.getCartById(idCart);
        cartService.updateTotalPrice(cart);
        cartService.update(cart);
        CartDAO cartDAO = converter.convertCartToCartDAO(cart);
        return cartDAO;
}

public void setAmount(Integer amount, Integer entryId){
        cartService.setAmount(amount, entryId);
}
```

```
	public void deleteEntry(Integer id){
		cartService.deleteEntry(id);
	}
}

package com.me.project.interfaces;
import java.util.HashMap;
import java.util.List;
import com.me.project.model.Category;
public interface CategoriesManagerInterface{
   List<Category> getAllTheCategories();
   HashMap<String, Integer> getAllTheCategoriesMap();
   //Integer getTheNumberOfProducts(Category category);
   String getCategory(String productName);
   Boolean addCategory(Category category);
}

package com.me.project.interfaces;
import com.me.project.model.users.User;
import com.me.project.model.users.UserDAO;
public interface IMedi_user_service {
 void registerNewUserAccount(UserDAO accountDAO);
   boolean login(User user);
   void addUser(User user);
   void activatingUserAccount(Integer id);
}
```

```
package com.me.project.interfaces;

import javax.validation.Constraint;

import javax.validation.Payload;


import com.me.project.model.validatorss.PasswordMatchesValidators;


import java.lang.annotation.Documented;

import java.lang.annotation.Retention;

import java.lang.annotation.Target;

import static java.lang.annotation.ElementType.ANNOTATION_TYPE;

import static java.lang.annotation.ElementType.TYPE;

import static java.lang.annotation.RetentionPolicy.RUNTIME;


@Target({TYPE,ANNOTATION_TYPE})

@Retention(RUNTIME)

@Constraint(validatedBy = PasswordMatchesValidators.class)

@Documented

public @interface PasswordMatches {

   String message() default "Passwords don't match";

   Class<?>[] groups() default {};

   Class<? extends Payload>[] payload() default {};

}
```

```
package com.me.project.interfaces;

import org.hibernate.validators.constraints.impl.EmailValidators;

import javax.validation.Constraint;

import javax.validation.Payload;

import java.lang.annotation.Documented;

import java.lang.annotation.Retention;

import java.lang.annotation.Target;

import static java.lang.annotation.ElementType.*;

import static java.lang.annotation.RetentionPolicy.RUNTIME;

@Target({TYPE, FIELD, ANNOTATION_TYPE})

@Retention(RUNTIME)

@Constraint(validatedBy = EmailValidators.class)

@Documented


public @interface ValidEmail {

   String message() default "Invalid email";

   Class<?>[] groups() default {};

   Class<? extends Payload>[] payload() default {};

}


package com.me.project.model;


import javax.persistence.*;

import java.util.List;

@Entity

@Table(name = "cart")
```

```
public class Cart {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "cartId")
    private Integer cartId;
    @Column(name = "totalCost")
    private Double totalCost;
    @Embedded
    @OneToMany(mappedBy = "cart", cascade = {CascadeType.ALL}, orphanRemoval = true)
    private List<CartEntry> listOfEntries;


        public List<CartEntry> getListOfEntries() {
        return listOfEntries;
    }
    public void setListOfEntries(List<CartEntry> listOfEntries) {
        this.listOfEntries = listOfEntries;
    }
    public Integer getCartId() {
        return cartId;
    }
    public void setCartId(Integer cartId) {
        this.cartId = cartId;
    }
```

```
    public Double getTotalCost() {

        return totalCost;

    }

    public void setTotalCost(Double totalCost) {

        this.totalCost = totalCost;

    }

}
```

```
package com.me.project.model;

import java.util.List;

import org.springframework.transaction.annotation.Transactional;

@Transactional
public class CartDAO {

    private final static Integer TA_RATES = 19;
    private Integer cartId;

    private Double totalCost;

    private Double TOTAL;

    public Double getTOTAL() {
        return TOTAL;
    }

    public void setTOTAL() {
        this.TOTAL = (TA_RATES / 100.0) * totalCost;
        this.TOTAL = Math.round(this.TOTAL * 100.0) / 100.0;
    }

    private List<CartEntryDAO> listOfEntries;

    public Double getTotalCost() {
        return totalCost;
    }

    public void setTotalCost(Double totalCost) {
        this.totalCost = totalCost;
    }

    public List<CartEntryDAO> getListOfEntries() {
        return listOfEntries;
    }

    public void setListOfEntries(List<CartEntryDAO> listOfEntries) {
        this.listOfEntries = listOfEntries;
```

```
    }

    public Integer getCartId() {
        return cartId;
    }

    public void setCartId(Integer cartId) {
        this.cartId = cartId;
    }
}
package com.me.project.model;

import javax.persistence.*;

@Embeddable
@Entity
@Table(name = "cartentry")
public class CartEntry {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "entryId")
    private Integer entryId;

    @ManyToOne
    @JoinColumn(name = "medId")
    private Product product;

    @ManyToOne
    @JoinColumn(name = "cartId")
    private Cart cart;

    @Column(name = "quantity")
    private Integer quantity;

    public Integer getEntryId() {
        return entryId;
    }

    public void setEntryId(Integer entryId) {
        this.entryId = entryId;
    }

    public Product getProduct() {
        return product;
    }

    public void setProduct(Product product) {
        this.product = product;
    }

    public Cart getCart() {
        return cart;
    }
```

```
    public void setCart(Cart cart) {
        this.cart = cart;
    }

    public Integer getQuantity() {
        return quantity;
    }

    public void setQuantity(Integer quantity) {
        this.quantity = quantity;
    }
}
package com.me.project.model;

import org.springframework.transaction.annotation.Transactional;

@Transactional
public class CartEntryDAO {

    private Integer entry_Id;

    private Product product;

    private Cart cart;

    private Integer quantity;

    public Integer getEntryId() {
        return entry_Id;
    }

    public void setEntryId(Integer entryId) {
        this.entry_Id = entryId;
    }

    public Product getProduct() {
        return product;
    }

    public void setProduct(Product product) {
        this.product = product;
    }

    public Cart getCart() {
        return cart;
    }

    public void setCart(Cart cart) {
        this.cart = cart;
    }

    public Integer getQuantity() {
        return quantity;
    }
```

```
    public void setQuantity(Integer quantity) {
        this.quantity = quantity;
    }
}
```

```
package com.me.project.model;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.Table;

import javax.persistence.GenerationType;

import javax.persistence.Column;

@Entity

@Table(name = "categories")

public class Category {

   @Id

   @GeneratedValue(strategy = GenerationType.IDENTITY)

   @Column(name = "id")

   private Integer categoryID;

   @Column

   private String name;

   public Category(){

   }

   public Category(Integer id, String name){

      this.categoryID = id;

      this.name = name;
```

```
    }

    public Integer getID(){
        return this.categoryID;
    }
    public void setCategoryID(Integer id){
        this.categoryID = id;
    }

     public String getName(){
        return this.name;
    }
    public void setName(String name){
        this.name = name;
    }
}
```

```java
package com.me.project.model;

public class CheckBox {

    private String name;
    private Boolean state;

    public CheckBox(String name, Boolean state){
        this.name = name;
        this.state = state;
    }

    public String getName(){
        return this.name;
    }
```

```
    public Boolean getState(){
        return this.state;
    }

    public void setName(String name){
        this.name = name;
    }

    public void setState(Boolean state){
        this.state = state;
    }

}
```

```
package com.me.project.model;

import java.util.ArrayList;

import java.util.List;

public class FilterOptions {

   private String category;

   private String year;

   private String type;

   public FilterOptions(){

      }

  public String getCategory(){

      return this.category;

