Contenido

[XSS 2](#_gjdgxs)

[SQL INYECTION 5](#_30j0zll)

[Explicación de clases 7](#_1fob9te)

[ERROR COMPRA PRODUCTOS NO BAJA CONTADOR 8](#_3znysh7)

[ERROR NO BAJA CONTADOR ORDERCONTROLLER.JAVA 12](#_2et92p0)

[Usuario base de datos root: 15](#_tyjcwt)

[HTTPS: 19](#_3dy6vkm)

[FILE UPLOAD: 22](#_1t3h5sf)

[Valorar productos que no compraste: 24](#_4d34og8)

[Validación de entidades: 26](#_2s8eyo1)

[Cookie segura: 33](#_17dp8vu)

[Limitar número errores: 34](#_3rdcrjn)

[Demasiado información al loguearse: 35](#_26in1rg)

# XSS

<!DOCTYPE HTML>

<html>

   <header th:replace="layout/Includes :: #include-fragment"></header>

   <body>

      <div th:replace="layout/NavBar :: #navbar-fragment"></div>

      <script>

         $(document).ready(function () {

             doConfigureFormValidation();

         });

      </script>

      <div class="container login">

         <div class="row w-100">

            <div class="col col-padding">

               <div class="card mx-auto bg-light center-block ui-helper-margin-top">

                  <div class="card-body">

                     <div class="row">

                        <div class="col-12 form-title">

                           <h2 th:utext="#{send.email.title}"></h2>

                        </div>

                     </div>

                     <div th:replace="layout/Messages :: #messages-fragment"></div>

                     <form  class="form-horizontal toggle-disabled" method="POST">

                        <div class="form-group row">

                           <label for="email" class="d-none d-sm-block col-sm-3 text-right input-label-middle"

                                  th:utext="#{profile.input.email}"></label>

                           <div class="input-group col-12 col-sm-6">

                              <div class="input-group-prepend">

                                 <span class="input-group-text"><i class="fas fa-at"></i></span>

                              </div>

                              <input type="email" class="form-control" id="email" name="email"

                                     th:placeholder="#{profile.input.email.placeholder}"

                                     data-validation-error-msg-container="#input-email-error"

                                     data-validation="email"

                                     th:attr="data-validation-error-msg=#{profile.input.email.validation}">

                           </div>

                        </div>

                        <div class="form-group row d-none">

                           <div class="d-none d-sm-block col-sm-3"></div>

                           <div class="col-12 col-sm-6 text-left" id="input-email-error"></div>

                        </div>

                        <div class="form-group row">

                           <div class="d-none d-sm-block col-sm-3"></div>

                           <div class="d-inline col-12 col-sm-6">

                              <button type="submit" class="btn btn-primary">

                                 <i class="fa fa-envelope"></i>

                                 <span th:utext="#{send.email.button}"></span>

                              </button>

                           </div>

                        </div>

                     </form>

                  </div>

               </div>

            </div>

         </div>

      </div>

   </div>

   <div th:replace="layout/Footer :: #footer-fragment"></div>

</body>

</html>

Para aumentar la seguridad contra ataques de XSS en este código, se deben aplicar las siguientes medidas: 1. Reemplazar "th:utext" por "th:text" para escapar automáticamente cualquier contenido HTML y evitar inyecciones de código:

html

<h2 th:text="#{send.email.title}"></h2>

html

<label for="email" class="d-none d-sm-block col-sm-3 text-right input-label-middle"

th:text="#{profile.input.email}"></label>

html

<span th:text="#{send.email.button}"></span>

2. Utilizar Content Security Policy (CSP) en el encabezado de la página para limitar la ejecución de recursos de terceros como scripts, imágenes, estilos, etc. Ejemplo de directiva CSP en el encabezado HTML:

html

<head>

<meta http-equiv="Content-Security-Policy" content="default-src 'self'; script-src 'self'; style-src 'self'; img-src 'self'; font-src 'self';">

</head>

html

<input type="email" class="form-control" id="email" name="email" autocomplete="off"

th:placeholder="#{profile.input.email.placeholder}"

data-validation-error-msg-container="#input-email-error"

data-validation="email"

th:attr="data-validation-error-msg=#{profile.input.email.validation}">

4. Asegurarse de que todas las peticiones al servidor sean enviadas usando métodos seguros como HTTPS en lugar de HTTP. 5. Para prevenir ataques CSRF, se debe incluir un token CSRF en el formulario. Este token puede ser generado por el servidor y añadido al formulario con el elemento input oculto:

5. Para prevenir ataques CSRF, se debe incluir un token CSRF en el formulario. Este token puede ser generado por el servidor y añadido al formulario con el elemento input oculto:

html

<form class="form-horizontal toggle-disabled" method="POST" th:action="@{/send-email}">

<input type="hidden" name="${\_csrf.parameterName}" th:value="${\_csrf.token}" />

...

</form>

# SQL INYECTION

package es.storeapp.business.repositories;

import es.storeapp.business.entities.User;

import java.text.MessageFormat;

import javax.persistence.NoResultException;

import javax.persistence.Query;

import org.springframework.stereotype.Repository;

@Repository

public class UserRepository extends AbstractRepository<User> {

    private static final String FIND\_USER\_BY\_EMAIL\_QUERY = "SELECT u FROM User u WHERE u.email = ''{0}''";

    private static final String COUNT\_USER\_BY\_EMAIL\_QUERY = "SELECT COUNT(\*) FROM User u WHERE u.email = ''{0}''";

    private static final String LOGIN\_QUERY = "SELECT u FROM User u WHERE u.email = ''{0}'' AND u.password = ''{1}''";

    public User findByEmail(String email) {

        try {

            Query query = entityManager.createQuery(MessageFormat.format(FIND\_USER\_BY\_EMAIL\_QUERY, email));

            return (User) query.getSingleResult();

        } catch (NoResultException e) {

            logger.error(e.getMessage(), e);

            return null;

        }

    }

    public boolean existsUser(String email) {

        Query query = entityManager.createQuery(MessageFormat.format(COUNT\_USER\_BY\_EMAIL\_QUERY, email));

        return ((Long) query.getSingleResult() > 0);

    }

    public User findByEmailAndPassword(String email, String password) {

        try {

            Query query = entityManager.createQuery(MessageFormat.format(LOGIN\_QUERY, email, password));

            return (User) query.getSingleResult();

        } catch (NoResultException e) {

            logger.error(e.getMessage(), e);

            return null;

        }

    }

}

Para evitar la vulnerabilidad de inyección SQL en esta clase, se deben utilizar parámetros de consulta en lugar de concatenar los valores directamente en las consultas. Para hacer esto, se pueden utilizar los métodos setParameter de la clase Query, en lugar de utilizar la concatenación de cadenas directamente en la consulta. Por ejemplo, la consulta FIND\_USER\_BY\_EMAIL\_QUERY se puede modificar de la siguiente manera:

private static final String FIND\_USER\_BY\_EMAIL\_QUERY = "SELECT u FROM User u WHERE u.email = :email";

Luego, en el método findByEmail, se puede utilizar el método setParameter para establecer el valor del parámetro:

public User findByEmail(String email) {

try {

Query query = entityManager.createQuery(FIND\_USER\_BY\_EMAIL\_QUERY);

query.setParameter("email", email);

return (User) query.getSingleResult();

} catch (NoResultException e) {

logger.error(e.getMessage(), e);

return null;

}

}

De esta manera, se evita la vulnerabilidad de inyección SQL al no permitir que los valores ingresados por el usuario se concatenen directamente en la consulta.

# Explicación de clases

BUSINESS:

Entities -> Los objetos de cada clase

Exceptions -> Las excepciones

Reposritories -> Los repositorios

Services -> Los servicios

WEB:

Controller -> controladores

Forms -> formularios en java

Los formularios html están en resources/templates/

# ERROR COMPRA PRODUCTOS NO BAJA CONTADOR

package es.storeapp.web.controller;

import es.storeapp.business.entities.Product;

import es.storeapp.business.exceptions.InstanceNotFoundException;

import es.storeapp.business.services.ProductService;

import es.storeapp.common.Constants;

import es.storeapp.web.exceptions.ErrorHandlingUtils;

import es.storeapp.web.session.ShoppingCart;

import java.text.MessageFormat;

import java.util.List;

import java.util.Locale;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.context.MessageSource;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

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

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

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

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

import org.springframework.web.servlet.mvc.support.RedirectAttributes;

@Controller

public class ShoppingCartController {

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

    @Autowired

    private ProductService productService;

    @Autowired

    private MessageSource messageSource;

    @Autowired

    ErrorHandlingUtils exceptionHandlingUtils;

    @GetMapping(value = {Constants.CART\_ENDPOINT})

    public String doGetCartPage(@SessionAttribute(Constants.SHOPPING\_CART\_SESSION) ShoppingCart shoppingCart,

                                Model model,

                                Locale locale) {

        return Constants.SHOPPING\_CART\_PAGE;

    }

    @PostMapping(value = {Constants.CART\_ADD\_PRODUCT\_ENDPOINT})

    public String doAddProductToCart(@PathVariable() Long id,

                                     @SessionAttribute(Constants.SHOPPING\_CART\_SESSION) ShoppingCart shoppingCart,

                                     RedirectAttributes redirectAttributes,

                                     Model model,

                                     Locale locale) {

        Product product;

        try {

            product = productService.findProductById(id);

            List<Product> products = shoppingCart.getProducts();

            if(logger.isDebugEnabled()) {

                logger.debug(MessageFormat.format("Adding product {0} to shopping cart", id));

            }

            for (Product p : products) {

                if (p.getProductId().equals(id)) {

                    String message = messageSource.getMessage(Constants.PRODUCT\_ALREADY\_IN\_SHOPPING\_CART,

                        new Object[]{p.getName()}, locale);

                    redirectAttributes.addFlashAttribute(Constants.ERROR\_MESSAGE, message);

                    return Constants.SEND\_REDIRECT + Constants.CART\_ENDPOINT;

