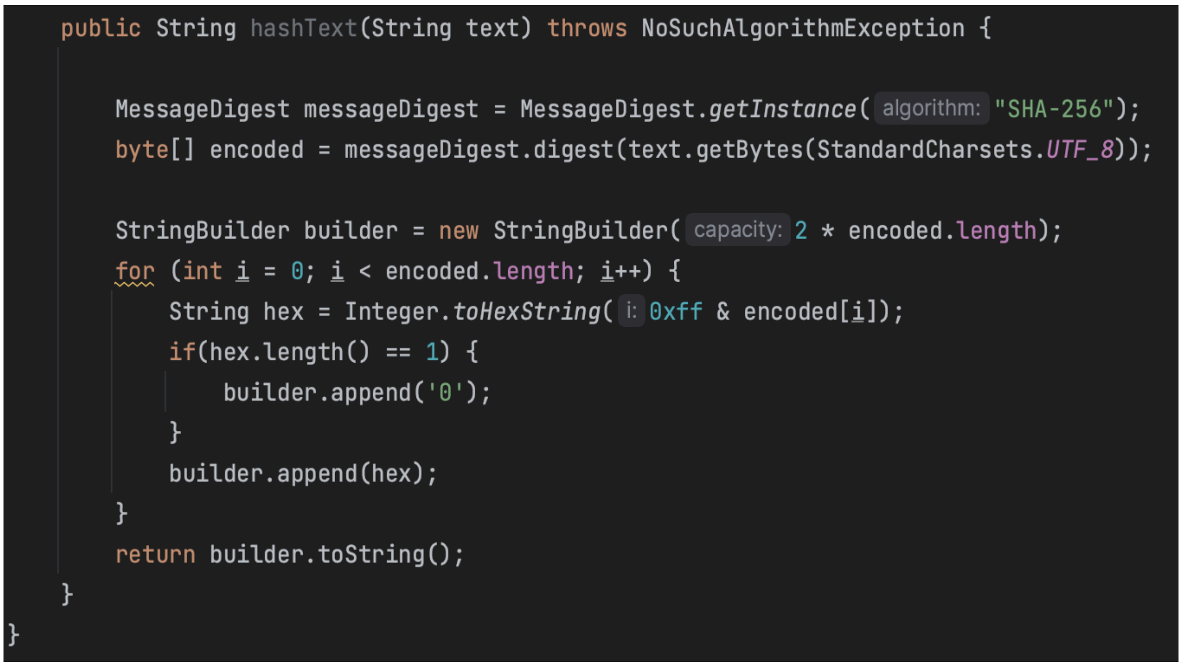
Criterion C: Development

Organization of the Website

The website is split into 2 parts: the customer page which hotel customers order food from, and the admin page which hotel staff manage the orders. These two pages are split with the use of account identification in the login page.

In the customer page, users are able to see the details of the products, add them to their carts and order them as a whole. In the admin page, users are able to see the orders pending to be completed, see an interactive table of products that are in sale, and a table which shows day by day the revenue brought.

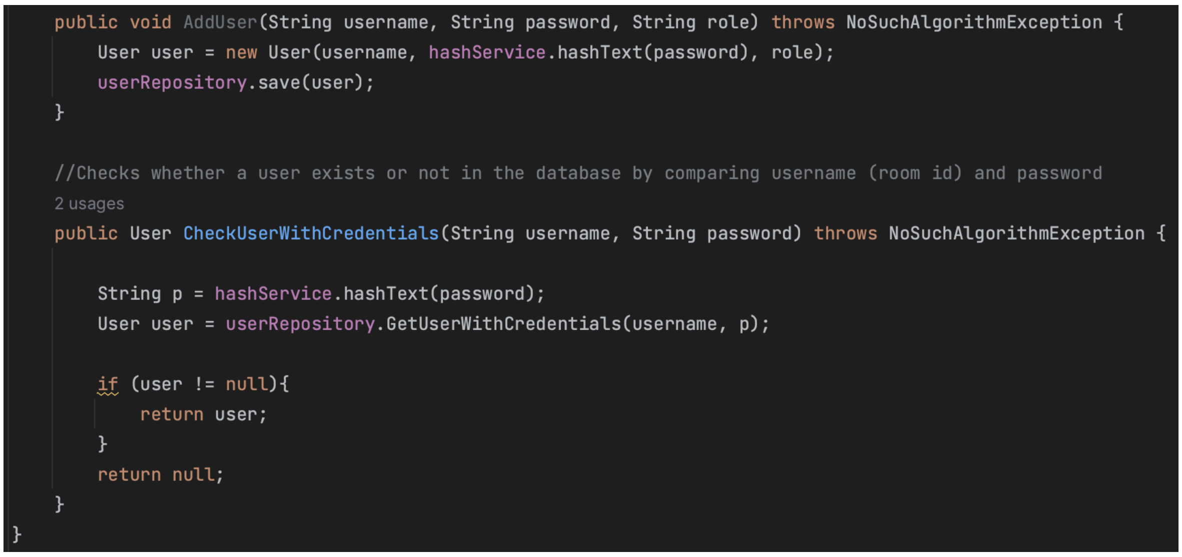
Important Code Parts



```java
    public String hashText(String text) throws NoSuchAlgorithmException {

        MessageDigest messageDigest = MessageDigest.getInstance( algorithm: "SHA-256");
        byte[] encoded = messageDigest.digest(text.getBytes(StandardCharsets.UTF_8));

        StringBuilder builder = new StringBuilder( capacity: 2 * encoded.length);
        for (int i = 0; i < encoded.length; i++) {
            String hex = Integer.toHexString( i: 0xff & encoded[i]);
            if(hex.length() == 1) {
                builder.append('0');
            }
            builder.append(hex);
        }
        return builder.toString();
    }
}
```