   }

   public String getYear(){

      return this.year;

   }

   public void setCategory(String category) {

      this.category = category;    }
```

```
    public void setYear(String year) {
        this.year = year;
    }
    public String getType() {
        return type;
    }
    public void setType(String type) {
        this.type = type;
    }
}
```

```
package com.me.project.model;

public enum MedicineType {
    PAINKILLERS("painkillers"), ANALGESIC("analgesic"), ANTACID("antacid"),
OTHERS("others");

    private String medTypeValue;

    MedicineType(String value) {
        this.medTypeValue = value;
    }

    public String getMediTypeValue() {
        return this.medTypeValue;
    }

}
```

```
package com.me.project.model;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.Table;
```

```
import javax.persistence.GenerationType;

import javax.persistence.Column;

@Entity

@Table(name = "medicine")

public class Product {

   @Id

   @GeneratedValue(strategy = GenerationType.IDENTITY)

   @Column(name = "medid")

   private int medId;

   @Column

   private String name;

   @Column

   private String brand;

   @Column

   private Double price;

   @Column

   private Integer stock;

   @Column(name = "type")

   private String medType;

   @Column

   private String countryOfOrigin;

   @Column

   private Integer yearOfProduction;
```

```
    @Column
    private Double volume;
    @Column
    private String description;
    @Column
    private String recommendations;
    @Column
    private String picture;
    @Column(name = "category")
    private Integer categoryID;
    public Product() {
    }
    public Product(String name, String brand, double price, int stock, String mediType, String
countryOfOrigin, int yearOfProduction, double volume, String description, String recommendations,
String categoryID) {
        this.name = name;
        this.brand = brand;
        this.price = price;
        this.stock = stock;
        this.medType = medType;
        this.countryOfOrigin = countryOfOrigin;
        this.yearOfProduction = yearOfProduction;
        this.volume = volume;
        this.description = description;
```

```
    this.recommendations = recommendations;
}
public int getMediId() {
    return medId;
}
public void setMediId(int mediId) {
    this.medId = mediId;
}
public String getName() {
    return name;
}
public void setName(String name) {
    this.name = name;
}
public String getBrand() {
    return brand;
}
public void setBrand(String brand) {
    this.brand = brand;
}
public Double getPrice() {
    return price;
}
```

```
public void setPrice(Double price) {
    this.price = price;
}
public Integer getStock() {
    return stock;
}
public void setStock(Integer stock) {
    this.stock = stock;
}
public String getmedType() {
    return medType;
}
public void setmedType(String mediType) {
    this.medType = mediType;
}
public String getCountryOfOrigin() {
    return countryOfOrigin;
}
public void setCountryOfOrigin(String countryOfOrigin) {
    this.countryOfOrigin = countryOfOrigin;
}
```

```
public Integer getYearOfProduction() {
    return yearOfProduction;
}
public void setYearOfProduction(Integer yearOfProduction) {
    this.yearOfProduction = yearOfProduction;
}
public Double getVolume() {
    return volume;
}
public void setVolume(Double volume) {
    this.volume = volume;
}
public String getDescription() {
    return description;
}

public void setDescription(String description) {
    this.description = description;
}
public String getRecommendations() {
    return recommendations;
}
```

```
public void setRecommendations(String recommendations) {
    this.recommendations = recommendations;
}
public String getPicture() {
    return picture;
}
public void setPicture(String picture) {
    this.picture = picture;
}
public Integer getCategoryID() {
    return categoryID;
}
public void setCategoryID(Integer categoryID) {
    this.categoryID = categoryID;
}
@Override
public String toString() {
    return "Product{" +
            "mediId=" + medId +
            ", name='" + name + '\'' +
            ", brand='" + brand + '\'' +
            ", price=" + price +
            ", stock=" + stock +
```

```
            ", mediType='" + medType + '\'' +
            ", countryOfOrigin='" + countryOfOrigin + '\'' +
            ", yearOfProduction=" + yearOfProduction +
            ", volume=" + volume +
            ", description='" + description + '\'' +
            ", recommendations='" + recommendations + '\'' +
            '}';
    }
}
```

```
package com.me.project.model;

import org.springframework.transaction.annotation.Transactional;

@Transactional

public class ProductDAO {
    private String name;
    private int mediId;
    private String brand;
    private Double price;
    private Integer stock;
    private String medType;
    private String countryOfOrigin;
    private Double volume;
    private String recommendations;
    private Integer yearOfProduction;
    private String description;
    private Integer categoryID;


    public ProductDAO() {
    }

    public ProductDAO(Product product) {

    }

    public String getName() {
        return name;
    }
```

```java
public void setName(String name) {
    this.name = name;
}

public int getMediId() {
    return mediId;
}

public void setMediId(int mediId) {
    this.mediId = mediId;
}

public String getBrand() {
    return brand;
}

public void setBrand(String brand) {
    this.brand = brand;
}

public Double getPrice() {
    return price;
}

public void setPrice(Double price) {
    this.price = price;
}

public Integer getStock() {
    return stock;
}

public void setStock(Integer stock) {
    this.stock = stock;
}

public String getMedType() {
    return medType;
}

public void setMedType(String medType) {
    this.medType = medType;
}

public String getCountryOfOrigin() {
    return countryOfOrigin;
}

public void setCountryOfOrigin(String countryOfOrigin) {
    this.countryOfOrigin = countryOfOrigin;
}

public Double getVolume() {
    return volume;
}
```

```
    public void setVolume(Double volume) {
        this.volume = volume;
    }

    public String getRecommendations() {
        return recommendations;
    }

    public void setRecommendations(String recommendations) {
        this.recommendations = recommendations;
    }

    public Integer getYearOfProduction() {
        return yearOfProduction;
    }

    public void setYearOfProduction(Integer yearOfProduction) {
        this.yearOfProduction = yearOfProduction;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public Integer getCategoryID() {
        return categoryID;
    }

    public void setCategoryID(Integer categoryID) {
        this.categoryID = categoryID;
    }
}
package com.me.project.model.users;

import javax.persistence.*;

@Entity
@Table (name = "users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Integer id;
    @Column(name = "name")
    private String name;
    @Column(name = "password")
    private String password;
    @Column(name = "email")
    private String email;
    @Column(name = "phoneNumber")
```

```
private Long phoneNumber;
@Column(name = "isActive")
private Boolean isActive = false;

public User() {
}

public User(String name, String password, String email, Long phoneNumber) {
    this.name = name;
    this.password = password;
    this.email = email;
    this.phoneNumber = phoneNumber;
    this.isActive = false;
}

public Integer getId() {
    return id;
}

public void setId(Integer id) {
    this.id = id;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getPassword() {
    return password;
}

public void setPassword(String password) {
    this.password = password;
}

public String getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}

public Long getPhoneNumber() {
    return phoneNumber;
}

public void setPhoneNumber(Long phoneNumber) {
    this.phoneNumber = phoneNumber;
}
```

```
    public Boolean getActive() {
        return isActive;
    }

    public void setActive(Boolean active) {
        isActive = active;
    }

    @Override
    public String toString() {
        return "User{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", password='" + password + '\'' +
                ", email='" + email + '\'' +
                ", phoneNumber=" + phoneNumber +
                ", isActive=" + isActive +
                '}';
    }
}
package com.me.project.model.users;

import org.springframework.transaction.annotation.Transactional;

@Transactional
public class UserDAO{

  private String name;
    private String password;
   private String email;
    private String phoneNumber;
    private String matchingPassword;
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }
```