                }

            }

            String message = messageSource.getMessage(Constants.PRODUCT\_ADDED\_TO\_THE\_SHOPPING\_CART,

                new Object[]{product.getName()}, locale);

            redirectAttributes.addFlashAttribute(Constants.SUCCESS\_MESSAGE, message);

            products.add(product);

        } catch (InstanceNotFoundException ex) {

            return exceptionHandlingUtils.handleInstanceNotFoundException(ex, model, locale);

        }

        return Constants.SEND\_REDIRECT + Constants.CART\_ENDPOINT;

    }

    @PostMapping(value = {Constants.CART\_REMOVE\_PRODUCT\_ENDPOINT})

    public String doRemoveProductFromCart(@PathVariable() Long id,

                                          @SessionAttribute(Constants.SHOPPING\_CART\_SESSION) ShoppingCart shoppingCart,

                                          RedirectAttributes redirectAttributes,

                                          Model model,

                                          Locale locale ) {

        Product product;

        try {

            product = productService.findProductById(id);

            List<Product> products = shoppingCart.getProducts();

            for (Product p : products) {

                if (p.getProductId().equals(id)) {

                    String message = messageSource.getMessage(Constants.PRODUCT\_REMOVED\_FROM\_THE\_SHOPPING\_CART,

                        new Object[]{product.getName()}, locale);

                    redirectAttributes.addFlashAttribute(Constants.SUCCESS\_MESSAGE, message);

                    products.remove(p);

                    return Constants.SEND\_REDIRECT + Constants.CART\_ENDPOINT;

                }

            }

            String message = messageSource.getMessage(Constants.PRODUCT\_NOT\_IN\_SHOPPING\_CART,

                new Object[]{product.getName()}, locale);

            redirectAttributes.addFlashAttribute(Constants.ERROR\_MESSAGE, message);

        } catch (InstanceNotFoundException ex) {

            return exceptionHandlingUtils.handleInstanceNotFoundException(ex, model, locale);

        }

        return Constants.SEND\_REDIRECT + Constants.CART\_ENDPOINT;

    }

}

Para solucionar este problema, podemos agregar un nuevo método en el controlador ShoppingCartController para manejar la cancelación de la compra del producto. Este método se activará cuando el usuario haga clic en el botón de cancelar en la interfaz de usuario. A continuación, se muestra el código modificado en el controlador:

// Añadir este import al comienzo de la clase

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

// Añadir este nuevo método en la clase ShoppingCartController

@DeleteMapping(value = {Constants.CART\_CANCEL\_PRODUCT\_ENDPOINT})

public String doCancelProductPurchase(@PathVariable() Long id,

@SessionAttribute(Constants.SHOPPING\_CART\_SESSION) ShoppingCart shoppingCart,

RedirectAttributes redirectAttributes,

Model model,

Locale locale) {

List<Product> products = shoppingCart.getProducts();

Product productToCancel = null;

for (Product product : products) {

if (product.getProductId().equals(id)) {

productToCancel = product;

break;

}

}

if (productToCancel != null) {

products.remove(productToCancel);

String message = messageSource.getMessage(Constants.PRODUCT\_PURCHASE\_CANCELLED,

new Object[]{productToCancel.getName()}, locale);

redirectAttributes.addFlashAttribute(Constants.SUCCESS\_MESSAGE, message);

} else {

String message = messageSource.getMessage(Constants.PRODUCT\_NOT\_IN\_SHOPPING\_CART,

new Object[]{id}, locale);

redirectAttributes.addFlashAttribute(Constants.ERROR\_MESSAGE, message);

}

return Constants.SEND\_REDIRECT + Constants.CART\_ENDPOINT;

}

También necesitarás agregar una nueva constante para la ruta de cancelación del producto en la clase Constants:

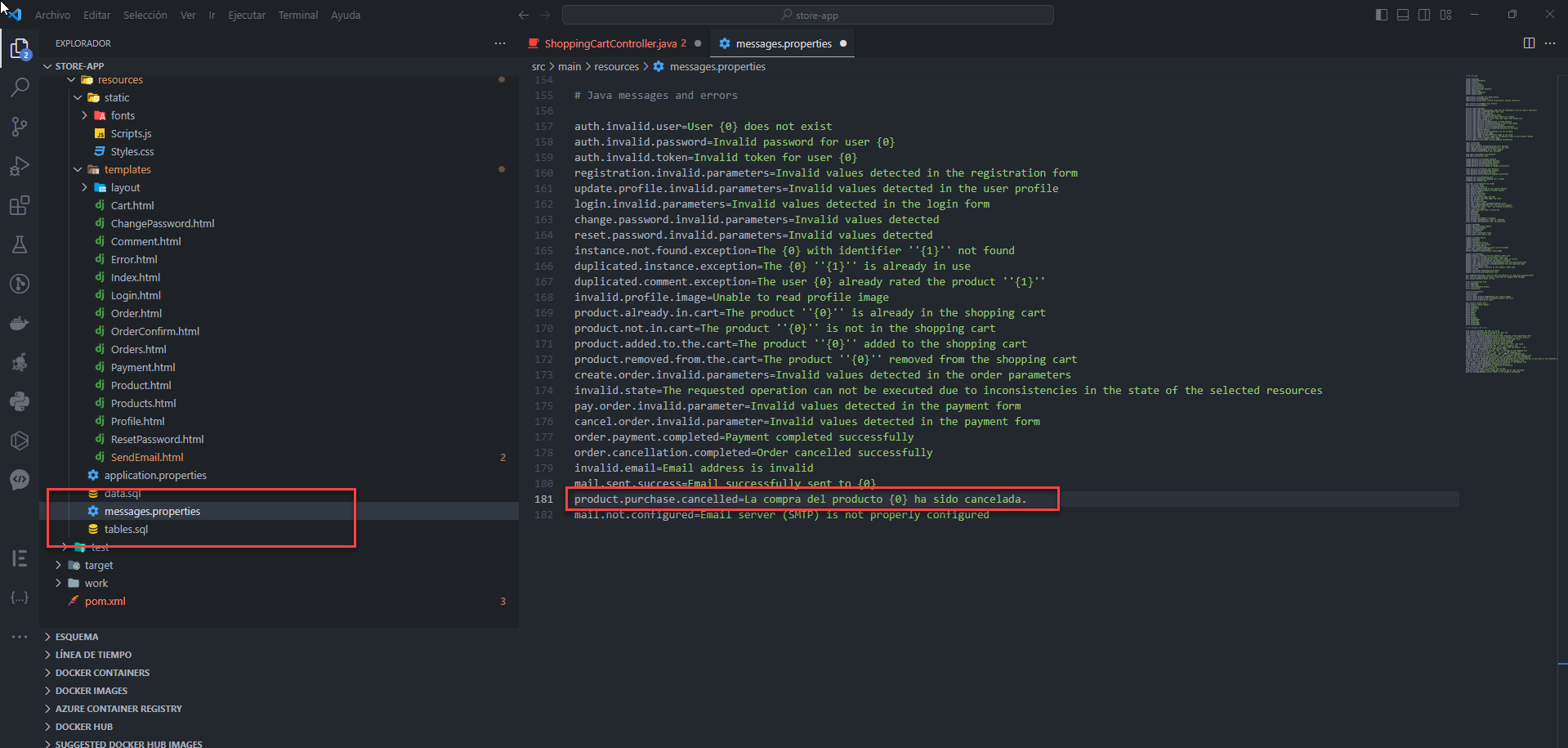
public static final String CART\_CANCEL\_PRODUCT\_ENDPOINT = "/cart/cancel/{id}";

Y agregar las entradas correspondientes en el archivo de mensajes para los mensajes de éxito y error:

product.purchase.cancelled=La compra del producto {0} ha sido cancelada.

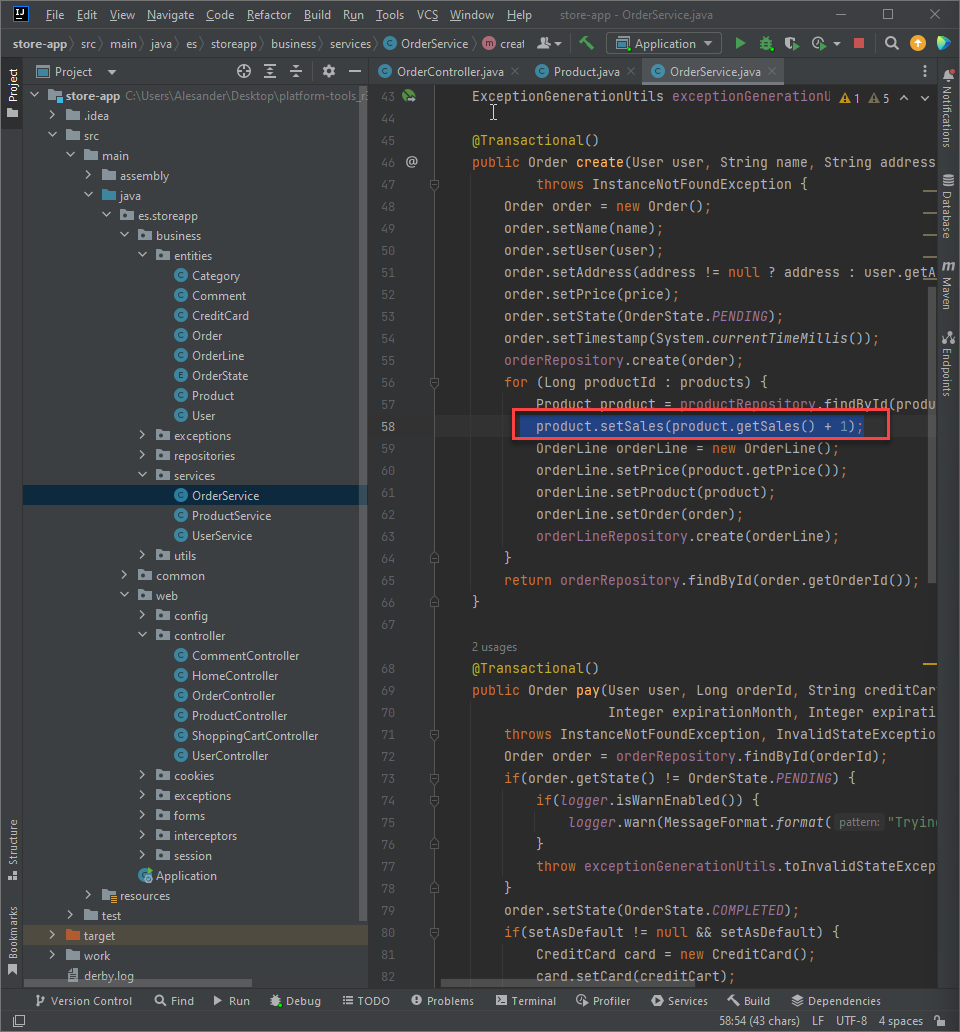
Explicación en español: Se ha agregado un nuevo método llamado doCancelProductPurchase , que se activa mediante una solicitud DELETE en la ruta Constants.CART\_CANCEL\_PRODUCT\_ENDPOINT . En este método, buscamos el producto en el carrito de compras según el id proporcionado en la ruta. Si encontramos el producto, lo eliminamos del carrito y mostramos un mensaje de éxito al usuario. Si el producto no está en el carrito, mostramos un mensaje de error al usuario.