Figure 1- Hashing algorithm used to scramble user passwords

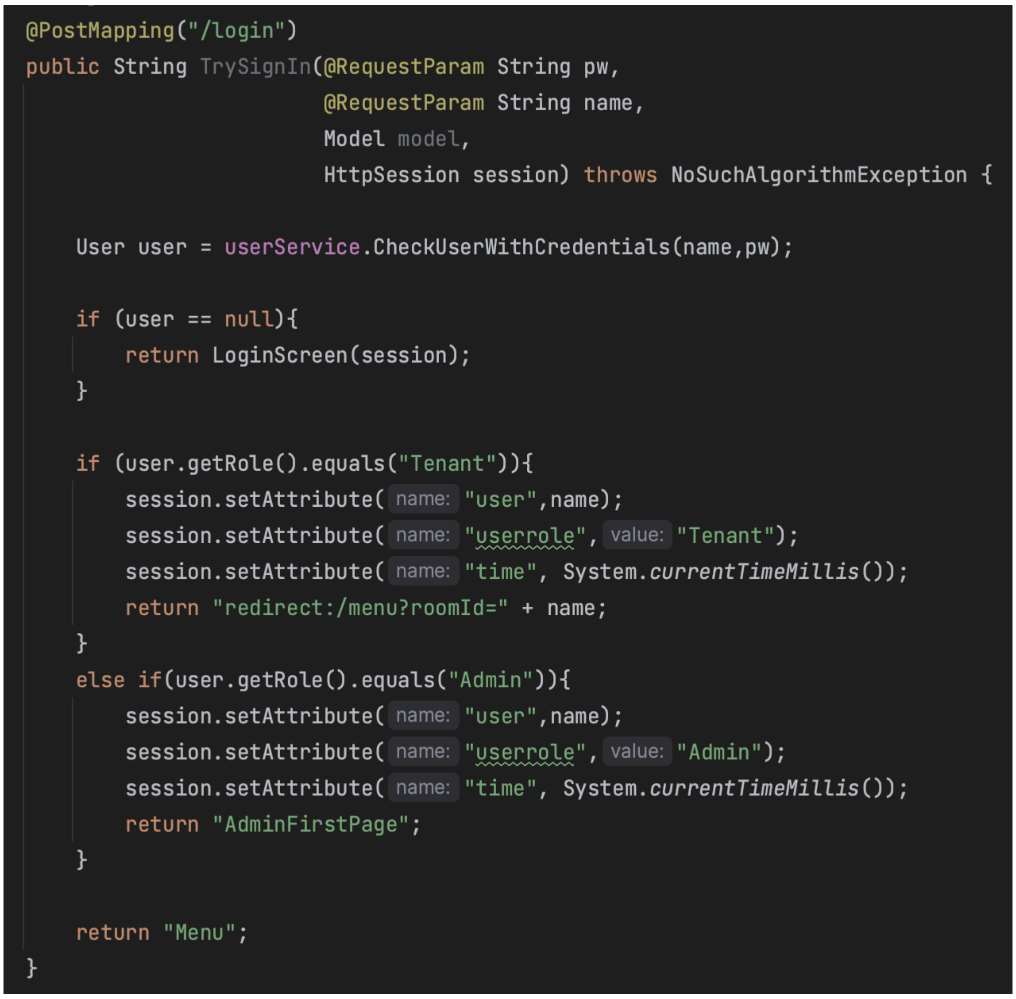
In figure 1, a hashing algorithm (SHA256) is used to make the website more secure in terms of user accounts since hashed passwords can't be undone into their original form. SHA256 was used due to its recognition in providing security, it uses a 256 bit long key to encrypt data.



```
    public void AddUser(String username, String password, String role) throws NoSuchAlgorithmException {
        User user = new User(username, hashService.hashText(password), role);
        userRepository.save(user);
    }

    //Checks whether a user exists or not in the database by comparing username (room id) and password
    2 usages
    public User CheckUserWithCredentials(String username, String password) throws NoSuchAlgorithmException {

        String p = hashService.hashText(password);
        User user = userRepository.GetUserWithCredentials(username, p);

        if (user != null){
            return user;
        }
        return null;
    }
}
```

Figure 2 - Code where the username and password inputted are matched with that of users existing in the database

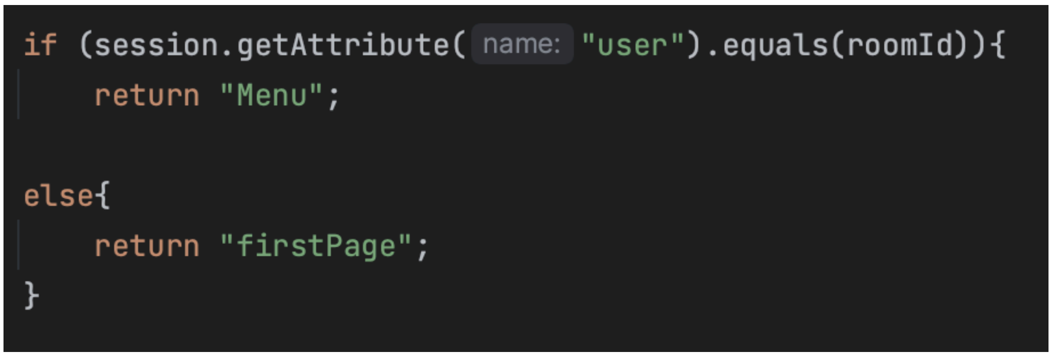
In figure 2, the AddUser function updates the userRepository database table with details of the user including the username, password and role. While doing so, the password gets transformed into an unreadable code using a hash algorithm and then gets stored in the table. The CheckUserWithCredentials function, checks whether a user exists in a database table in order to allow access to the website.



```
@PostMapping("/login")
public String TrySignIn(@RequestParam String pw,
                        @RequestParam String name,
                        Model model,
                        HttpSession session) throws NoSuchAlgorithmException {

    User user = userService.CheckUserWithCredentials(name,pw);

    if (user == null){
        return LoginScreen(session);
    }

    if (user.getRole().equals("Tenant")){
        session.setAttribute( name: "user",name);
        session.setAttribute( name: "userrole", value: "Tenant");
        session.setAttribute( name: "time", System.currentTimeMillis());
        return "redirect:/menu?roomId=" + name;
    }
    else if(user.getRole().equals("Admin")){
        session.setAttribute( name: "user",name);
        session.setAttribute( name: "userrole", value: "Admin");
        session.setAttribute( name: "time", System.currentTimeMillis());
        return "AdminFirstPage";
    }

    return "Menu";
}
```