```
    public String getPhoneNumber() {
        return phoneNumber;
    }

    public void setPhoneNumber(String phoneNumber) {
        this.phoneNumber = phoneNumber;
    }

    public String getMatchingPassword() {
        return matchingPassword;
    }

    public void setMatchingPassword(String matchingPassword) {
        this.matchingPassword = matchingPassword;
    }
}
```

```
package com.me.project.model.users;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Component;

import org.springframework.validation.Errors;

import org.springframework.validation.ValidationUtils;

import org.springframework.validation.Validators;

import com.me.project.model.validatorss.EmailValidators;

@Component

public class UserFormValidators implements Validators {

   @Autowired

   EmailValidators emailValidators;

   @Override

   public boolean supports(Class<?> clazz) {

      return User.class.equals(clazz);

   }
```

```
   @Override

   public void validate(Object target, Errors errors) {

      UserDAO user = (UserDAO) target;

      ValidationUtils.rejectIfEmptyOrWhitespace(errors, "name", "NotEmpty.userForm.name","Name is
required!");

      ValidationUtils.rejectIfEmptyOrWhitespace(errors, "email", "NotEmpty.userForm.email","Email is
required!");

      ValidationUtils.rejectIfEmptyOrWhitespace(errors, "password",
"NotEmpty.userForm.password","Password is required!");

      ValidationUtils.rejectIfEmptyOrWhitespace(errors, "phoneNumber",
"NotEmpty.userForm.phoneNumber","Phone Number is required!");

      ValidationUtils.rejectIfEmptyOrWhitespace(errors, "matchingPassword",
"NotEmpty.userForm.matchingPassword","Mtching password is required!");

      if(!emailValidators.isValid(user.getEmail()))

         errors.rejecTotallue("email","Pattern.userForm.email","Invalid Email format!");

      if(emailValidators.exists(user.getEmail()))

         errors.rejecTotallue("email","Email.Exists.Error","Email already exists!");

      if(!user.getPhoneNumber().matches("^[0-9]\\d*$"))

         errors.rejecTotallue("phoneNumber","Pattern.useForm.phoneNumber","Invalid Phone Number
format!");

      if (!user.getPassword().equals(user.getMatchingPassword())) {

         errors.rejecTotallue("matchingPassword", "Diff.userform.confirmPassword","Passwords do not
match, please retype!");

      }

   }
```

```
}

package com.me.project.model.validatorss;

import com.me.project.model.users.User;

import com.me.project.persistence.UserPersistence;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Component;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

@Component

public class EmailValidators {

   @Autowired

   private UserPersistence userPersistence;

   private static final String EMAIL_PATTERN = "^[_A-Za-z0-9-+]+(.[_A-Za-z0-9-]+)*@" + "[A-Za-z0-9-
]+(.[A-Za-z0-9]+)*(.[A-Za-z]{2,})$";

   public boolean isValid(String email){

      return (validateEmail(email));

   }

   private boolean validateEmail(String email) {

      Pattern pattern = Pattern.compile(EMAIL_PATTERN);

      Matcher matcher = pattern.matcher(email);

      return matcher.matches();

   }
```

```
    private boolean emailExist(String email) {
        User user = userPersistence.findByEmail(email);
        if (user != null) {
            return true;
        }
        return false;
    }
    public boolean exists(String email){
        return emailExist(email);
    }
}
package com.me.project.model.validatorss;
import com.me.project.interfaces.PasswordMatches;
import com.me.project.model.users.UserDAO;
import javax.validation.ConstrainTotallidators;
import javax.validation.ConstrainTotallidatorsContext;
public class PasswordMatchesValidators
        implements ConstrainTotallidators<PasswordMatches, Object> {
    @Override
    public void initialize(PasswordMatches constraintAnnotation) {
    }
    @Override
    public boolean isValid(Object obj, ConstrainTotallidatorsContext context){
```

```
        UserDAO user = (UserDAO) obj;
        return user.getPassword().equals(user.getMatchingPassword());
    }
}
package com.me.project.persistence;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;
import com.me.project.model.Cart;
import com.me.project.model.CartEntry;
import java.util.List;
@Repository
public class CartPersistence {
    @Autowired
    private SessionFactory sessionFactory;
    private static final Logger logger = LoggerFactory.getLogger(ProductPersistence.class);
```

```
    public void persistCart(Cart cart){

    }
public List<CartEntry> getAllEntries(Integer cartId){
        List<CartEntry> allEntries;
        Session session = sessionFactory.getCurrentSession();
        allEntries = session.createQuery("from CartEntry where cartId=" + cartId).list();
        return  allEntries;
    }
    public Cart getCart(Integer id) {
        Session session = sessionFactory.getCurrentSession();
        Cart cart = (Cart) session.get(Cart.class, id);
        return cart;
    }
    public Integer addCart(Cart cart) {
        Session session = sessionFactory.getCurrentSession();
        session.save(cart);
        return cart.getCartId();
    }


    public void updateCart(Cart cart){
        Session session = sessionFactory.getCurrentSession();
        session.saveOrUpdate(cart);
    }
```

```
public int lastAddedCartId(){
   List<Cart> allEntries;
   Session session = sessionFactory.getCurrentSession();
   allEntries = session.createQuery("from Cart").list();
   return  allEntries.get(allEntries.size()-1).getCartId();
}
public void setAmount(Integer amount, Integer entryId){
   Session session = sessionFactory.getCurrentSession();
   String stringQuery = "UPDATE CartEntry SET quantity = :amount WHERE entryId = :id";
   Query query = session.createQuery(stringQuery);
   query.setParameter("amount", amount);
   query.setParameter("id", entryId);
   query.executeUpdate();
}
public void deleteEntry(Integer entryId){
   Session session = sessionFactory.getCurrentSession();
   String stringQuery = "DELETE FROM CartEntry WHERE entryId = :id";
   Query query = session.createQuery(stringQuery);
   query.setParameter("id", entryId);
   query.executeUpdate();
}
```

```
    public CartEntry getCartEntry(Integer entryId){
        Session session = sessionFactory.getCurrentSession();
        CartEntry cartEntry = (CartEntry) session.get(CartEntry.class, entryId);
        return cartEntry;
    }
}
package com.me.project.persistence;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;
import com.me.project.model.Category;
import com.me.project.model.Product;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
@Repository
public class ProductPersistence {
    @Autowired
    private SessionFactory sessionFactory;
```

```
    private static final Logger logger = LoggerFactory.getLogger(ProductPersistence.class);

    public List<Product> getAllProducts() {
        List<Product> products;
        Session session = sessionFactory.openSession();
        products = session.createQuery("from Product").list();
        session.close();
        return products;
    }

    public void addProduct(Product product) {
        Session session = this.sessionFactory.openSession();
        session.save(product);
        session.close();
    }

    public Product getProduct(Integer id) {
        Session session = sessionFactory.openSession();
        Product product = (Product) session.get(Product.class, id);
        session.close();
        return product;
    }

    public void update(Product product){
        Session session = sessionFactory.openSession();
//      Product updatedProduct = (Product) session.merge(product);
        session.saveOrUpdate(product);
```

```
        session.flush();

        session.close();

    }

    public void deleteProduct(Product product){

        logger.info("Removing product with id " + product.getMediId());

        Session session = sessionFactory.openSession();

        session.delete(product);

        session.flush();

        session.close();

    }

    public String getProductPicture(int productId){

        Session session = sessionFactory.openSession();

        String productPicture = (String)session.createQuery("select p.picture from Product p where
p.mediId = :id")

                .setParameter("id", productId).uniqueResult();

        session.close();

        return productPicture;

    }

    public List<Category> getCategories() {

        List<Category> categories;