Añadiendo aquí elmensaje de error:



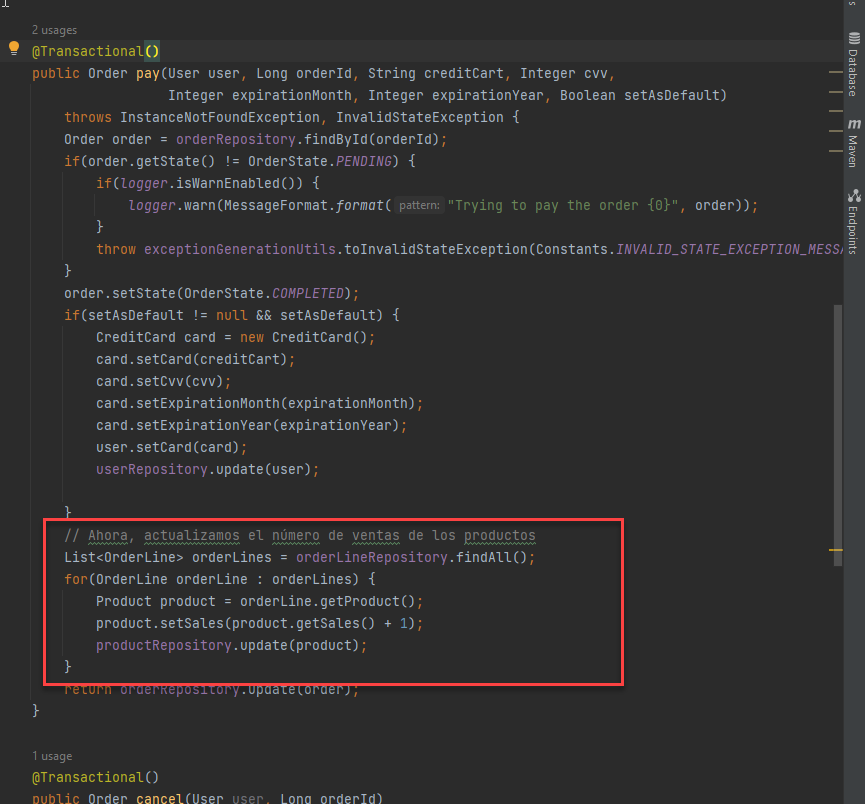
# ERROR NO BAJA CONTADOR ORDERCONTROLLER.JAVA

Para modificar esto en la clase es.storeapp/bussiness/services/OrderService cambio esta línea :



De este método al método de pagar para que solo se haga cuando esté completamente pagado:

@Transactional()  
public Order pay(User user, Long orderId, String creditCart, Integer cvv,   
 Integer expirationMonth, Integer expirationYear, Boolean setAsDefault)  
 throws InstanceNotFoundException, InvalidStateException {  
 Order order = orderRepository.findById(orderId);  
 if(order.getState() != OrderState.*PENDING*) {  
 if(*logger*.isWarnEnabled()) {  
 *logger*.warn(MessageFormat.*format*("Trying to pay the order {0}", order));  
 }  
 throw exceptionGenerationUtils.toInvalidStateException(Constants.*INVALID\_STATE\_EXCEPTION\_MESSAGE*);  
 }  
 order.setState(OrderState.*COMPLETED*);  
 if(setAsDefault != null && setAsDefault) {  
 CreditCard card = new CreditCard();  
 card.setCard(creditCart);  
 card.setCvv(cvv);  
 card.setExpirationMonth(expirationMonth);  
 card.setExpirationYear(expirationYear);  
 user.setCard(card);  
 userRepository.update(user);  
  
 }  
 // Ahora, actualizamos el número de ventas de los productos  
 List<OrderLine> orderLines = orderLineRepository.findAll();  
 for(OrderLine orderLine : orderLines) {  
 Product product = orderLine.getProduct();  
 product.setSales(product.getSales() + 1);  
 productRepository.update(product);  
 }  
 return orderRepository.update(order);  
}