Figure 3 - Code used for directing the user into the right account's page

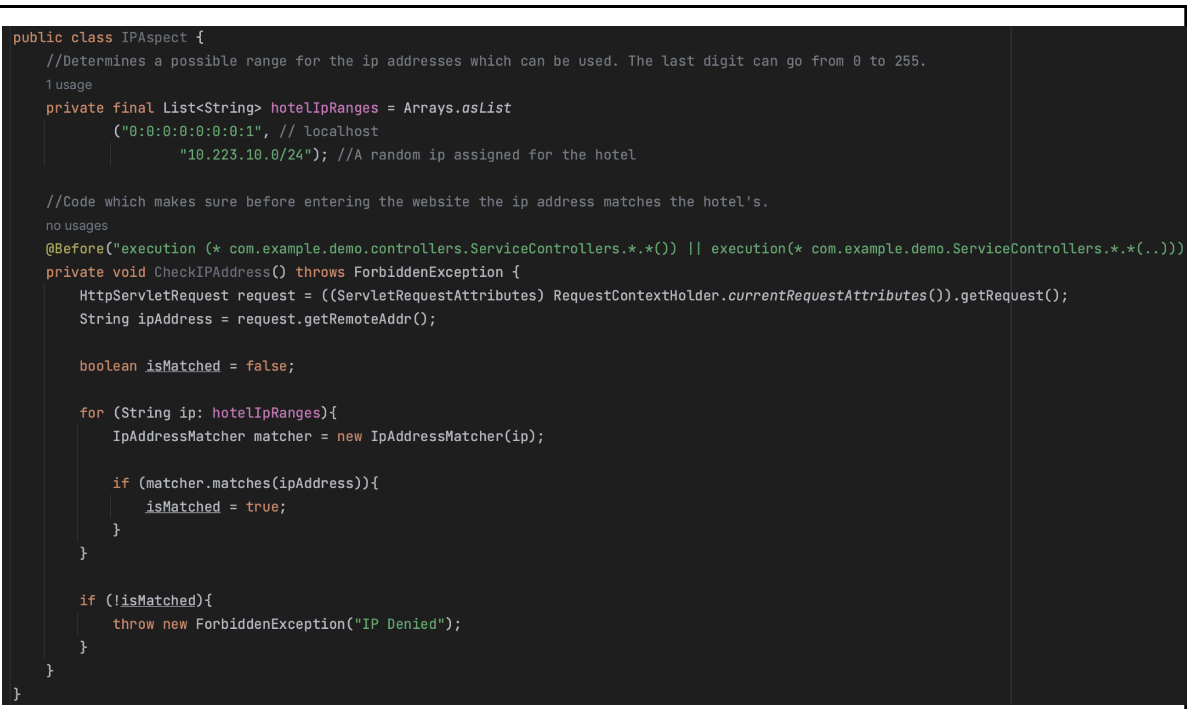
The code in figure 3 decides on the role of the user based on their role attribute. A model is used to store information being transferred to the view. Here the roles being set with setAttribute() contributes to overall user experience by disabling entrance to unintended pages.



```
if (session.getAttribute( name: "user").equals(roomId)){
    return "Menu";

else{
    return "firstPage";
}
```