        Session session = sessionFactory.openSession();

        categories = session.createQuery("from Category").list();

        session.close();
```

```
    return categories;
}
public Boolean addCategory(Category category) {
    Session session = sessionFactory.openSession();
    session.save(category);
    session.close();
    return true;
}
public List<Product> sortProducts(String sortingCriteria,Integer startIndex,Integer maxNb) {
    List<Product> products;
    Session session = sessionFactory.openSession();
    String query=new String("from Product");
    if(sortingCriteria.startsWith("name"))
        if(sortingCriteria.contains("desc"))
            query+=" ORDER BY name DESC";
        else
            query+=" ORDER BY name ASC";
    if(sortingCriteria.startsWith("price"))
        if(sortingCriteria.contains("desc"))
            query+=" ORDER BY price DESC";
        else
            query+=" ORDER BY price ASC";
    Query q = session.createQuery(query);
```

```
    q.setFirstResult(startIndex);
    q.setMaxResults(maxNb);
    products=q.list();
    session.close();
    return products;
}
private String sortFilter(String sortingCriteria,String query){
    if(sortingCriteria.startsWith("name"))
        if(sortingCriteria.contains("desc"))
            query+=" ORDER BY name DESC";
        else
            query+=" ORDER BY name ASC";
    if(sortingCriteria.startsWith("price"))
        if(sortingCriteria.contains("desc"))
            query+=" ORDER BY price DESC";
        else
            query+=" ORDER BY price ASC";
     return  query;
}
public void deleteAll(String products){
    Session session = sessionFactory.openSession();
    String deleteQuerry = "DELETE FROM Product medi WHERE medi.mediId in (:productsIds)";
    List<Integer> ids = new ArrayList<Integer>();
```

```
        for(String currentString : Arrays.asList(products.split("\\s*,\\s*")))

            ids.add(Integer.valueOf(currentString));

        session.createQuery(deleteQuerry).setParameterList("productsIds", ids).executeUpdate();

    }

    public List<Product> getFilteredProducts(String category,Integer year,String type,Integer
startIndex,Integer maxNb, String args){

        List<Product> productList=new ArrayList<Product>();

        int categoryId=1;

        Session session=sessionFactory.openSession();

        String query=new String("from Category");

        List<Category> categories = session.createQuery(query).list();

        for (Category c:categories){

            if (c.getName().equals(category)){

                categoryId=c.getID();

            }

        }

        Query q=null;

        query="from Product";

        if (!category.equals("null")){

            if (!(year==0)){

                if (!type.equals("null")){

                    query+=" where category = :categoryParam and yearOfProduction = :yearParam and type =
:typeParam";

                    query=sortFilter(args,query);
```

```
            q =
session.createQuery(query).setParameter("categoryParam",categoryId).setParameter("yearParam",year
).setParameter("typeParam",type);

          }

          else {

            query+=" where category = :categoryParam and yearOfProduction = :yearParam";

            query=sortFilter(args,query);


q=session.createQuery(query).setParameter("categoryParam",categoryId).setParameter("yearParam",y
ear);

          }

        }

        else {

          if (!type.equals("null")){

            query+=" where category = :categoryParam  and type = :typeParam";

            query=sortFilter(args,query);

            q =
session.createQuery(query).setParameter("categoryParam",categoryId).setParameter("typeParam",type
);

          }

          else{

            query+=" where category = :categoryParam";

            query=sortFilter(args,query);

            q=session.createQuery(query).setParameter("categoryParam",categoryId);

          }

        }
```

```
        }
        else {
            if (!(year==0)){
                if (!type.equals("null")){
                    query+=" where yearOfProduction = :yearParam and type = :typeParam";
                    query=sortFilter(args,query);

q=session.createQuery(query).setParameter("yearParam",year).setParameter("typeParam",type);
                }
                else {
                    query+=" where yearOfProduction = :yearParam";
                    query=sortFilter(args,query);
                    q=session.createQuery(query).setParameter("yearParam",year);
                }
            }
            else {
                if (!type.equals("null")){
                    query+=" where type = :typeParam";
                    query=sortFilter(args,query);
                    q=session.createQuery(query).setParameter("typeParam",type);
                }
                else {
                    q=session.createQuery(query);
```

```
            }
          }
        }

        q.setFirstResult(startIndex);
        q.setMaxResults(maxNb);
        productList=q.list();
        return productList;
    }
}

package com.me.project.persistence;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;
import com.me.project.model.users.User;
import java.util.List;
@Repository
public class UserPersistence {
```

```
private static final Logger logger = LoggerFactory.getLogger(UserPersistence.class);

@Autowired

private SessionFactory sessionFactory;

public List<User> getAll() {

   List<User> users;

   Session session = sessionFactory.openSession();

   users = session.createQuery("from User").list();

   session.close();

   return users;

}

public void add(User user) {

   Session session = this.sessionFactory.openSession();

   session.save(user);

   session.close();

}

public void update(User user){

   Session session = sessionFactory.openSession();

   session.saveOrUpdate(user);

   session.flush();

   session.close();

}

public User findByEmail(String email){

   Session session = sessionFactory.openSession();
```

```
        String hql = "FROM User u WHERE u.email = '" + email + "'";
        Query query = session.createQuery(hql);
        List results = query.list();
        session.close();
        if (!results.isEmpty())
            return (User) results.get(0);
        return null;
    }
    public void delete(User user) {
        logger.info("Removing user with id " + user.getId());
        Session session = sessionFactory.openSession();
        session.delete(user);
        session.flush();
        session.close();
    }
    public User getUser(Integer id) {
        Session session = sessionFactory.openSession();
        User user = (User) session.get(User.class, id);
        session.close();
        return user;
    }
}
package com.me.project.providers;
```

```
import com.me.project.model.users.User;
import com.me.project.service.Medi_user_service;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.AuthenticationProvider;
import org.springframework.security.authentication.BadCredentialsException;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Component;
import java.util.ArrayList;
import java.util.List;
@Component
public class CustomAuthenticationProvider implements AuthenticationProvider {
    @Autowired
    private Medi_user_service medi_user_service;
    @Override
    public Authentication authenticate(Authentication authentication) throws AuthenticationException {
        String email = authentication.getName();
```

```
        Object credentials = authentication.getCredentials();
        String password = credentials.toString();
        if (email == null || password == null) {
            throw new BadCredentialsException("Invalid username/password");
        }
        User user = medi_user_service.getUserByEmail(email);
        if (user == null) {
            throw new UsernameNotFoundException("Invalid username/password");
        }
        if (! password.equals(user.getPassword())) {
            throw new BadCredentialsException("Invalid username/password");
        }
        if (!user.getActive()) {
            throw new BadCredentialsException("Inactive account!");
        }
        List<GrantedAuthority> grantedAuthorities = new ArrayList<>();
        grantedAuthorities.add(new SimpleGrantedAuthority("ROLE_USER"));
        Authentication auth = new UsernamePasswordAuthenticationToken(email, password,
grantedAuthorities);
        return auth;
    }
```

```
    @Override
    public boolean supports(Class<?> authentication) {
        return authentication.equals(UsernamePasswordAuthenticationToken.class);
    }
}
```

```
package com.me.project.rest;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;
import java.util.List;
@Controller
@RequestMapping("/reviews")
public class RestController {
    @Autowired
    private RestService restService;
    @RequestMapping(value = "/{productId}", method = RequestMethod.GET, produces =
MediaType.APPLICATION_JSON_VALUE)
    public @ResponseBody
    List<Review> getReview(@PathVariable Integer productId) {
```

```
        List<Review> reviews = restService.getReviews(productId);

        return reviews;