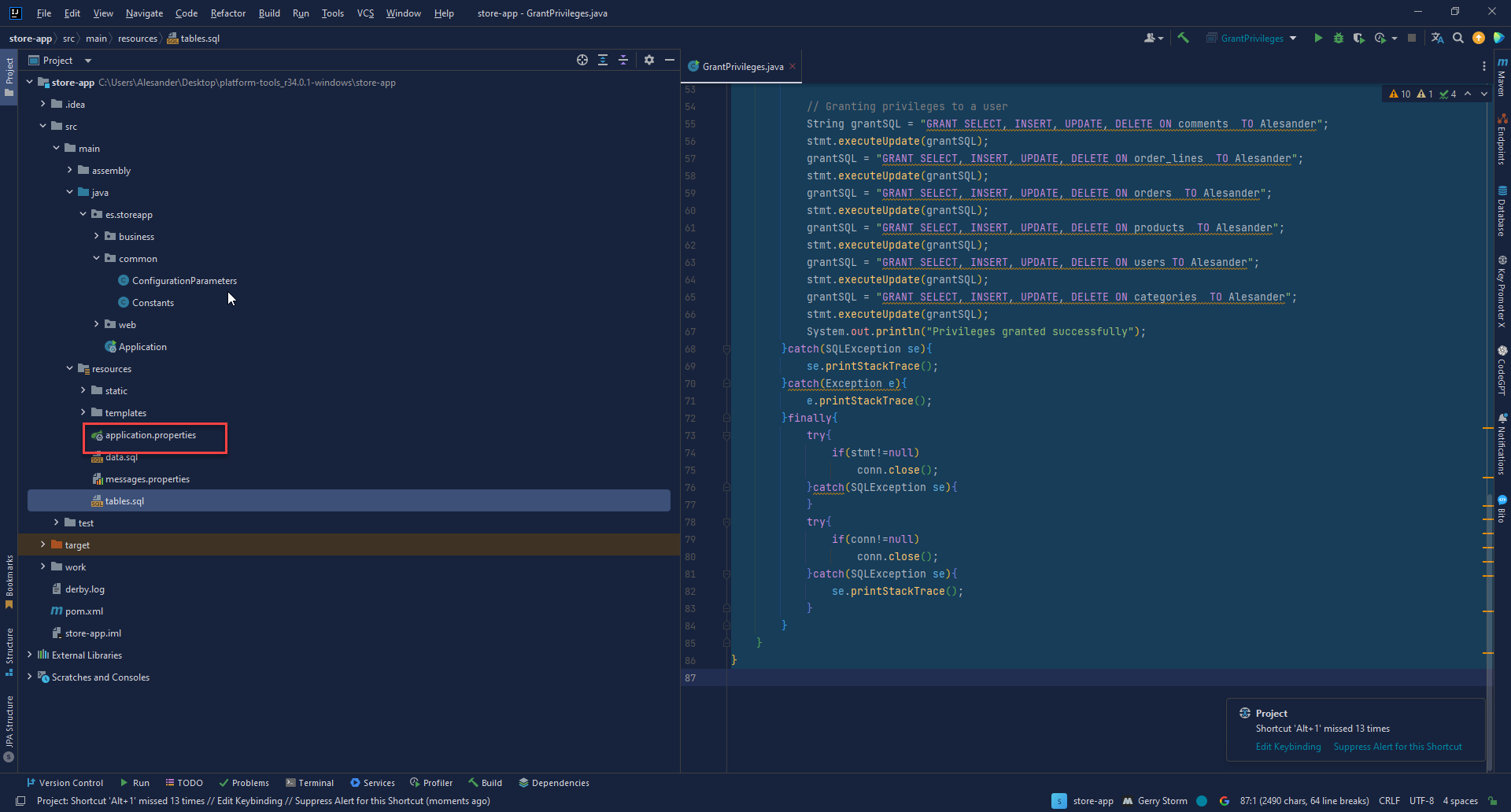


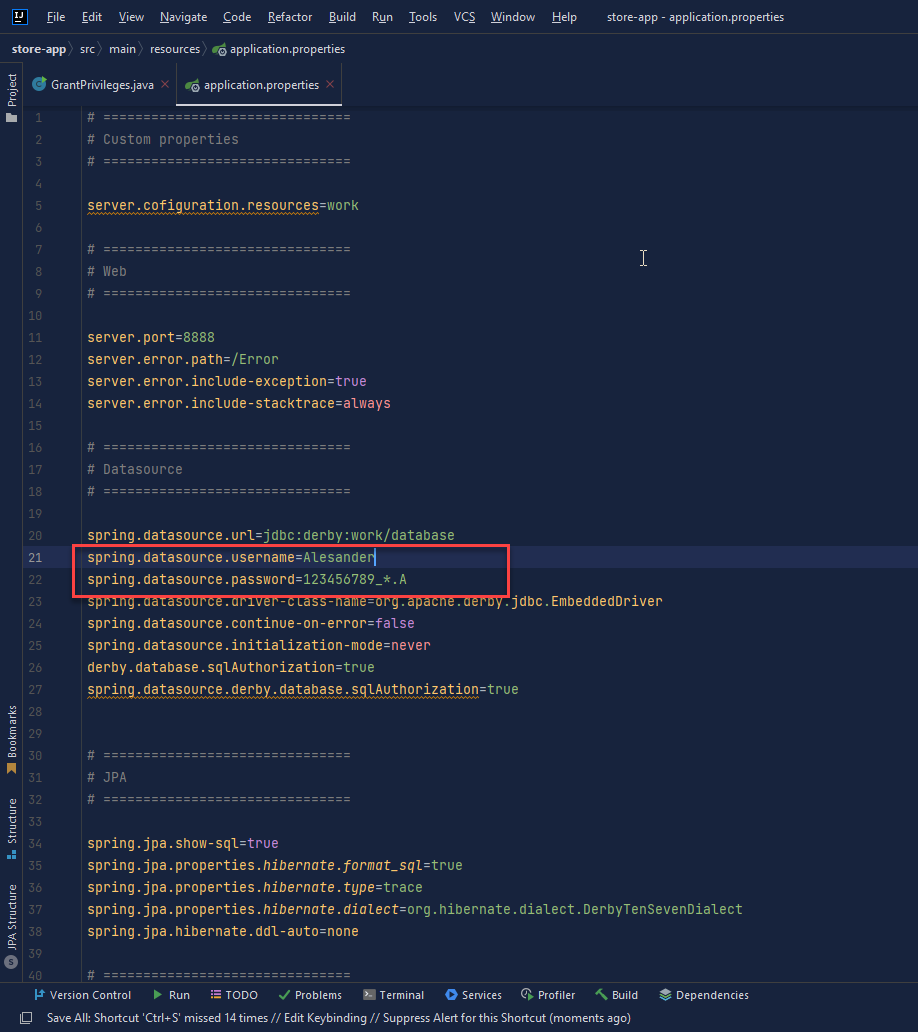
# Usuario base de datos root:

Creo esta clase que creará un nuevo uusario y le dará permisos en las tablas:

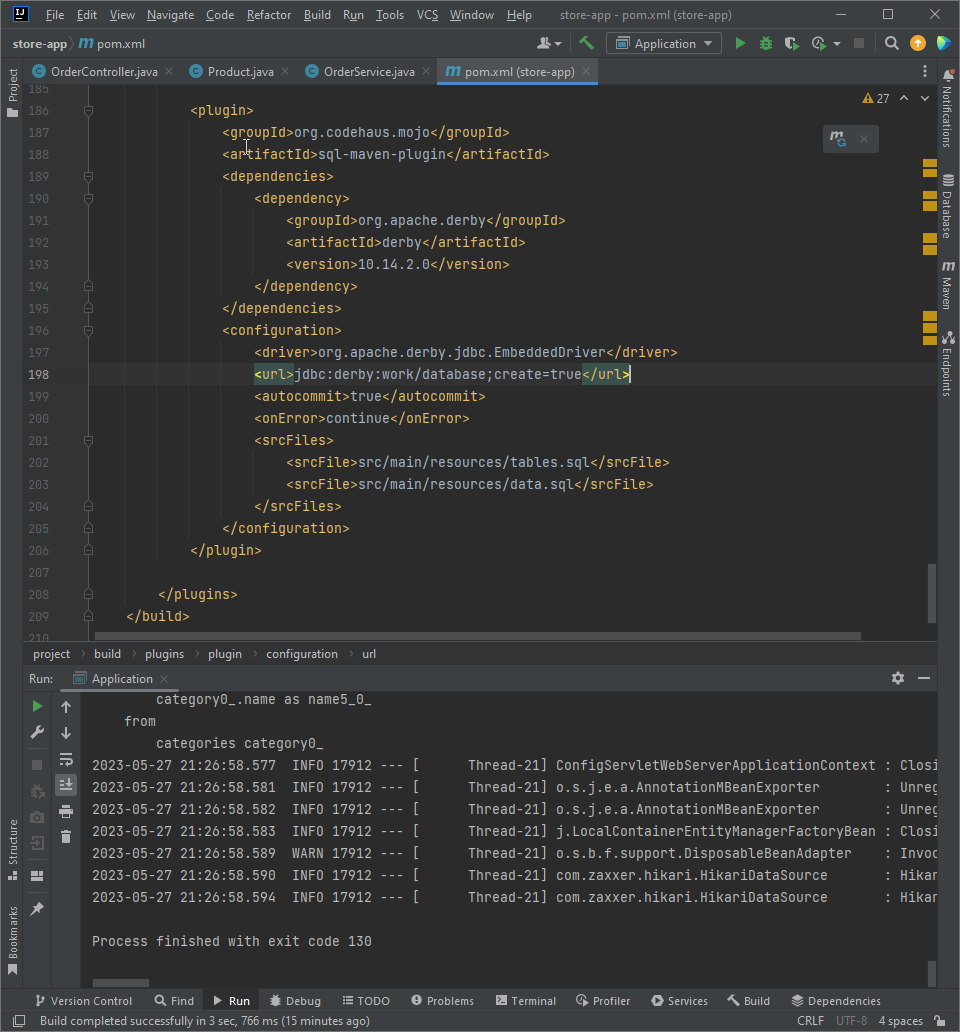
public class GrantPrivileges {  
  
 static final String DB\_URL = "jdbc:derby:work/database;";  
 static final String USER = "Alesander";  
 static final String PASS = "123456789\_\*.A";  
  
 public static void main(String[] args) {  
 Connection conn = null;  
 Statement stmt = null;  
 try{  
 System.setProperty("derby.database.sqlAuthorization", "TRUE");  
 Properties props = new Properties();  
 props.put("user","app");  
 props.put("password","secr3t");  
  
  
 conn = DriverManager.getConnection(DB\_URL,props);  
 stmt = conn.createStatement();  
  
 //Create user  
 CallableStatement cs = conn.prepareCall(  
 "CALL SYSCS\_UTIL.SYSCS\_SET\_DATABASE\_PROPERTY(?, ?)");  
 // cs.setString(1, USER);  
 cs.setString(1, "derby.user." + USER);  
 cs.setString(2, PASS);  
 cs.execute();  
  
 CallableStatement cs2 = conn.prepareCall(  
 "CALL SYSCS\_UTIL.SYSCS\_SET\_DATABASE\_PROPERTY('derby.database.sqlAuthorization', 'TRUE')");  
 cs2.execute();  
  
 // Granting privileges to a user  
 String grantSQL = "GRANT SELECT, INSERT, UPDATE, DELETE ON comments TO Alesander";  
 stmt.executeUpdate(grantSQL);  
 grantSQL = "GRANT SELECT, INSERT, UPDATE, DELETE ON order\_lines TO Alesander";  
 stmt.executeUpdate(grantSQL);  
 grantSQL = "GRANT SELECT, INSERT, UPDATE, DELETE ON orders TO Alesander";  
 stmt.executeUpdate(grantSQL);  
 grantSQL = "GRANT SELECT, INSERT, UPDATE, DELETE ON products TO Alesander";  
 stmt.executeUpdate(grantSQL);  
 grantSQL = "GRANT SELECT, INSERT, UPDATE, DELETE ON users TO Alesander";  
 stmt.executeUpdate(grantSQL);  
 grantSQL = "GRANT SELECT, INSERT, UPDATE, DELETE ON categories TO Alesander";  
 stmt.executeUpdate(grantSQL);  
 System.out.println("Privileges granted successfully");  
 }catch(SQLException se){  
 se.printStackTrace();  
 }catch(Exception e){  
 e.printStackTrace();  
 }finally{  
 try{  
 if(stmt!=null)  
 conn.close();  
 }catch(SQLException se){  
 }  
 try{  
 if(conn!=null)  
 conn.close();  
 }catch(SQLException se){  
 se.printStackTrace();  
 }  
 }  
 }  
}

Y hora modifico el usuario en el archivo de propiedades de la db en :





Y elimino del pom.xml el otro usuario:



# HTTPS:

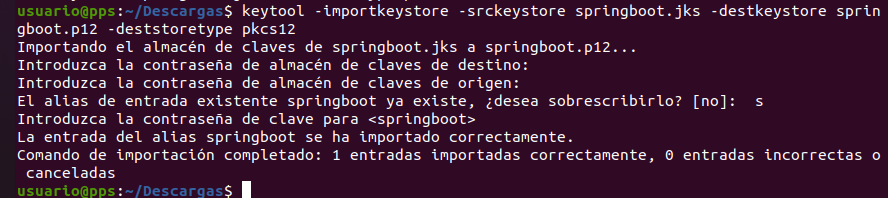
keytool -genkeypair -alias springboot -keyalg RSA -keysize 4096 -storetype JKS -keystore springboot.jks -validity 3650 -storepass password



keytool -genkeypair -alias springboot -keyalg RSA -keysize 4096 -storetype JKS -keystore springboot.jks -validity 3650 -storepass password



keytool -importkeystore -srckeystore springboot.jks -destkeystore springboot.p12 -deststoretype pkcs12