Figure 4 - Code used to block customers from accessing other customers' accounts

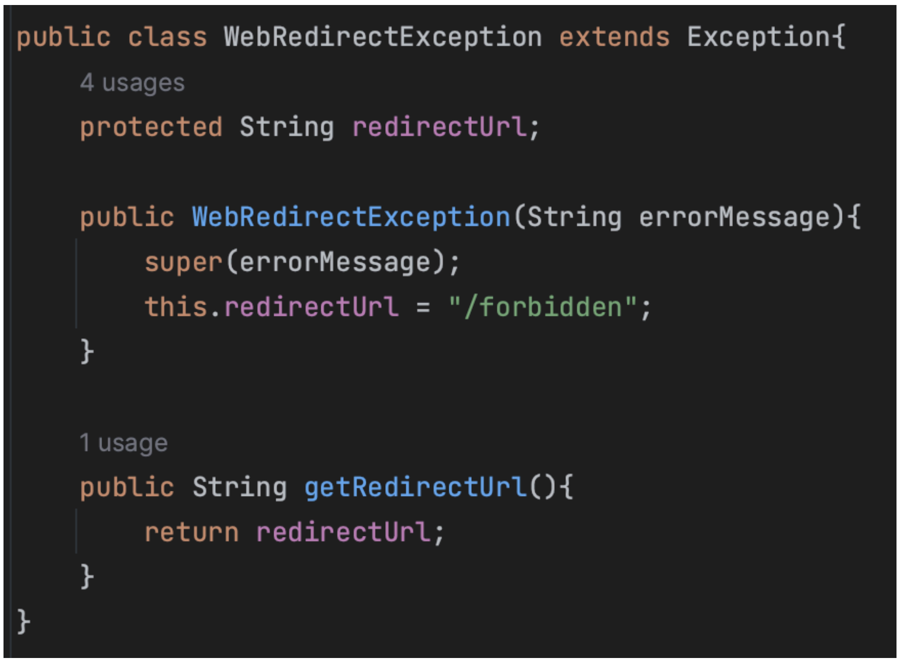
This if statement checks if the roomId in the url (session) is the same as the roomId entered to log in. If it isn't, for example someone wanted to access someone else's account, they are redirected to the login page.



```java
public class IPAspect {
    //Determines a possible range for the ip addresses which can be used. The last digit can go from 0 to 255.
    1 usage
    private final List<String> hotelIpRanges = Arrays.asList
            ("0:0:0:0:0:0:0:1", // localhost
                    "10.223.10.0/24"); //A random ip assigned for the hotel

    //Code which makes sure before entering the website the ip address matches the hotel's.
    no usages
    @Before("execution (* com.example.demo.controllers.ServiceControllers.*.*()) || execution(* com.example.demo.ServiceControllers.*.*(..)))
    private void CheckIPAddress() throws ForbiddenException {
        HttpServletRequest request = ((ServletRequestAttributes) RequestContextHolder.currentRequestAttributes()).getRequest();
        String ipAddress = request.getRemoteAddr();

        boolean isMatched = false;

        for (String ip: hotelIpRanges){
            IpAddressMatcher matcher = new IpAddressMatcher(ip);

            if (matcher.matches(ipAddress)){
                isMatched = true;
            }
        }

        if (!isMatched){
            throw new ForbiddenException("IP Denied");
        }
    }
}
```

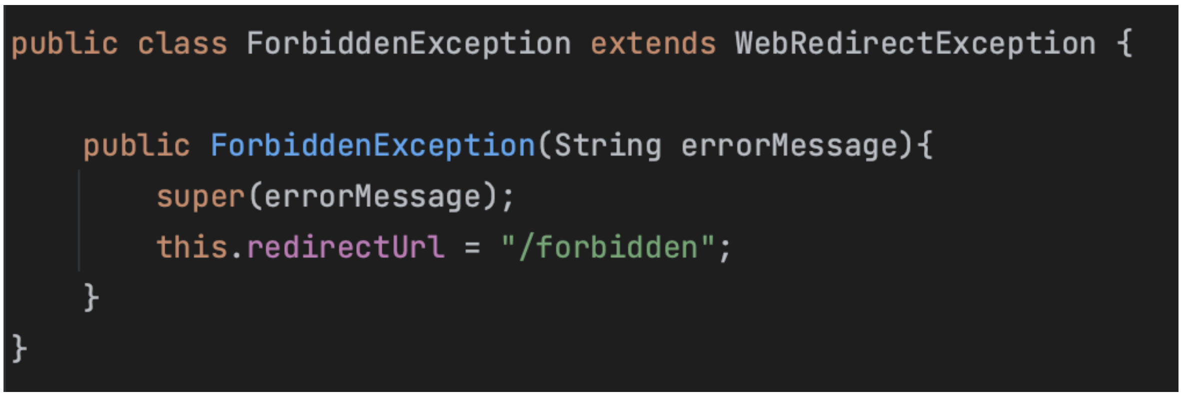
Figure 5 - Code used for enabling access to the website based on the IP address of the user

The code in figure 5 denies requests to enter the website of the user if their ip address isn't compatible with the host's (hotel) ip address. Aspect (AOP) is used to determine that this process takes priority and must be done firstly. If ForbiddenException is thrown, the user is redirected to a page which tells them that their entry request to the website is denied.



```
public class WebRedirectException extends Exception{
    4 usages
    protected String redirectUrl;

    public WebRedirectException(String errorMessage){
        super(errorMessage);
        this.redirectUrl = "/forbidden";
    }

    1 usage
    public String getRedirectUrl(){
        return redirectUrl;
    }
}
```