    }

    @RequestMapping(value = "/", method = RequestMethod.POST, produces =
MediaType.APPLICATION_JSON_VALUE)

    public ResponseEntity<Review> addReview(@RequestBody Review review) {

        restService.addReview(review);

        return new ResponseEntity<Review>(review, HttpStatus.OK);

    }

    @RequestMapping(value = "/{reviewId}", method = RequestMethod.DELETE, produces =
MediaType.APPLICATION_JSON_VALUE)

    public ResponseEntity deleteReview(@PathVariable Integer reviewId) {

        restService.deleteReview(reviewId);

        return new ResponseEntity(HttpStatus.OK);

    }

    @RequestMapping(value = "/", method = RequestMethod.PUT, produces =
MediaType.APPLICATION_JSON_VALUE)

    public ResponseEntity<Review> updateReview(@RequestBody Review review) {

        restService.updateReview(review);

        return new ResponseEntity<Review>(review, HttpStatus.OK);

    }

}

package com.me.project.rest;
```

```
import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Repository;

import java.util.List;

@Repository

public class RestPersistence {

   @Autowired

   private SessionFactory sessionFactory;

   public List<Review> getReviews(Integer productId) {

      List<Review> reviews;

      Session session = sessionFactory.openSession();

      reviews = session.createQuery("from Review where productId = (:id)").setParameter("id",
productId).list();

      session.close();

      return reviews;

   }

   public void addReview(Review review) {

      Session session = sessionFactory.openSession();

      session.save(review);

      session.close();

   }
```

```
    public void updateReview(Review review) {
        Session session = sessionFactory.openSession();
        session.saveOrUpdate(review);
        session.flush();
        session.close();
    }
    public void deleteReview(Integer reviewId) {
        Session session = sessionFactory.openSession();
        Review review = new Review();
        review.setId(reviewId);
        session.createQuery("delete from Review where id =(:id)").setParameter("id",
reviewId).executeUpdate();
        session.flush();
        session.close();
    }
}
package com.me.project.rest;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List;
@Service
public class RestService {
```

```
    @Autowired
    private RestPersistence restPersistence;
    public List<Review> getReviews(Integer productId) {
        return restPersistence.getReviews(productId);
    }
    public void add_Review(Review review) {
        restPersistence.add_Review(review);
    }
    public void delete_Review(Integer reviewId) {
        restPersistence.delete_Review(reviewId);
    }
    public void update_Review(Review review) {
        restPersistence.update_Review(review);
    }
}
```

```
package com.me.project.rest;

import javax.persistence.*;

@Entity
@Table(name = "reviews")
public class Review {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Integer id;
    @Column
    private Integer productId;
    @Column
    private Integer rating;
    @Column
    private String review;

    public Integer getId() {
```

```
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public Integer getProductId() {
        return productId;
    }

    public void setProductId(Integer productId) {
        this.productId = productId;
    }

    public Integer getRating() {
        return rating;
    }

    public void setRating(Integer rating) {
        this.rating = rating;
    }

    public String getReview() {
        return review;
    }

    public void setReview(String review) {
        this.review = review;
    }
}
```

```
package com.me.project.service;

import com.me.project.model.Cart;

import com.me.project.model.CartDAO;

import com.me.project.model.CartEntry;

import com.me.project.model.Product;

import com.me.project.persistence.CartPersistence;

import com.me.project.persistence.ProductPersistence;

import com.me.project.service.convertors.DAOConverter;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import javax.smartcardio.CardTerminal;
import java.util.List;
import java.util.Map;
@Service
public class CartService {
   @Autowired
   CartPersistence cartPersistence;
   @Autowired
   private ProductPersistence productPersistence;
   @Transactional
   public void adDAOCart(Integer cartId,Integer productId,Integer quantity){
      Cart cart=cartPersistence.getCart(cartId);
      Product product=productPersistence.getProduct(productId);
      List<CartEntry> cartEntryList = cart.getListOfEntries();
      int quantityWantToBuy=quantity;
      boolean alreadyInCart=false;
      int indexOfProductInEntryList=0;
```

```
for (CartEntry cartEntry:cartEntryList) {
   if(cartEntry.getProduct().getMediId()==product.getMediId()){
      quantityWantToBuy+=cartEntry.getQuantity();
      alreadyInCart=true;
      indexOfProductInEntryList=cartEntryList.indexOf(cartEntry);
   }
}
int stock=0;
if(product.getStock()!=null)stock=product.getStock();
if(stock<quantityWantToBuy) {
   quantityWantToBuy=stock;
}
CartEntry cartEntry;
if(alreadyInCart==false) {
  cartEntry = new CartEntry();
  cartEntry.setCart(cart);
  cartEntry.setProduct(product);
  cartEntry.setQuantity(quantityWantToBuy);
  cartEntryList.add(cartEntry);
  cart.setListOfEntries(cartEntryList);
}
else{
```

```
      cartEntryList.get(indexOfProductInEntryList).setQuantity(quantityWantToBuy);
    }
    updateTotalPrice(cart);
    cartPersistence.updateCart(cart);
}
@Transactional
public Cart getCart(int cartId){
   return cartPersistence.getCart(cartId);
}
@Transactional
public void update(Cart cart){
    cartPersistence.updateCart(cart);
}
public void updateTotalPrice(Cart cart){
    Double totalCost=0.0;
    List<CartEntry> cartEntryList=cart.getListOfEntries();
    for (CartEntry cartEntry : cartEntryList){
        totalCost+=cartEntry.getProduct().getPrice()*cartEntry.getQuantity();
    }
    totalCost = Math.round(totalCost * 100.0) / 100.0;
    cart.setTotalCost(totalCost);
}
```

```
@Transactional
public Cart getCartById(Integer idCart){
   return cartPersistence.getCart(idCart);
}
public int getCartIDFromCookie(HttpServletRequest request){
   Integer cartId=0;
   Cookie cookie[]=request.getCookies();
   for (Cookie cookieIterator:cookie
         ) {
      if(cookieIterator.getName().equals("medisShopCartId")){
         cartId=Integer.parseInt(cookieIterator.geTotallue());
         break;
      }
   }
  return  cartId;
}
@Transactional
public int createNewCart() {
   Cart newCart = new Cart();
   Integer cartId = cartPersistence.addCart(newCart);
   return cartId;
}
```

```
@Transactional
public void setAmount(Integer amount, Integer entryId){
   CartEntry cartEntry = cartPersistence.getCartEntry(entryId);
   Integer stock = cartEntry.getProduct().getStock();
   if(stock==null) stock=0;
   if(stock < amount)
      amount = stock;
   if(amount <= 0)
      cartPersistence.deleteEntry(entryId);
   else
      cartPersistence.setAmount(amount, entryId);
}
@Transactional
public void deleteEntry(Integer id){
   cartPersistence.deleteEntry(id);
}
@Transactional
public Integer getNrProductsFromCart(Integer cartId) {
   List<CartEntry> cartEntries = cartPersistence.getAllEntries(cartId);
   Integer nrProductsFromCart = 0;
   for (CartEntry entry : cartEntries) {
      nrProductsFromCart += entry.getQuantity();
   }
```

```
        return nrProductsFromCart;
    }
}
package com.me.project.service;
import com.me.project.interfaces.CategoriesManagerInterface;
import com.me.project.model.Category;
import com.me.project.model.Product;
import com.me.project.persistence.ProductPersistence;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import java.util.HashMap;
import java.util.List;
@Component
public class CategoriesService implements CategoriesManagerInterface {
    @Autowired
    private ProductPersistence productPersistence;
    public List<Category> getAllTheCategories() {
        List<Category> categories = productPersistence.getCategories();
        return categories;
    }
    public HashMap<String, Integer> getAllTheCategoriesMap() {
        HashMap<String, Integer> categoryQuantityMap = new HashMap<String, Integer>();
```