Para habilitar HTTPS para nuestra aplicación Spring Boot, abramos nuestro archivo application.yml (o application.properties) y definamos las siguientes propiedades:

server:

ssl:

key-store: classpath:keystore.p12

key-store-password: password

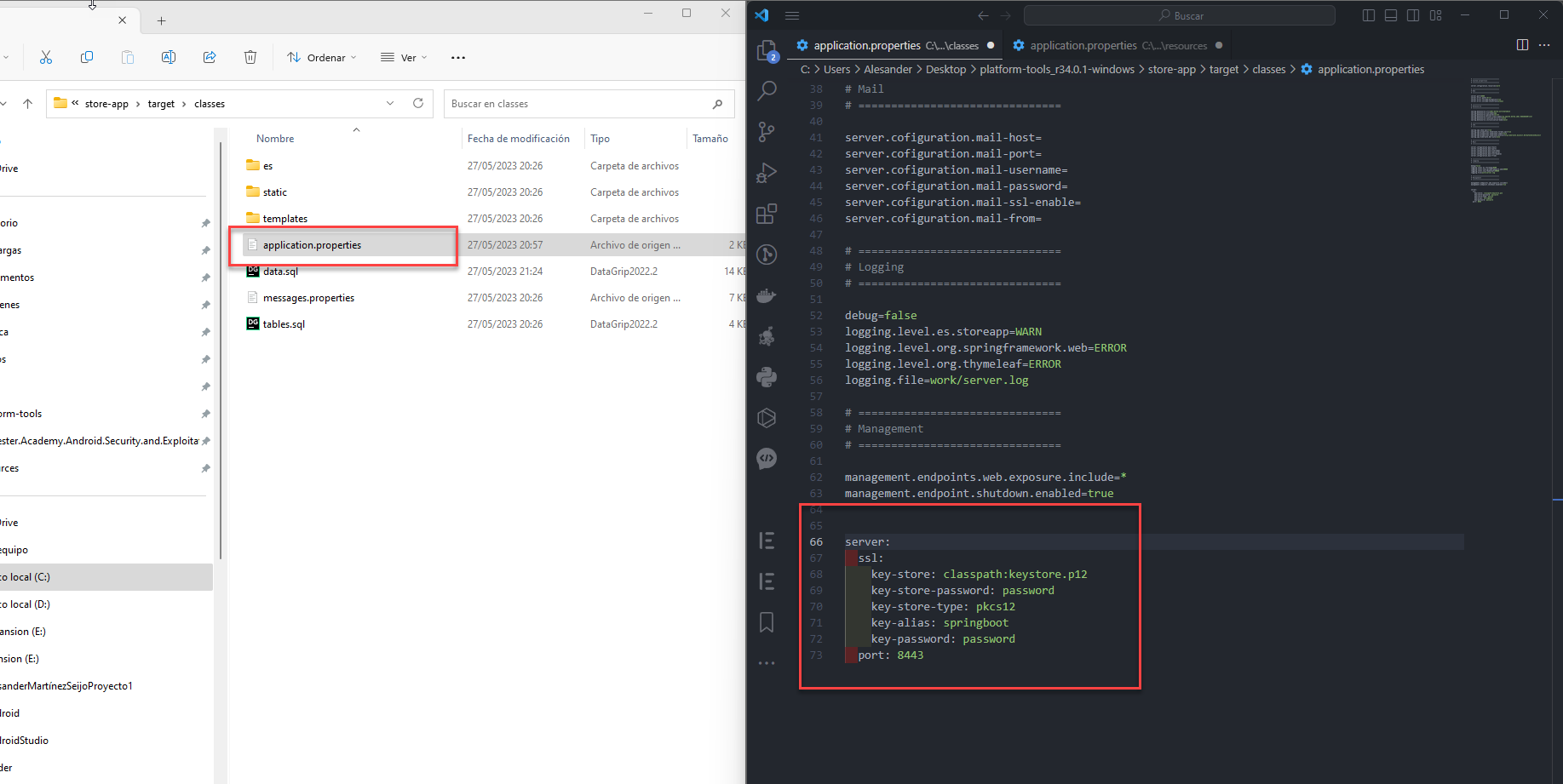
key-store-type: pkcs12

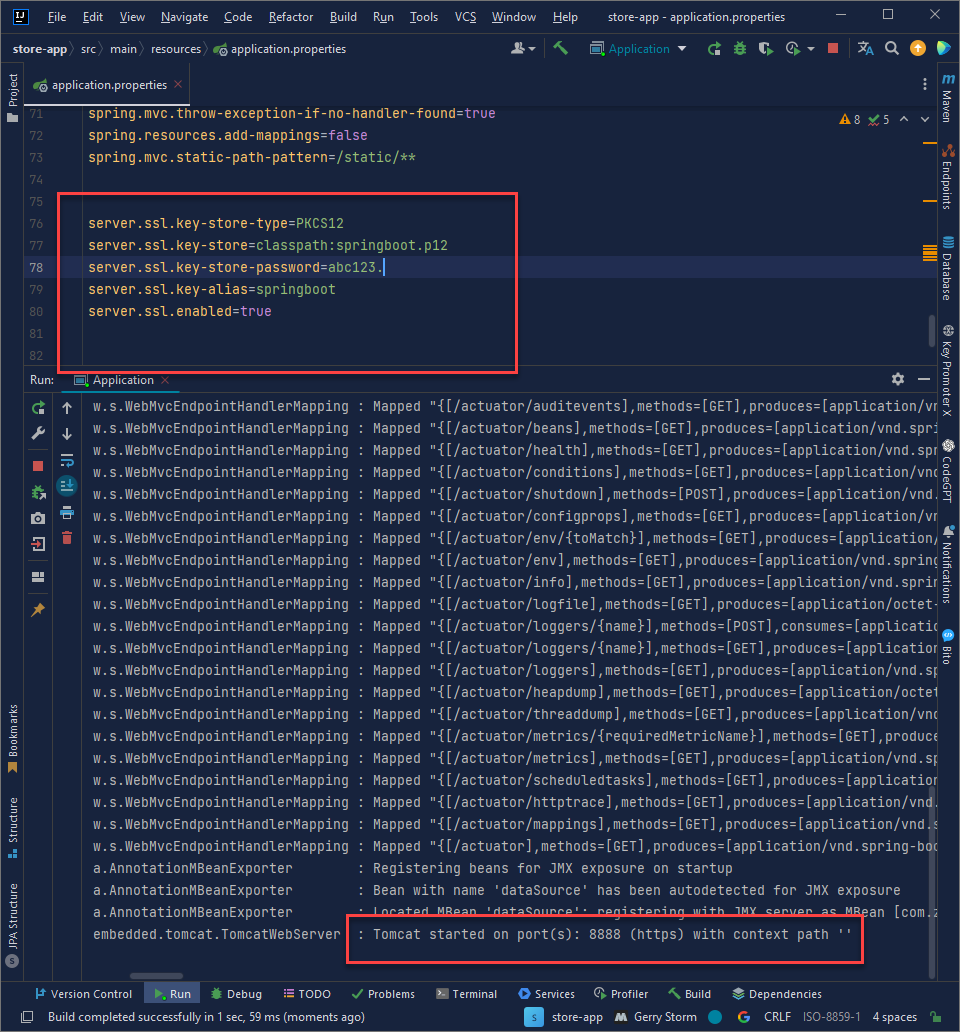
key-alias: springboot

key-password: password

port: 8443

server.ssl.key-store-type=PKCS12  
server.ssl.key-store=classpath:certificado.p12  
server.ssl.key-store-password=123456  
server.ssl.key-alias=certificadoAlias  
server.ssl.enabled=true





<https://www.thomasvitale.com/https-spring-boot-ssl-certificate/>

<https://medium.com/@santiagojmamani/agregar-certificado-ssl-a-aplicaci%C3%B3n-java-con-spring-boot-98e6be92c4ed>

# FILE UPLOAD:

Para esto tendré que modificar la clase y concretamente el método saveProfileImage:

private static final List<String> *ALLOWED\_EXTENSIONS* = Arrays.*asList*("jpg", "jpeg", "png", "gif");  
private static final List<String> *ALLOWED\_MIME\_TYPES* = Arrays.*asList*("image/jpeg", "image/png", "image/gif");  
  
  
private boolean saveProfileImage(Long id, String imageName, byte[] imageContents) {  
 boolean resultado = false;  
 if (imageName != null && !imageName.trim().isEmpty() && imageContents != null) {  
 String extension = getExtension(imageName);  
 String mimeType = null;  
 try {  
 mimeType = Files.*probeContentType*(Paths.*get*(imageName));  
 } catch (IOException e) {  
 *logger*.error(e.getMessage(), e);  
 }  
 if (extension != null && mimeType != null && isAllowedExtension(extension) && isAllowedMimeType(mimeType)) {  
 resultado=!resultado;  
 File userDir = new File(resourcesDir, id.toString());  
 userDir.mkdirs();  
 String safeName = generateSafeFileName(imageName);  
 File profilePicture = new File(userDir, safeName);  
 try (FileOutputStream outputStream = new FileOutputStream(profilePicture)) {  
 IOUtils.*copy*(new ByteArrayInputStream(imageContents), outputStream);  
 } catch (Exception e) {  
 *logger*.error(e.getMessage(), e);  
  
 }  
 }  
 }  
  
 return resultado;  
}

private String getExtension(String fileName) {  
 int dotIndex = fileName.lastIndexOf('.');  
 if (dotIndex > 0 && dotIndex < fileName.length() - 1) {  
 return fileName.substring(dotIndex + 1);  
 }  
 return null;  
}  
  
private boolean isAllowedExtension(String extension) {  
 return *ALLOWED\_EXTENSIONS*.contains(extension.toLowerCase());  
}  
  
private boolean isAllowedMimeType(String mimeType) {  
 return *ALLOWED\_MIME\_TYPES*.contains(mimeType.toLowerCase());  
}  
  
private String generateSafeFileName(String originalName) {  
 String safeName = originalName.replaceAll("[^a-zA-Z0-9\\.\\-\_]", "\_");  
 return safeName;  
}

@Transactional  
public User update(Long id, String name, String email, String address, String image, byte[] imageContents)  
 throws DuplicatedResourceException, InstanceNotFoundException, ServiceException {  
 User user = userRepository.findById(id);  
 User emailUser = userRepository.findByEmail(email);  
 if(emailUser != null && !Objects.*equals*(emailUser.getUserId(), user.getUserId())) {  
 throw exceptionGenerationUtils.toDuplicatedResourceException(Constants.*EMAIL\_FIELD*, email,  
 Constants.*DUPLICATED\_INSTANCE\_MESSAGE*);  
 }  
 user.setName(name);  
 user.setEmail(email);  
 user.setAddress(address);  
 if (image != null && image.trim().length() > 0 && imageContents != null) {  
 try {  
 deleteProfileImage(id, user.getImage());  
 } catch (Exception ex) {  
 *logger*.error(ex.getMessage(), ex);  
 }  
   
 boolean resultado = saveProfileImage(id, image, imageContents);  
   
 if (resultado)  
 user.setImage(image);  
 }  
 return userRepository.update(user);  
}