Figure 6 - Code which redirects the user to the forbidden page

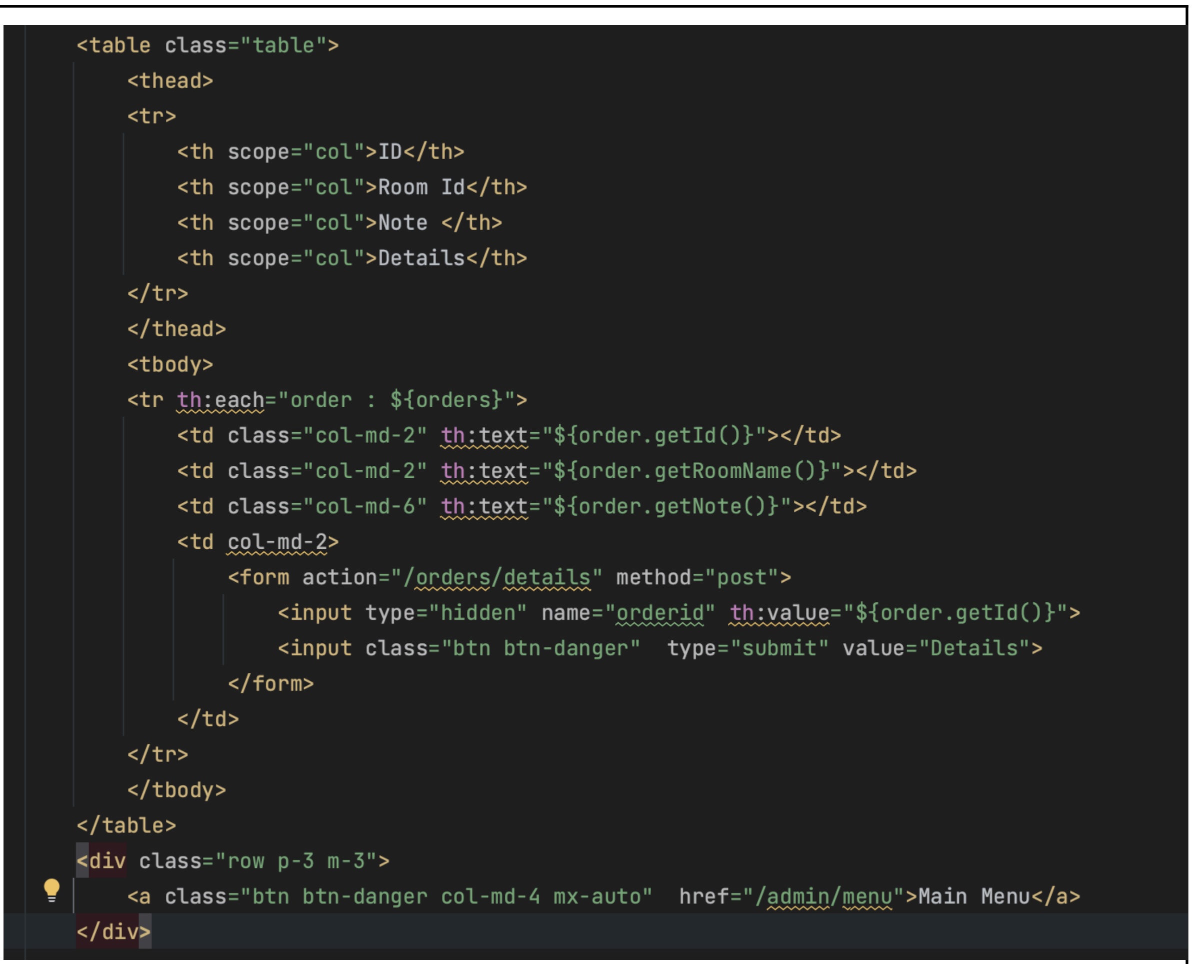


```
public class ForbiddenException extends WebRedirectException {

    public ForbiddenException(String errorMessage){
        super(errorMessage);
        this.redirectUrl = "/forbidden";
    }
}
```

Figure 7 - Subclass of WebRedirectException class which redirects the user to the forbidden page

Both figures 6 and 7 show that object oriented programming (OOP) and its inheritance property is used when redirecting the user to the “access denied” page. This is done because it allows the same result to be reached (redirection to forbidden page) from different methods and places of the website. In figure 6, the errorMessage function of the parent class Exception is called and in figure 7, the errorMessage function is reached through the WebRedirectException class, which is the parent class of Forbidden Exception.

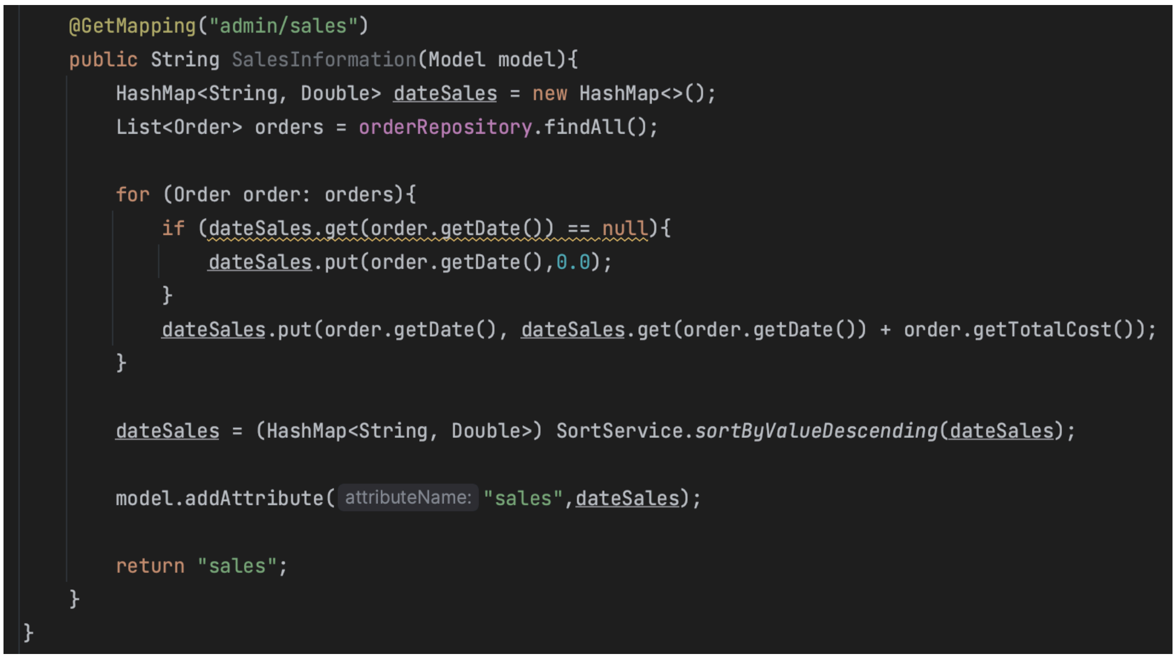
(/forbidden is a page which states that access is denied)



```html
<table class="table">
    <thead>
    <tr>
        <th scope="col">ID</th>
        <th scope="col">Room Id</th>
        <th scope="col">Note </th>
        <th scope="col">Details</th>
    </tr>
    </thead>
    <tbody>
    <tr th:each="order : ${orders}">
        <td class="col-md-2" th:text="${order.getId()}"></td>
        <td class="col-md-2" th:text="${order.getRoomName()}"></td>
        <td class="col-md-6" th:text="${order.getNote()}"></td>
        <td col-md-2>
            <form action="/orders/details" method="post">
                <input type="hidden" name="orderid" th:value="${order.getId()}">
                <input class="btn btn-danger"  type="submit" value="Details">
            </form>
        </td>
    </tr>
    </tbody>
</table>
<div class="row p-3 m-3">
    <a class="btn btn-danger col-md-4 mx-auto"  href="/admin/menu">Main Menu</a>
</div>
```