```
    List<Category> categories = productPersistence.getCategories();
    Integer productsNumber;
    for (Category category : categories) {
        productsNumber = getTheNumberOfProducts(category);
        categoryQuantityMap.put(category.getName(), productsNumber);
    }
    return categoryQuantityMap;
}
public Integer getTheNumberOfProducts(Category category) {
    Integer productsNumber = 0;
    List<Product> products = productPersistence.getAllProducts();
    for (Product product : products) {
        if (category.getID().equals(product.getCategoryID())) {
            productsNumber++;
        }
    }
    return productsNumber;
}
public String getCategory(String productName) {
    String categoryName = null;
    List<Product> products = productPersistence.getAllProducts();
    List<Category> categories = productPersistence.getCategories();
```

```
        for (Product p : products) {
            if (p.getName().equals(productName)) {
                for (Category c : categories) {
                    if (c.getID() == p.getCategoryID())
                        categoryName = c.getName();
                }
            }
        }
        return categoryName;
    }
    public Boolean addCategory(Category category) {
        Boolean succesfulOp = false;
        if (productPersistence.addCategory(category))
            succesfulOp = true;
        return succesfulOp;
    }
}
package com.me.project.service;
import org.apache.commons.io.FilenameUtils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.PropertySource;
```

```
import org.springframework.stereotype.Service;

import org.springframework.web.multipart.MultipartFile;

import java.io.File;

import java.io.FileInputStream;

import java.io.IOException;

import java.io.InputStream;

import java.nio.file.Files;

import java.nio.file.Path;

import java.nio.file.Paths;

import java.nio.file.StandardCopyOption;

import java.util.UUID;

@Service

@PropertySource("classpath:application.properties")

public class FileSystemStorageService {

   private static final Logger logger = LoggerFactory.getLogger(FileSystemStorageService.class);

   @Value("${file.upload.directory}")

   private String UPLOAD_DIRECTORY;

   public String store(MultipartFile file) throws IOException {

      String generatedID = UUID.randomUUID().toString();

      String fileName = generatedID + "." + FilenameUtils.getExtension(file.getOriginalFilename());

      final Path UPLOAD_ABSOLUTE_PATH = Paths.get(UPLOAD_DIRECTORY);

      try {

         if (file.isEmpty()) {
```

```
            logger.warn("Failed to store empty file " + fileName);
            throw new IOException("Failed to store empty file ");
        }
        InputStream inputStream = file.getInputStream();
        Path resolve = UPLOAD_ABSOLUTE_PATH.resolve(fileName);
        Files.copy(inputStream, resolve, StandardCopyOption.REPLACE_EXISTING);
      }
      catch (IOException e) {
        logger.warn("Failed to store file " + fileName, e);
        throw new IOException(e);
      }
      String fullPath =
UPLOAD_ABSOLUTE_PATH.resolve(fileName).toUri().toURL().toString().substring(6);
      return fullPath;
    }
    public String storeDefault() throws IOException {
      ClassLoader classLoader = getClass().getClassLoader();
      File file = new File(classLoader.getResource("../../resources/images/default-image.jpg").getFile());
      String generatedID = UUID.randomUUID().toString();
      String fileName = generatedID + ".jpg";
      final Path UPLOAD_ABSOLUTE_PATH = Paths.get(UPLOAD_DIRECTORY);
      try {
        InputStream inputStream = new FileInputStream(file);
```

```
            Path resolve = UPLOAD_ABSOLUTE_PATH.resolve(fileName);

            Files.copy(inputStream, resolve, StandardCopyOption.REPLACE_EXISTING);

        }

        catch (IOException e) {

            logger.warn("Failed to store file " + fileName, e);

            throw new IOException(e);

        }

        String fullPath =
UPLOAD_ABSOLUTE_PATH.resolve(fileName).toUri().toURL().toString().substring(6);

        return fullPath;

    }

}

package com.me.project.service;

import com.me.project.model.Category;

import com.me.project.model.CheckBox;

import com.me.project.model.FilterOptions;

import com.me.project.model.Product;

import com.me.project.persistence.ProductPersistence;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;
```

```
import java.util.Map;

@Service

public class Filt_serv {

   @Autowired

   private ProductPersistence productPersistence;

   @Autowired

   private CategoriesService categoriesService;

   public List<Product> applyFilters(FilterOptions filterOptions, Integer start, Integer max, String args) {

      String category = "null";

      Integer year = 0;

      String type = "null";

      if(!(filterOptions.getCategory() == null))

         category = filterOptions.getCategory();

      if(!(filterOptions.getYear() == null))

         year = Integer.parseInt(filterOptions.getYear());

      if(!(filterOptions.getType() == null))

         type = filterOptions.getType();

      List<Product> products = productPersistence.getFilteredProducts(category, year, type, start, max,
args);

      return products;

   }

   public List<String> getYears() {

      List<Product> products = productPersistence.getAllProducts();
```

```
    List<String> years = new ArrayList<String>();
    for (Product p : products) {
        String year = String.valueOf(p.getYearOfProduction());
        if (!years.contains(year))
            years.add(year);
    }
    return years;
}
public List<Category> getCategories() {
    return categoriesService.getAllTheCategories();
}
public List<String> getTypes() {
    List<Product> products = productPersistence.getAllProducts();
    List<String> types = new ArrayList<String>();
    for (Product p : products) {
        String year = String.valueOf(p.getmedType());
        if (!types.contains(year))
            types.add(year);
    }
    return types;
}
```

```
    public List<CheckBox> getCategoriesFilteredProducts(List<Product> filteredProducts, FilterOptions
filterOptions, List<Category> allCategories) {

        List<CheckBox> categories = new ArrayList<>();

        for (Product product : filteredProducts) {

            Integer categoryId = product.getCategoryID();

            String categoryString = null;

            boolean alreadyExists = false;

            for (Category category : allCategories) {

                if (category.getID() == categoryId) {

                    categoryString = category.getName();

                    break;

                }

            }

            for (CheckBox categoryCheckBox : categories) {

                if (categoryCheckBox.getName().equals(categoryString)) {

                    alreadyExists = true;

                }

            }

            if (!alreadyExists) {

                if (filterOptions.getCategory() != null && filterOptions.getCategory().equals(categoryString))

                    categories.add(new CheckBox(categoryString, true));

                else

                    categories.add(new CheckBox(categoryString, false));
```

```
        }
    }
    return categories;
}
public List<CheckBox> getYearsFilteredProducts(List<Product> filteredProducts, FilterOptions
filterOptions) {
    List<CheckBox> years = new ArrayList<CheckBox>();
    for (Product product : filteredProducts) {
        Integer year = product.getYearOfProduction();
        boolean alreadyExists = false;
        for (CheckBox yearCheckBox : years) {
            if (yearCheckBox.getName().equals(String.valueOf(year))) {
                alreadyExists = true;
            }
        }
        if (!alreadyExists) {
            if (filterOptions.getYear() != null && filterOptions.getYear().equals(String.valueOf(year)))
                years.add(new CheckBox(String.valueOf(year), true));
            else
                years.add(new CheckBox(String.valueOf(year), false));
        }
    }
    return years;
```

```
    }

    public List<CheckBox> getTypesFilteredProducts(List<Product> filteredProducts, FilterOptions
filterOptions) {

        List<CheckBox> types = new ArrayList<CheckBox>();

        for (Product product : filteredProducts) {

            String type = product.getmedType();

            boolean alreadyExists = false;

            for (CheckBox typeCheckBox : types) {

                if (typeCheckBox.getName().equals(String.valueOf(type))) {

                    alreadyExists = true;