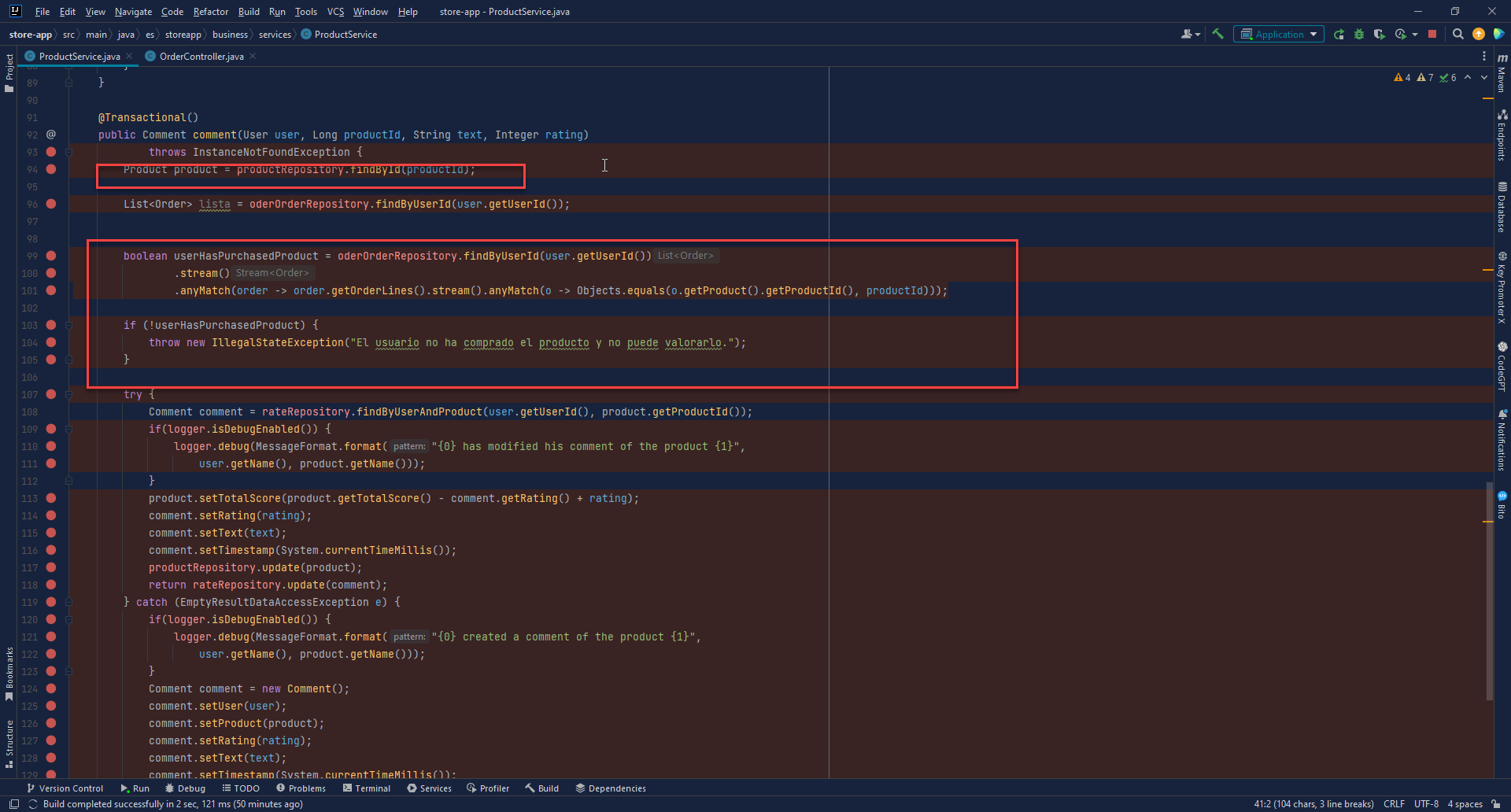
# Valorar productos que no compraste:

Para esto modificaré la clase ProductService y dentro de ella el método comment;

@Transactional()  
public Comment comment(User user, Long productId, String text, Integer rating)  
 throws InstanceNotFoundException {  
 Product product = productRepository.findById(productId);  
  
 List<Order> lista = oderOrderRepository.findByUserId(user.getUserId());  
  
  
 boolean userHasPurchasedProduct = oderOrderRepository.findByUserId(user.getUserId())  
 .stream()  
 .anyMatch(order -> order.getOrderLines().stream().anyMatch(o -> Objects.equals(o.getProduct().getProductId(), productId)));  
  
 if (!userHasPurchasedProduct) {  
 throw new IllegalStateException("El usuario no ha comprado el producto y no puede valorarlo.");  
 }  
  
 try {  
 Comment comment = rateRepository.findByUserAndProduct(user.getUserId(), product.getProductId());  
 if(logger.isDebugEnabled()) {  
 logger.debug(MessageFormat.format("{0} has modified his comment of the product {1}",   
 user.getName(), product.getName()));  
 }  
 product.setTotalScore(product.getTotalScore() - comment.getRating() + rating);  
 comment.setRating(rating);  
 comment.setText(text);  
 comment.setTimestamp(System.currentTimeMillis());  
 productRepository.update(product);  
 return rateRepository.update(comment);  
 } catch (EmptyResultDataAccessException e) {  
 if(logger.isDebugEnabled()) {  
 logger.debug(MessageFormat.format("{0} created a comment of the product {1}",   
 user.getName(), product.getName()));  
 }  
 Comment comment = new Comment();  
 comment.setUser(user);  
 comment.setProduct(product);  
 comment.setRating(rating);  
 comment.setText(text);  
 comment.setTimestamp(System.currentTimeMillis());  
 product.setTotalComments(product.getTotalComments() + 1);  
 product.setTotalScore(product.getTotalScore() + rating);  
 productRepository.update(product);  
 return rateRepository.create(comment);  
 }  
}

Añadiendo esto al inicio de la clase:

@Autowired  
OrderRepository oderOrderRepository;  
@Autowired  
OrderLineRepository orderLine;



# Validación de entidades:

Se validan en /src/main/java/storeapp/web/forms:

@Pattern(regexp = "^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\\d)(?=.\*[@$!%\*?&])[A-Za-z\\d@$!%\*?&]{10,}$")

private String password;

/\*\*

\* ^ - Comienzo de la cadena

\* (?=.\*[a-z]) - Al menos una letra minúscula

\* (?=.\*[A-Z]) - Al menos una letra mayúscula

\* (?=.\*\\d) - Al menos un dígito

\* (?=.\*[@$!%\*?&]) - Al menos un carácter especial entre @$!%\*?&

\* [A-Za-z\\d@$!%\*?&]{10,} - Debe tener 10 o más caracteres, y puede contener letras mayúsculas o minúsculas, dígitos, y carácteres especiales @$!%\*?&

\* $ - Fin de la cadena

\*/

@NotNull

@Pattern(regexp = "^-?\\d{1,19}$")

private Long productId;

/\*\*

\*^ - Comienzo de la cadena

\* -? - Un signo negativo opcional

\* \\d{1,19} - Entre 1 y 19 dígitos

\* $ - Fin de la cadena

\*/

@NotNull

@Pattern(regexp = "^[a-zA-Z0-9áéíóúÁÉÍÓÚ\\.\\s]+$")

private String text;

/\*\*

\* ^ - Comienzo de la cadena

\* [a-zA-Z0-9áéíóúÁÉÍÓÚ\\.\\s]+ - Uno o más caracteres alfabéticos, numéricos, acentos, puntos y espacios en blanco

\* $ - Fin de la cadena

\*/

@NotNull

@Pattern(regexp = "^[1-5]|5$")

private Integer rating;

/\*\*

\* ^: Indica que la coincidencia debe comenzar desde el inicio de la cadena.

\* [1-5]: Indica que debe coincidir con cualquier dígito entre 1 y 5

\* |: Indica una opción alternativa, es decir, puede coincidir con la expresión anterior o la siguiente.

\* 5: Indica que debe coincidir con el número 5.

\* $: Indica que la coincidencia debe terminar al final de la cadena.Co

\* @return

\*/

@NotNull

@Pattern(regexp = "^(.+)@(\\S+)$")

private String email;

/\*\*

\* ^ - Comienzo de la cadena

\* (.+) - Uno o más caracteres de cualquier tipo (capturados en el primer grupo de captura)

\* @ - El carácter @

\* (\\S+) - Uno o más caracteres que no son espacios en blanco (capturados en el segundo grupo de captura)

\* $ - Fin de la cadena

\*/

@Pattern(regexp = "^[a-zA-Z0-9.\_-]{3,}$")

private String name;

/\*\*

\* ^ - Comienzo de la cadena

\* [a-zA-Z0-9.\_-] - Cualquier carácter alfanumérico o un guion bajo, un punto o un guión.