Figure 8 - Usage of tables in html and iteration with Thymeleaf

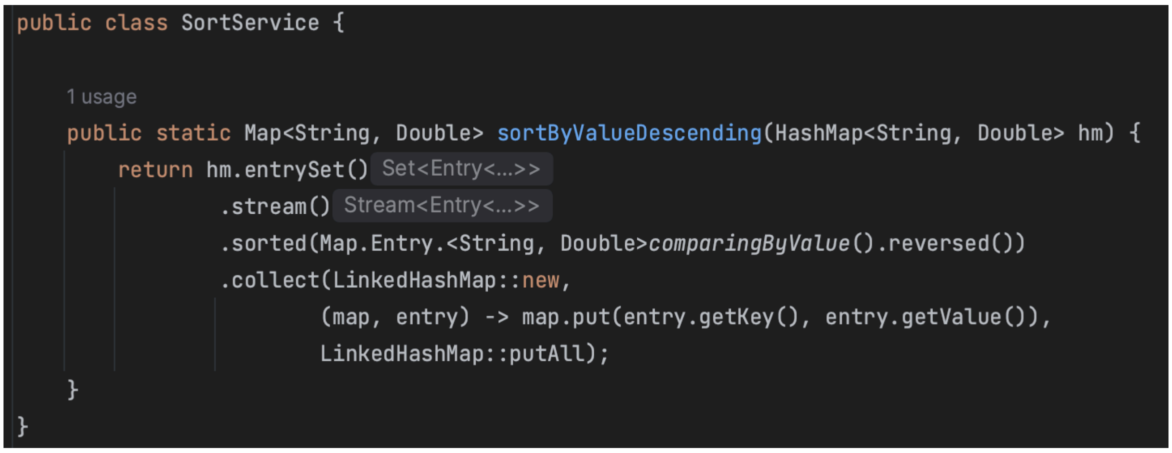
Thymeleaf is a java template engine that focuses on natural html templates for web and standalone applications. In figure 8, a table is initialized with columns ID, Room Id, Note, and Details. The rows of the table are filled with the th:each statement (which is like a for loop in html), where for every order in list orders, the first column is the order's id (order.getId()), the second is the room the order is taken from (getRoomName()), and the third is the note attached by the customer (getNote()).



```
    @GetMapping("admin/sales")
    public String SalesInformation(Model model){
        HashMap<String, Double> dateSales = new HashMap<>();
        List<Order> orders = orderRepository.findAll();

        for (Order order: orders){
            if (dateSales.get(order.getDate()) == null){
                dateSales.put(order.getDate(),0.0);
            }
            dateSales.put(order.getDate(), dateSales.get(order.getDate()) + order.getTotalCost());
        }

        dateSales = (HashMap<String, Double>) SortService.sortByValueDescending(dateSales);

        model.addAttribute( attributeName: "sales",dateSales);

        return "sales";
    }
}
```

Figure 9 - Usage of Hashmaps and sorting for storing and displaying date-money spent pairs

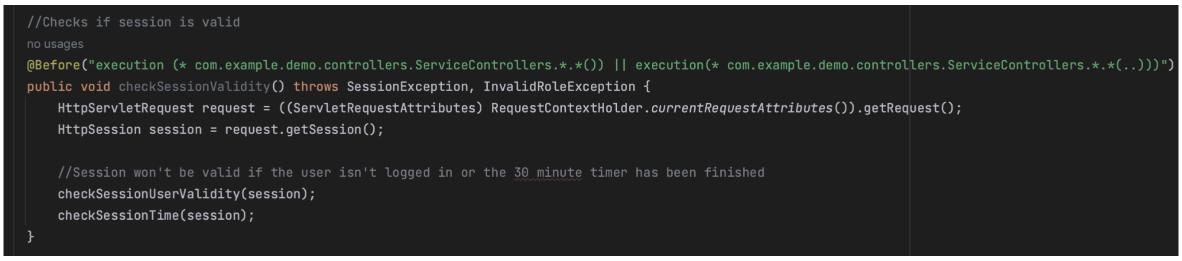
A hashmap is a data structure which stores data in key-value pairs. A value can be reached from its key. Here, a hashmap dateSales is initialized whose key is a string and value is a double. The key is then assigned to be a date value (by order.getDate()) and the value is assigned to be the cost of purchases made in a single day (order.getTotalCost()). This hashmap makes organization more efficient so that it is possible to later on sort all these dates to determine in which days the most money has been spent by customers.



```
public class SortService {

    1 usage
    public static Map<String, Double> sortByValueDescending(HashMap<String, Double> hm) {
        return hm.entrySet() Set<Entry<...>>
                .stream() Stream<Entry<...>>
                .sorted(Map.Entry.<String, Double>comparingByValue().reversed())
                .collect(LinkedHashMap::new,
                        (map, entry) -> map.put(entry.getKey(), entry.getValue()),
                        LinkedHashMap::putAll);
    }
}
```

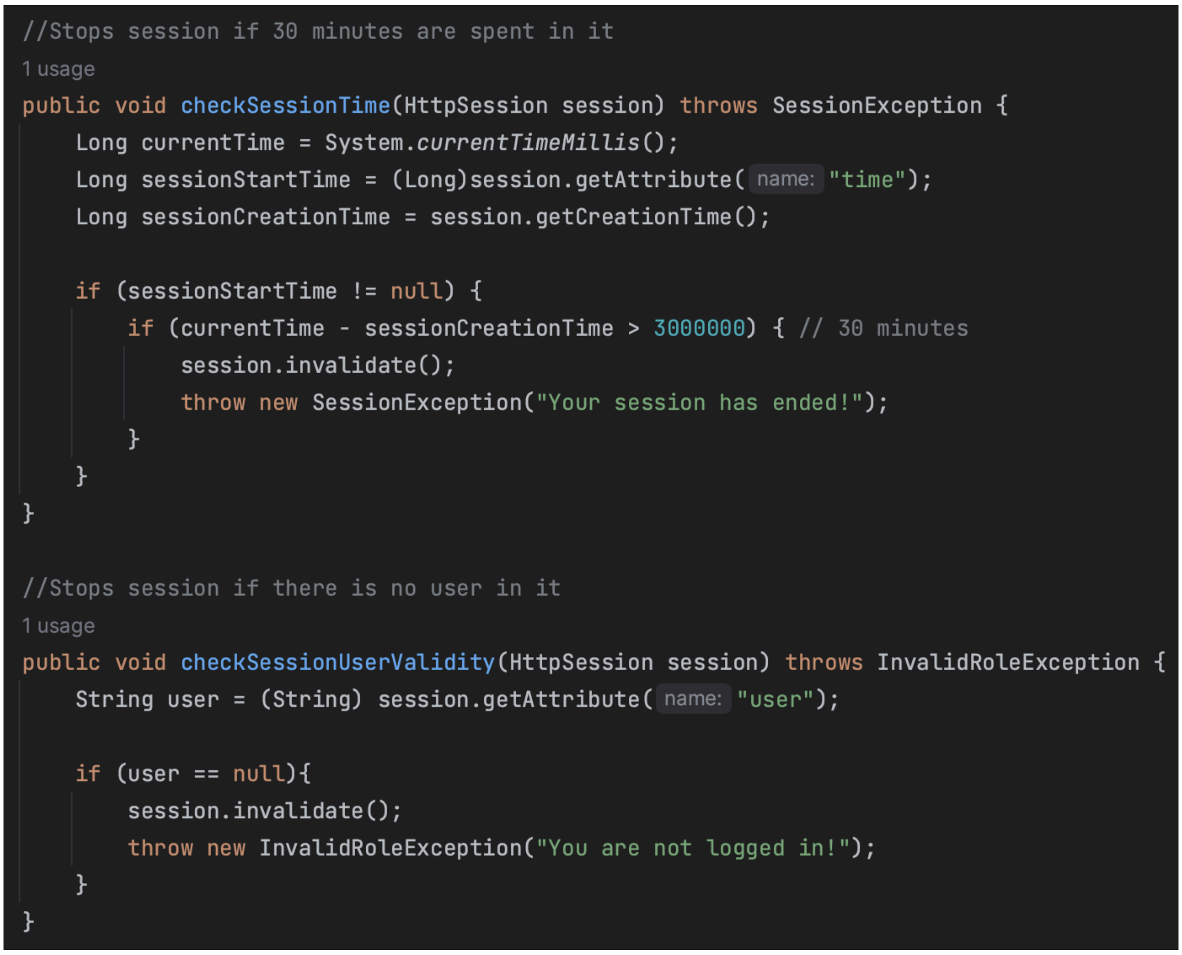
Figure 10 - SortService class and sortByValueDescending method used in figure 9.

This code sorts in a descending order the hashmap entered as a parameter when the sortByValueDescending() method is used.



```
//Checks if session is valid
no usages
@Before("execution (* com.example.demo.controllers.ServiceControllers.*.*()) || execution(* com.example.demo.controllers.ServiceControllers.*.*(..)))")
public void checkSessionValidity() throws SessionException, InvalidRoleException {
    HttpServletRequest request = ((ServletRequestAttributes) RequestContextHolder.currentRequestAttributes()).getRequest();
    HttpSession session = request.getSession();

    //Session won't be valid if the user isn't logged in or the 30 minute timer has been finished
    checkSessionUserValidity(session);
    checkSessionTime(session);
}
```

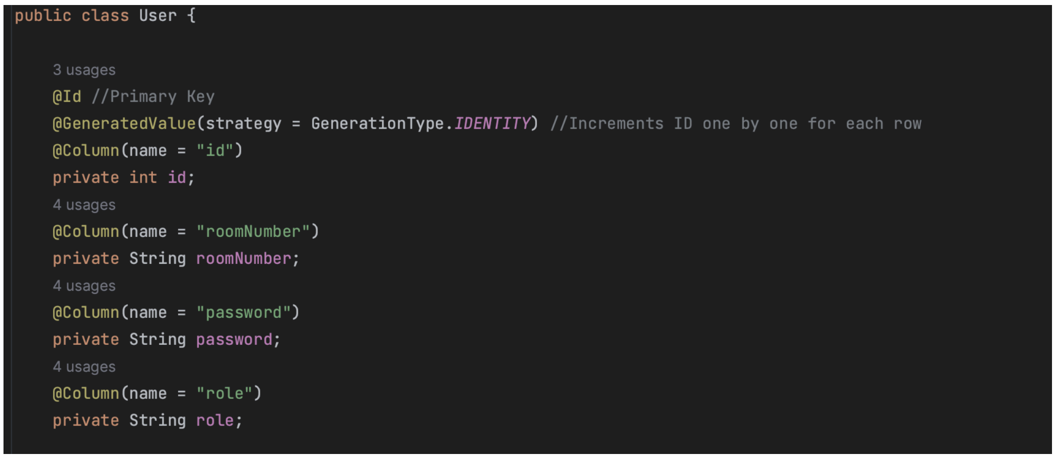
Figure 11- Code which stops the session if either there is no active user or 30 minutes have been spent on the website



```
//Stops session if 30 minutes are spent in it
1 usage
public void checkSessionTime(HttpSession session) throws SessionException {
    Long currentTime = System.currentTimeMillis();
    Long sessionStartTime = (Long)session.getAttribute( name: "time");
    Long sessionCreationTime = session.getCreationTime();

    if (sessionStartTime != null) {
        if (currentTime - sessionCreationTime > 3000000) { // 30 minutes
            session.invalidate();
            throw new SessionException("Your session has ended!");
        }
    }
}

//Stops session if there is no user in it
1 usage
public void checkSessionUserValidity(HttpSession session) throws InvalidRoleException {
    String user = (String) session.getAttribute( name: "user");

    if (user == null){
        session.invalidate();
        throw new InvalidRoleException("You are not logged in!");
    }
}
```