                }

            }

            if (!alreadyExists) {

                if (filterOptions.getType() != null && filterOptions.getType().equals(type))

                    types.add(new CheckBox(type, true));

                else

                    types.add(new CheckBox(type, false));

            }

        }

        return types;

    }

    public Map<String, Integer> getNumberOfProductsPerCategory(List<Product> filteredProducts,
List<Category> allCategories){

        Map<String, Integer> numberOfProductsPerCategory = new HashMap<String, Integer>();
```

```
        String categoryString = null;
        for(Product product: filteredProducts){
            for (Category category : allCategories) {
                if (category.getID() == product.getCategoryID()) {
                    categoryString = category.getName();
                    break;
                }
            }
            if(numberOfProductsPerCategory.get(categoryString) != null){
                Integer initialVal = numberOfProductsPerCategory.get(categoryString);
                numberOfProductsPerCategory.put(categoryString, initialVal+1);
            } else {
                numberOfProductsPerCategory.put(categoryString, 1);
            }
        }
        return numberOfProductsPerCategory;
    }
    public Map<String, Integer> getNumberOfProductsPerYear(List<Product> filteredProducts){
        Map<String, Integer> numberOfProductsPerYear = new HashMap<String, Integer>();
        for(Product product: filteredProducts){
            if(numberOfProductsPerYear.get(String.valueOf(product.getYearOfProduction())) != null){
                Integer initialVal =
numberOfProductsPerYear.get(String.valueOf(product.getYearOfProduction()));
```

```
            numberOfProductsPerYear.put(String.valueOf(product.getYearOfProduction()), initialVal+1);
        } else {
            numberOfProductsPerYear.put(String.valueOf(product.getYearOfProduction()), 1);
        }
    }
    return numberOfProductsPerYear;
  }
  public Map<String, Integer> getNumberOfProductsPerType(List<Product> filteredProducts){
    Map<String, Integer> numberOfProductsPerType = new HashMap<String, Integer>();

    for(Product product: filteredProducts){
        if(numberOfProductsPerType.get(product.getmedType()) != null){
            Integer initialVal = numberOfProductsPerType.get(product.getmedType());
            numberOfProductsPerType.put(product.getmedType(), initialVal+1);
        } else {
            numberOfProductsPerType.put(product.getmedType(), 1);
        }
    }
    return numberOfProductsPerType;
  }
}
```

```
package com.me.project.service;

import com.me.project.model.Product;

import com.me.project.persistence.ProductPersistence;

import com.me.project.rest.RestPersistence;

import com.me.project.rest.Review;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import java.io.File;

import java.io.FileWriter;

import java.io.IOException;

import java.net.URI;

import java.net.URISyntaxException;

import java.util.AbstractMap;

import java.util.List;

@Service

public class Prod_services {

   private static final Logger logger = LoggerFactory.getLogger(Prod_services.class);

   private static final int DEFAULT_CATEGORY_ID = 1;

   private static final int DEFAULT_STOCK = 0;

   @Autowired

   private ProductPersistence productPersistence;
```

```
@Autowired
private RestPersistence restPersistence;
@Autowired
private FileSystemStorageService fileStorage;
public List<Product> getAllProducts() {
   return productPersistence.getAllProducts();
}
public void addProduct(Product product) {
   //todo: transform Enum to string
   productPersistence.addProduct(product);
}
public AbstractMap.SimpleEntry<Double, List<Review>> getReviews(Integer id) {
   Double average = (double) 0;
   int numberOfReviews = 0;
   List<Review> list = restPersistence.getReviews(id);
   for (Review review : list) {
      average += review.getRating();
      numberOfReviews++;
   }
   if (average > 0 && numberOfReviews > 0) {
      average /= numberOfReviews;
      average = Math.round(average * 10.0) / 10.0;
      return new AbstractMap.SimpleEntry(average, list);
```

```
    } else {
        return new AbstractMap.SimpleEntry(0, list);
    }
}
public Product getProduct(Integer id) {
    return productPersistence.getProduct(id);
}
public void updateProduct(Product product) {
    productPersistence.update(product);
}
public void deleteProduct(String id) {
    Product product = getProduct(Integer.valueOf(id));
    productPersistence.deleteProduct(product);
}
public String getProductPicture(int id) {
    return productPersistence.getProductPicture(id);
}
public File exportProducts(String listOfids) throws URISyntaxException {
    String[] productIds;
    productIds = listOfids.split(",");
    String fileName = "exportedProducts.csv";
    File file = null;
    try {
```

```
String filePath = getClass().getClassLoader().getResource("../../resources").toString();
String filePathURI = new URI(filePath).getPath();
file = new File(filePathURI + fileName);
file.createNewFile();
FileWriter fileWriter = new FileWriter(file);
for (int i = 0; i < productIds.length; i++) {
   Product p = getProduct(Integer.parseInt(productIds[i]));
   fileWriter.append(p.getName());
   fileWriter.append(",");
   fileWriter.append(p.getBrand());
   fileWriter.append(",");
   fileWriter.append(p.getmedType());
   fileWriter.append(",");
   fileWriter.append(p.getCountryOfOrigin());
   fileWriter.append("," + p.getYearOfProduction());
   fileWriter.append("," + p.getVolume());
   fileWriter.append("," + p.getPrice());
   fileWriter.append("," + p.getStock());
   if (p.getDescription() != null) {
      fileWriter.append("," + p.getDescription());
   } else {
      fileWriter.append(", ");
   }
```

```
            if (p.getRecommendations() != null) {
                fileWriter.append("," + p.getRecommendations());
            } else {
                fileWriter.append(", ");
            }
            if (p.getPicture() != null) {
                fileWriter.append("," + p.getPicture());
            } else {
                fileWriter.append(", ");
            }
            if (i != productIds.length - 1)
                fileWriter.append("\n");
        }
        fileWriter.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return file;
}
public int manageImports(String fileData) {
    int nbOfImportedProducts = 0;
    String[] lines = fileData.split("\n");
    for (int i = 0; i < lines.length; i++) {
```

```
boolean valid = true;
String[] myLine = lines[i].split(",");
Product newP = new Product();
for (int j = 0; j < myLine.length; j++) {
   if (!myLine[j].equals("")) {
      setProduct_Info(newP, j, myLine[j]);
   } else {
      if (j <= 6) {
         valid = false;
         break;
      }
   }
}
if (valid && myLine.length > 6) {
   logger.info(newP.toString());
   if (newP.getCategoryID() == null) {
      newP.setCategoryID(DEFAULT_CATEGORY_ID);
   }
   if (newP.getStock() == null) {
      newP.setStock(DEFAULT_STOCK);
   }
   try {
      String fileName = fileStorage.storeDefault();
```

```
                newP.setPicture(fileName);
            } catch (IOException exp) {
                logger.debug("pictureExceptionDefault: The default picture could not be added also!");
            }
            productPersistence.addProduct(newP);
            nbOfImportedProducts++;
        }
    }
    return nbOfImportedProducts;
}
private void setProduct_Info(Product newP, int j, String info) {
    switch (j) {
        case 0:
            newP.setName(info);
            break;
        case 1:
            newP.setBrand(info);
            break;
        case 2:
            newP.setmedType(info);
            break;
        case 3:
            newP.setCountryOfOrigin(info);
```

```
            break;
        case 4:
            newP.setYearOfProduction(Integer.parseInt(info));
            break;
        case 5:
            newP.setVolume(Double.parseDouble(info));
            break;
        case 6:
            newP.setPrice(Double.parseDouble(info));
            break;
        case 7:
            newP.setStock(Integer.parseInt(info));
            break;
        case 8:
            newP.setDescription(info);
            break;
        case 9:
            newP.setRecommendations(info);
            break;
    }
}
public List<Product> sortProducts(String sortingCriteria, Integer startIndex, Integer maxNb) {
    return productPersistence.sortProducts(sortingCriteria, startIndex, maxNb);
```

```
    }

    public void deleteAll(String products) {

        productPersistence.deleteAll(products);

    }

}

package com.me.project.service;

import com.me.project.Util.SendEmail;

import com.me.project.interfaces.IMedi_user_service;

import com.me.project.model.users.User;

import com.me.project.model.users.UserDAO;

import com.me.project.persistence.UserPersistence;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.security.core.Authentication;

import org.springframework.security.core.context.SecurityContextHolder;

import org.springframework.stereotype.Service;

import java.util.List;

import java.util.UUID;

@Service

public class Medi_user_service implements IMedi_user_service {

    @Autowired

    private UserPersistence userPersistence;
```

```
    @Override
    public void registerNewUserAccount(UserDAO accountDAO) {
        User user = new User();
        user.setName(accountDAO.getName());
        user.setPassword(accountDAO.getPassword());
        user.setEmail(accountDAO.getEmail());
        user.setPhoneNumber(Long.parseLong(accountDAO.getPhoneNumber()));
        userPersistence.add(user);
        String token = UUID.randomUUID().toString();
        SendEmail.sendMethod(user.getEmail(), user.getId(),token);
    }
    @Override
    public boolean login(User user) {
        List<User> users = userPersistence.getAll();
        boolean found = false;
        for (User existingUser : users) {
            if (existingUser.getEmail().equals(user.getEmail()) &&
existingUser.getPassword().equals(user.getPassword()) && existingUser.getActive()) {
                found = true;
                break;
            }
        }
```

```
    return found;
}
@Override
public void addUser(User user){
    userPersistence.add(user);
}
@Override
public void activatingUserAccount(Integer id) {
    User user = userPersistence.getUser(id);
    if (user != null)
        user.setActive(true);
        userPersistence.update(user);
}
public List<User> getAll() {
    return userPersistence.getAll();
}
public User getUserByEmail(String email) {
    List<User> users = getAll();
    User foundUser = null;
    for (User user: users) {
        if (user.getEmail().equals(email)) {
            foundUser = user;
        }
```

```
        }
        return foundUser;
    }
    public User getAuthenticatedUser() {
        Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
        String email = authentication.getPrincipal().toString();
        return getUserByEmail(email);
    }
}
package com.me.project.service.convertors;
import org.springframework.stereotype.Component;
import com.me.project.model.*;
import java.util.ArrayList;
import java.util.List;
@Component
public class DAOConverter {
    public Product convertDAOToProduct(ProductDAO productDAO) {
        Product product = new Product();
        product.setName(productDAO.getName());
        product.setBrand(productDAO.getBrand());
        product.setPrice(productDAO.getPrice());
        if (productDAO.getStock() == null) {
            product.setStock(0);
```

```
    } else {
        product.setStock(productDAO.getStock());
    }
    product.setmedType(productDAO.getMedType());
    product.setCountryOfOrigin(productDAO.getCountryOfOrigin());
    product.setYearOfProduction(productDAO.getYearOfProduction());
    product.setVolume(productDAO.getVolume());
    product.setDescription(productDAO.getDescription());
    product.setRecommendations(productDAO.getRecommendations());
    product.setMediId(productDAO.getMediId());
    product.setCategoryID(productDAO.getCategoryID());
    return product;
}
public ProductDAO convertProductToDAO(Product product) {
    ProductDAO productDAO = new ProductDAO();
    productDAO.setName(product.getName());
    productDAO.setMediId(product.getMediId());
    productDAO.setBrand(product.getBrand());
    productDAO.setPrice(product.getPrice());
    productDAO.setStock(product.getStock());
    productDAO.setMedType(product.getmedType());
    productDAO.setCountryOfOrigin(product.getCountryOfOrigin());
    productDAO.setVolume(product.getVolume());
```

```
    productDAO.setRecommendations(product.getRecommendations());
    productDAO.setYearOfProduction(product.getYearOfProduction());
    productDAO.setDescription(product.getDescription());
    productDAO.setCategoryID(product.getCategoryID());
    return productDAO;
}
public CartDAO convertCartToCartDAO(Cart cart){
    CartDAO cartDAO = new CartDAO();
    List<CartEntry> listOfEntries = cart.getListOfEntries();
    List<CartEntryDAO> listOfCartEntryDAO = new ArrayList<>();
    for(CartEntry cartEntry : listOfEntries){
        CartEntryDAO cartTemp = new CartEntryDAO();
        cartTemp.setCart(cartEntry.getCart());
        cartTemp.setEntryId(cartEntry.getEntryId());
        cartTemp.setProduct(cartEntry.getProduct());
        cartTemp.setQuantity(cartEntry.getQuantity());
        listOfCartEntryDAO.add(cartTemp);
    }
    cartDAO.setListOfEntries(listOfCartEntryDAO);
    cartDAO.setTotalCost(cart.getTotalCost());
    cartDAO.setCartId(cart.getCartId());
    cartDAO.setTOTAL();
```

```
        return cartDAO;

    }

}

package com.me.project.service.validatorss;

import org.springframework.stereotype.Component;

import org.springframework.validation.Errors;

import org.springframework.validation.ValidationUtils;

import org.springframework.validation.Validators;

import com.me.project.model.ProductDAO;

import java.util.Calendar;

@Component

public class ProducTotallidators implements Validators {

    public static final String IS_REQUIRED = " is required!";

    public void validate(Object object, Errors errors) {

        ProductDAO product = (ProductDAO) object;

        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "name", "name.empty.error", "Name" +
IS_REQUIRED);

      ValidationUtils.rejectIfEmptyOrWhitespace(errors, "brand", "brand.empty.error", "Brand" +
IS_REQUIRED);

        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "medType", "type.empty.error", "Type" +
IS_REQUIRED);

        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "countryOfOrigin", "country.empty.error",
"Country of origin" + IS_REQUIRED);

        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "price", "price.empty.error", "Price" +
IS_REQUIRED);
```

```
    ValidationUtils.rejectIfEmptyOrWhitespace(errors, "volume", "volume.empty.error", "Volume" +
IS_REQUIRED);

    ValidationUtils.rejectIfEmptyOrWhitespace(errors, "yearOfProduction", "year.empty.error", "Year
of production" + IS_REQUIRED);

    Double price = product.getPrice();

    if (price != null && price <= 0) {

      errors.rejecTotallue("price", "price.incorrect.error", new Object[]{"args"}, "Price must be greater
than 0!");

    }

    Integer stock = product.getStock();

    if (stock != null && stock < 0) {

      errors.rejecTotallue("stock", "stock.incorrect.error", new Object[]{"args"}, "Stock must be greater
or equal with 0!");

    }

    Double volume = product.getVolume();

    if (volume != null && volume < 0) {

      errors.rejecTotallue("volume", "volume.incorrect.error", new Object[]{"args"}, "Volume must be
greater than 0!");

    }

    int currentYear = Calendar.getInstance().get(Calendar.YEAR);

    Integer yearOfProduction = product.getYearOfProduction();

    if ((yearOfProduction != null) && (yearOfProduction < 0 || yearOfProduction > currentYear)) {

      errors.rejecTotallue("yearOfProduction", "year.incorrect.error", new Object[]{"args"}, "Year of
production must be lower than the currrent year and greater than 0!");

  }  }  public boolean supports(Class<?> aClass) {

    return ProductDAO.class.equals(aClass);  }}
```