\* {3,} - Longitud mínima de 3 caracteres

\* $ - Fin de la cadena

\*/

@NotNull

@Pattern(regexp = "^\\d+(\\.\\d{1,2})?$")

private int price;

/\*\*

\* ^ - Comienzo de la cadena

\* \\d+ - Uno o más dígitos

\* (\\.\\d{1,2})? - Un punto seguido de uno o dos dígitos opcionales

\* ?$ - Fin de la cadena

\*/

@NotNull

@Pattern(regexp = "^\\d+\\s+([a-zA-Z]+\\s)\*[a-zA-Z]+$")

private String address;

/\*\*

\* ^ - Comienzo de la cadena

\* \\d+ - Uno o más dígitos

\* \\s+ - Uno o más espacios en blanco

\* ([a-zA-Z]+\\s)\* - Cero o más palabras separadas por un espacio en blanco (una palabra se define como una secuencia de una o más letras mayúsculas o minúsculas)

\* [a-zA-Z]+ - Una palabra (definida como una secuencia de una o más letras mayúsculas o minúsculas)

\* $ - Fin de la cadena

\*/

private Boolean defaultCreditCard;

@Pattern(regexp = "^(4[0-9]{12}(?:[0-9]{3})?|5[1-5][0-9]{14})$")

private String creditCard;

/\*\*

\* ^ - Comienzo de la cadena

\* 4[0-9]{12} - Empieza con el número 4 seguido de 12 dígitos

\* (?:[0-9]{3})? - Opcionalmente, puede tener tres dígitos adicionales

\* | - Alternativamente, puede ser

\* 5[1-5][0-9]{14} - El número 5 seguido de un número entre 1 y 5, y 14 dígitos adicionales

\* $ - Fin de la cadena

\*/

@Pattern(regexp = "^[0-9]{3,4}$")

private Integer cvv;

/\*\*

\* ^ - Comienzo de la cadena

\* [0-9]{3,4} - Tres o cuatro dígitos consecutivos

\* $ - Fin de la cadena

\*/

@Pattern(regexp = "^(0?[1-9]|1[0-2])$")

private Integer expirationMonth;

/\*\*

\* ^ - Comienzo de la cadena

\* 0?[1-9] - Un dígito opcional 0 seguido de un dígito entre 1 y 9

\* | - Alternativamente, puede ser

\* 1[0-2] - El número 1 seguido de un dígito entre 0 y 2

\* $ - Fin de la cadena

\*/

@Pattern(regexp = "^(\\d{2}|\\d{4})$")

private Integer expirationYear;

/\*\*

\* ^ - Comienzo de la cadena

\* \\d{2} - Dos dígitos consecutivos

\* | - Alternativamente, puede ser

\* \\d{4} - Cuatro dígitos consecutivos

\* $ - Fin de la cadena

\*/

@Pattern(regexp = "^[A-Za-z]\\d{3}$")

private String category;

/\*\*

\*

\* ^ - Comienzo de la cadena

\* [A-Za-z] - Una letra mayúscula o minúscula

\* \\d{3} - Tres dígitos consecutivos

\* $ - Fin de la cadena

\*/

private String token;

@NotNull

@Pattern(regexp = "^(.+)@(\\S+)$")

private String email;

/\*\*

\* ^ - Comienzo de la cadena

\* (.+) - Uno o más caracteres de cualquier tipo (capturados en el primer grupo de captura)

\* @ - El carácter @

\* (\\S+) - Uno o más caracteres que no son espacios en blanco (capturados en el segundo grupo de captura)

\* $ - Fin de la cadena

\*/

@NotNull

@Pattern(regexp = "^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\\d)(?=.\*[@$!%\*?&])[A-Za-z\\d@$!%\*?&]{10,}$")

private String password;

/\*\*

\* ^ - Comienzo de la cadena

\* (?=.\*[a-z]) - Al menos una letra minúscula

\* (?=.\*[A-Z]) - Al menos una letra mayúscula

\* (?=.\*\\d) - Al menos un dígito

\* (?=.\*[@$!%\*?&]) - Al menos un carácter especial entre @$!%\*?&

\* [A-Za-z\\d@$!%\*?&]{10,} - Debe tener 10 o más caracteres, y puede contener letras mayúsculas o minúsculas, dígitos, y carácteres especiales @$!%\*?&

\* $ - Fin de la cadena

\*/

@NotNull

@Size(min=4)

@Pattern(regexp = "^[a-zA-Z0-9.\_-]{3,}$")

private String name;

/\*\*

\* ^ - Comienzo de la cadena

\* [a-zA-Z0-9.\_-] - Cualquier carácter alfanumérico o un guion bajo, un punto o un guión.

\* {3,} - Longitud mínima de 3 caracteres

\* $ - Fin de la cadena

\*/

@NotNull

@Pattern(regexp = "^(.+)@(\\S+)$")

private String email;

/\*\*

\* ^ - Comienzo de la cadena

\* (.+) - Uno o más caracteres de cualquier tipo (capturados en el primer grupo de captura)

\* @ - El carácter @

\* (\\S+) - Uno o más caracteres que no son espacios en blanco (capturados en el segundo grupo de captura)

\* $ - Fin de la cadena

\*/

@Pattern(regexp = "^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\\d)(?=.\*[@$!%\*?&])[A-Za-z\\d@$!%\*?&]{10,}$")

private String password;

/\*\*

\* ^ - Comienzo de la cadena

\* (?=.\*[a-z]) - Al menos una letra minúscula

\* (?=.\*[A-Z]) - Al menos una letra mayúscula

\* (?=.\*\\d) - Al menos un dígito

\* (?=.\*[@$!%\*?&]) - Al menos un carácter especial entre @$!%\*?&

\* [A-Za-z\\d@$!%\*?&]{10,} - Debe tener 10 o más caracteres, y puede contener letras mayúsculas o minúsculas, dígitos, y carácteres especiales @$!%\*?&

\* $ - Fin de la cadena

\*/

@NotNull

@Pattern(regexp = "^\\d+\\s+([a-zA-Z]+\\s)\*[a-zA-Z]+$")

private String address;

/\*\*

\* ^ - Comienzo de la cadena

\* \\d+ - Uno o más dígitos

\* \\s+ - Uno o más espacios en blanco

\* ([a-zA-Z]+\\s)\* - Cero o más palabras separadas por un espacio en blanco (una palabra se define como una secuencia de una o más letras mayúsculas o minúsculas)

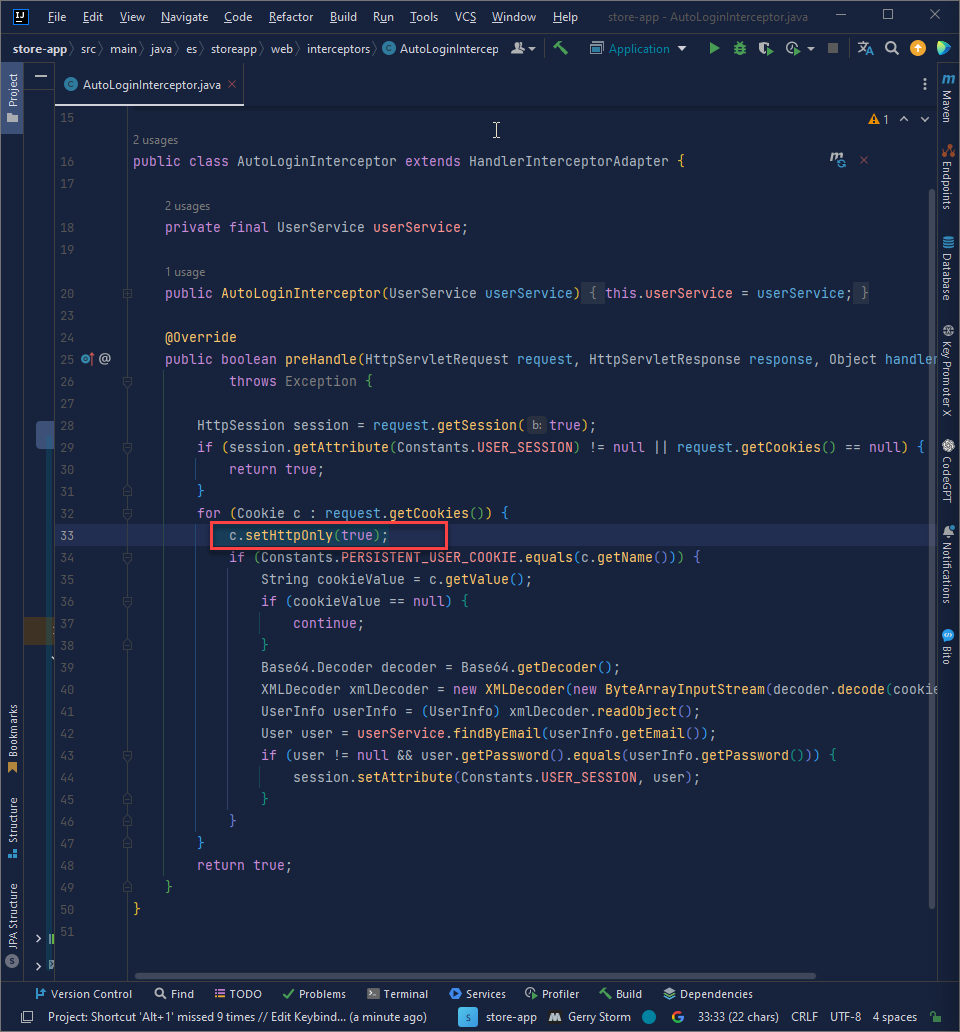
\* [a-zA-Z]+ - Una palabra (definida como una secuencia de una o más letras mayúsculas o minúsculas)