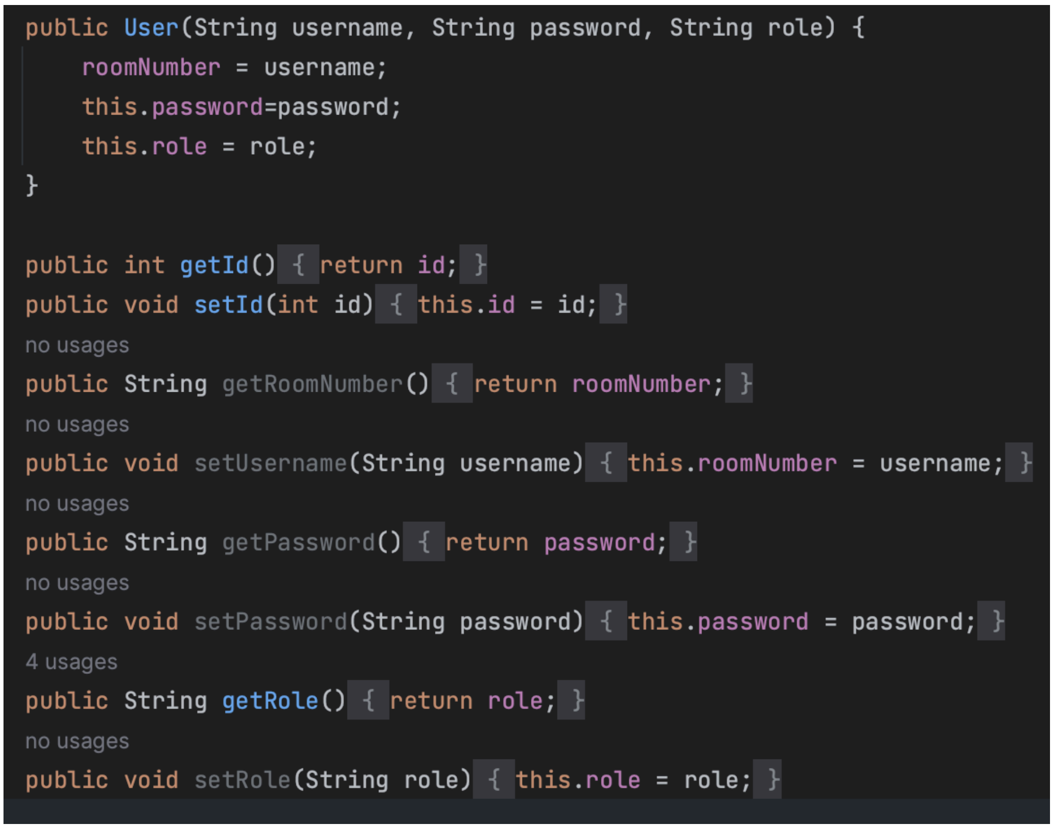
Figure 12- Code blocks which stops and deactivates the session

The checkSessionTime function stops the session if 30 minutes (3000000 milliseconds) have passed in it. The checkSessionUserValidity stops the session if there is no active user in it.



```
public class User {

    3 usages
    @Id //Primary Key
    @GeneratedValue(strategy = GenerationType.IDENTITY) //Increments ID one by one for each row
    @Column(name = "id")
    private int id;
    4 usages
    @Column(name = "roomNumber")
    private String roomNumber;
    4 usages
    @Column(name = "password")
    private String password;
    4 usages
    @Column(name = "role")
    private String role;
```

Figure 13 - Example code which creates tables in the database (in this case the Users database)



```
public User(String username, String password, String role) {
    roomNumber = username;
    this.password=password;
    this.role = role;
}

public int getId() { return id; }
public void setId(int id) { this.id = id; }
no usages
public String getRoomNumber() { return roomNumber; }
no usages
public void setUsername(String username) { this.roomNumber = username; }
no usages
public String getPassword() { return password; }
no usages
public void setPassword(String password) { this.password = password; }
4 usages
public String getRole() { return role; }
no usages
public void setRole(String role) { this.role = role; }
```

Figure 14- Functions used to control user values

Figure 13 is a sample of how the databases are created, in this case the user database includes id as the primary key and roomNumber, password, and role as fields. This database is useful as it makes reaching user information much clearer and easier.

The functions in figure 14 are used to change or detect the users current state in any attribute in order to send them to the right page, and to change or access information about users which will be useful somewhere.

Word count: 982

Used Third-Party Tools

- Bootstrap 4
  CSS library for the front end

- Logback
  The tool which allows tracking of errors

- JPA Hibernate
  ORM tool used for easy database connection between the database and objects used, and for easy use of save, update and select statements

- Spring Framework
  The Spring Framework is an open-source framework for building enterprise Java applications.

- Spring Security
  Used for IP control

- PostgreSQL
  Used database

- Thymeleaf
  Front-end template engine for web development

- Maven
  External libraries management tool

- Docs.oracle