\* $ - Fin de la cadena

\*/

# Cookie segura:

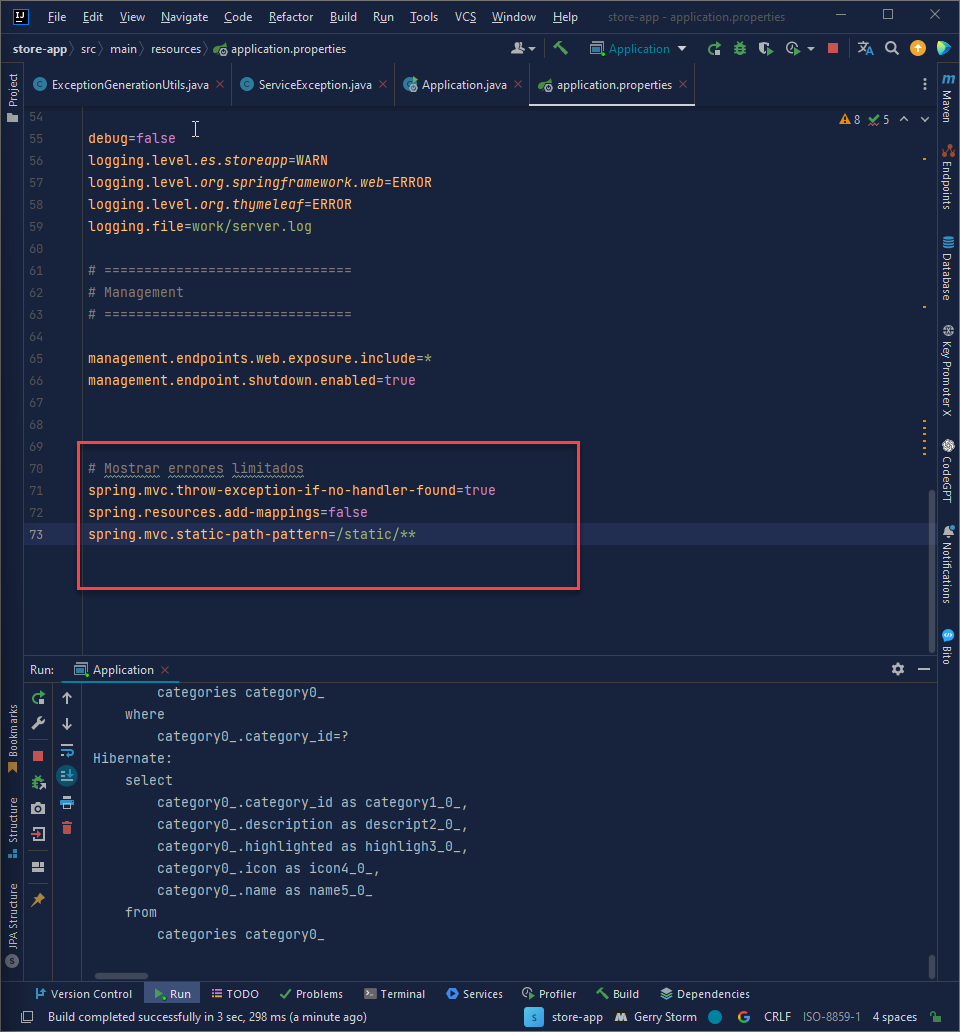
Añado el HttpOnly en la cookie, en la clase AutoLoginInterceptor:



# Limitar número errores:

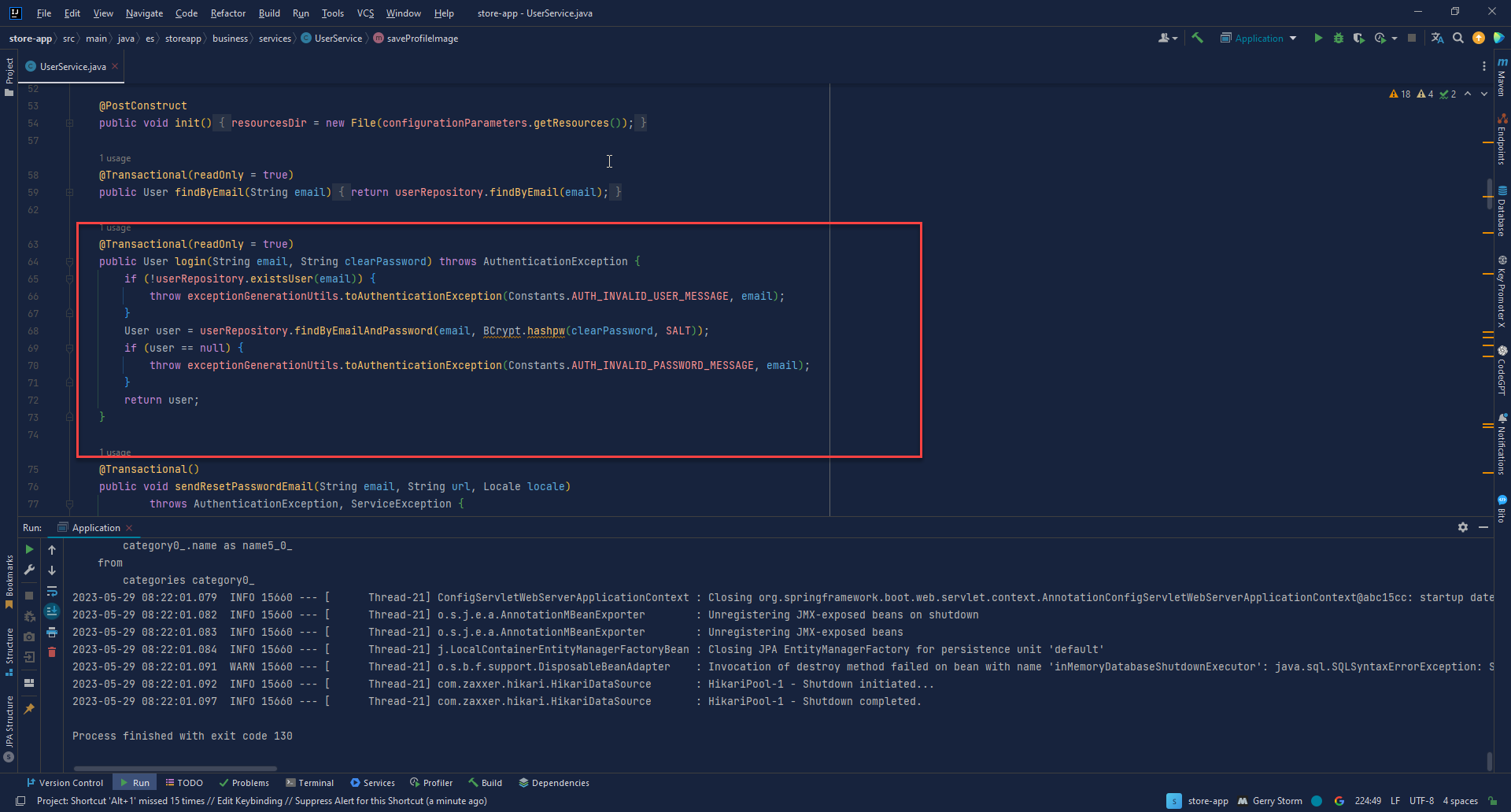
Añado esto en applicaion properties:

# Mostrar errores limitados  
spring.mvc.throw-exception-if-no-handler-found=true  
spring.resources.add-mappings=false  
spring.mvc.static-path-pattern=/static/\*\*

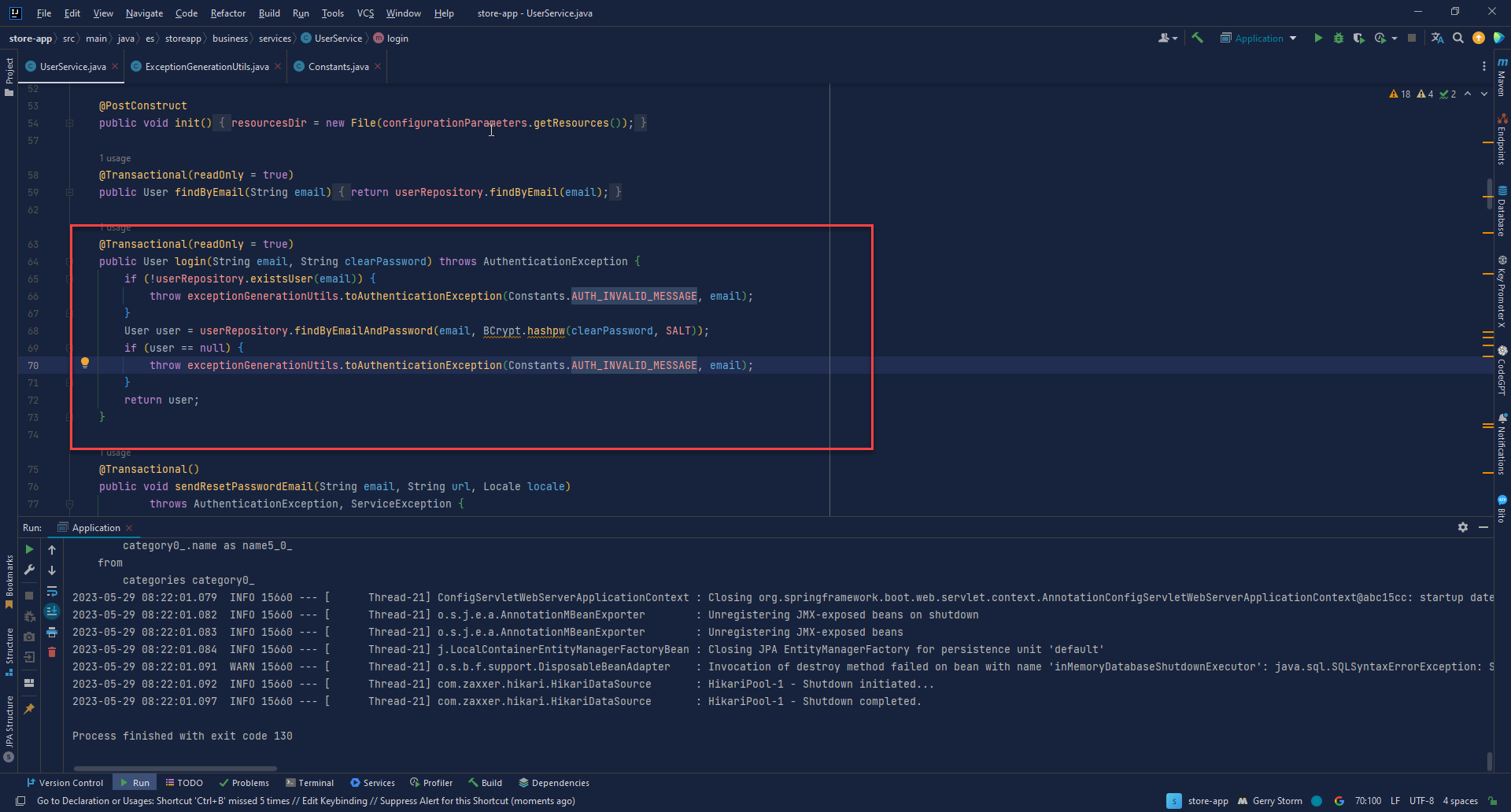


# Demasiado información al loguearse:

Para corregir esto modificaremos los mensajes en esta clase, no pude aparecer un mensaje diferente si falla el usuario o la contraseña:







Aquí crearé los mensajes de error:

