

```
1 package dk.binfo;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.
  SpringApplication;
5
6 @SpringBootApplication
7 public class BinfoApplication {
8
9     public static void main(String[] args) {
10         SpringApplication.run(BinfoApplication.class, args)
11     ;
12     }
13 }
```

```
1 package dk.binfo.models;
2
3 import javax.persistence.Column;
4 import javax.persistence.Entity;
5 import javax.persistence.GeneratedValue;
6 import javax.persistence.GenerationType;
7 import javax.persistence.Id;
8 import javax.persistence.Table;
9 /**
10  * @author Patrick Klæbel
11  * @author Jens Bäckvall
12  * @author Steen Petersen
13  * @author Morten Olsen
14  *
15  */
16 @Entity
17 @Table(name = "role")
18 public class Role {
19     @Id
20     @GeneratedValue(strategy = GenerationType.AUTO)
21     @Column(name="role_id")
22     private int id;
23     @Column(name="role")
24     private String role;
25
26     public int getId() {
27         return id;
28     }
29     public void setId(int id) {
30         this.id = id;
31     }
32     public String getRole() {
33         return role;
34     }
35     public void setRole(String role) {
36         this.role = role;
37     }
38
39
40 }
41
```

```
1 package dk.binfo.models;
2
3 import java.util.Set;
4
5 import javax.persistence.CascadeType;
6 import javax.persistence.Column;
7 import javax.persistence.Entity;
8 import javax.persistence.Id;
9 import javax.persistence.JoinColumn;
10 import javax.persistence.JoinTable;
11 import javax.persistence.ManyToMany;
12 import javax.persistence.Table;
13
14 import org.springframework.data.annotation.Transient;
15 /**
16  * @author Patrick Klæbel
17  * @author Jens Bäckvall
18  * @author Steen Petersen
19  * @author Morten Olsen
20  *
21  */
22 @Entity
23 @Table(name = "user")
24 public class User {
25
26     @Id
27     @Column(name = "email")
28     private String email;
29
30     @Column(name = "password")
31     @Transient
32     private String password;
33
34     @Column(name = "name")
35     private String name;
36
37     @Column(name = "last_name")
38     private String lastName;
39
40     @Column(name = "active")
41     private boolean active;
42
43     @ManyToMany(cascade = CascadeType.ALL)
44     @JoinTable(name = "user_role", joinColumns = @
JoinColumn(name = "user_id"), inverseJoinColumns = @
JoinColumn(name = "role_id"))
45     private Set<Role> roles;
46
47     @Column(name = "phone_number")
48     private String phoneNumber;
```

```
49
50     @Column(name = "my_apartment")
51     private int myApartment;
52
53     public int getMyApartment() {
54         return myApartment;
55     }
56
57     public void setMyApartment(int myApartment) {
58         this.myApartment = myApartment;
59     }
60
61     public String getPhoneNumber() {
62         return phoneNumber;
63     }
64
65     public void setPhoneNumber(String phoneNumber) {
66         this.phoneNumber = phoneNumber;
67     }
68
69     public String getPassword() {
70         return password;
71     }
72
73     public void setPassword(String password) {
74         this.password = password;
75     }
76
77     public String getName() {
78         return name;
79     }
80
81     public void setName(String name) {
82         this.name = name;
83     }
84
85     public String getLastName() {
86         return lastName;
87     }
88
89     public void setLastName(String lastName) {
90         this.lastName = lastName;
91     }
92
93     public String getEmail() {
94         return email;
95     }
96
97     public void setEmail(String email) {
98         this.email = email;
```

```
99     }
100
101     public boolean isActive() {
102         return active;
103     }
104
105     public void setActive(boolean active) {
106         this.active = active;
107     }
108
109     public Set<Role> getRoles() {
110         return roles;
111     }
112
113     public void setRoles(Set<Role> roles) {
114         this.roles = roles;
115     }
116
117 }
118
```

```
1 package dk.binfo.models;
2
3 import javax.persistence.Column;
4 import javax.persistence.Entity;
5 import javax.persistence.GeneratedValue;
6 import javax.persistence.GenerationType;
7 import javax.persistence.Id;
8 import javax.persistence.Table;
9 /**
10  * @author Patrick Klæbel
11  * @author Jens Bäckvall
12  * @author Steen Petersen
13  * @author Morten Olsen
14  *
15  */
16 @Entity
17 @Table(name = "apartment")
18 public class Apartment {
19     @Id
20     @GeneratedValue(strategy = GenerationType.AUTO)
21     @Column(name = "id")
22     private int id;
23     @Column(name = "address")
24     private String address;
25     @Column(name = "number")
26     private String number;
27     @Column(name = "rooms")
28     private Integer rooms;
29     @Column(name = "garden")
30     private boolean garden;
31     @Column(name = "size")
32     private Integer size;
33     @Column(name = "floor")
34     private Integer floor;
35     @Column(name = "floors")
36     private Integer floors;
37
38     public int getId() {
39         return id;
40     }
41
42     public void setId(int id) {
43         this.id = id;
44     }
45
46     public String getAddress() {
47         return address;
48     }
49
50     public void setAddress(String address) {
```

```
51         this.address = address;
52     }
53
54     public String getNumber() {
55         return number;
56     }
57
58     public void setNumber(String number) {
59         this.number = number;
60     }
61
62     public Integer getRooms() {
63         return rooms;
64     }
65
66     public void setRooms(Integer rooms) {
67         this.rooms = rooms;
68     }
69
70     public boolean isGarden() {
71         return garden;
72     }
73
74     public void setGarden(boolean garden) {
75         this.garden = garden;
76     }
77
78     public Integer getSize() {
79         return size;
80     }
81
82     public void setSize(Integer size) {
83         this.size = size;
84     }
85
86     public Integer getFloor() {
87         return floor;
88     }
89
90     public void setFloor(Integer floor) {
91         this.floor = floor;
92     }
93
94     public Integer getFloors() {
95         return floors;
96     }
97
98     public void setFloors(Integer floors) {
99         this.floors = floors;
100    }
```

```
101 }  
102
```



```
1 package dk.binfo.services;
2
3 import dk.binfo.models.User;
4
5 import java.util.ArrayList;
6 import java.util.List;
7 import com.itextpdf.text.*;
8
9 /**
10  * The Interface for creating waitinglists for displaying
11  * on screen
12  * or in a PDF
13  *
14  * @author jensbackvall
15  */
16 public interface ListService {
17     void generateSingleApartmentPDF(int listLength, int
18         apartmentNumber, String filePath);
19     void generateCompleteListPDF(int listLength, int
20         priority, String filePath);
21     List<User> generateList(int length, int priority);
22     List<User> generateSingleApartmentList(int length, int
23         ApartmentId);
24     void loopThroughEmailList(ArrayList<String> emailList,
25         Document theList, String filePath);
26 }
```

```
1 package dk.binfo.services;
2
3 import dk.binfo.models.User;
4
5 import java.util.List;
6 /**
7  * @author Patrick Klæbel
8  * @author Jens Bäckvall
9  * @author Steen Petersen
10 * @author Morten Olsen
11 *
12 */
13 public interface UserService {
14     User findUserByEmail(String email);
15     User updateUserSettings(User user);
16     User deleteUser(String email);
17     User update(User user);
18     List<User> findAll();
19     void register(User user);
20     void adminRegisterUser(User user);
21 }
22
```

```

1 package dk.binfo.services;
2
3 import java.sql.PreparedStatement;
4 import java.sql.ResultSet;
5 import java.util.ArrayList;
6
7 import org.codehaus.groovy.runtime.powerassert.SourceText;
8 import org.springframework.beans.factory.annotation.
  Autowired;
9 import org.springframework.jdbc.core.JdbcTemplate;
10 import org.springframework.stereotype.Service;
11
12 /**
13  * A service we can use to make a waitinglist
14  * the waitinglist generated can be based on
15  * a priority or an apartment
16  *
17  * The waitinglist length can also be based on a
18  * specific length input
19  *
20  * @author Stonie
21  */
22 @Service("waitinglist")
23 public class Waitinglist {
24
25     /**
26      * A pointer we can use to refer to springs
27      JdbcTemplate
28      * so we can use the same database connection and can
29      avoid
30      * creating multiple connections that overlap eachother
31      *
32      * @author Stonie
33      */
34     private final JdbcTemplate jdbcTemplate;
35
36     /**
37      * An autowired method that gets the jdbcTemplate
38      * from spring so we can use the same database
39      * connection and can avoid creating multiple
40      * connections that overlap eachother
41      *
42      * @param jdbcTemplate The length of the
43      waitinglist to be generated
44      * @author Stonie
45      */
46     @Autowired
47     public Waitinglist(JdbcTemplate jdbcTemplate) {
48         this.jdbcTemplate = jdbcTemplate;
49     }
50 }

```

```

47
48     /**
49      * The main method, used to get an ArrayList
50      * containing sorted results based on both
51      * length and ApartmentId.
52      * <p>
53      * It uses almost all methods in the class to
54      * do this in the most optimized way.
55      *
56      * @param      length The length of the waitinglist to
        be generated
57      * @param      ApartmentId The id of the apartment
58      * @return     ArrayList containing emails on the
        waitinglist
59      * @author     Stonie
60      */
61     public ArrayList<String> getWaitinglist(int length,int
        ApartmentId){
62         if(getPreferences(length,ApartmentId)==null){
63             return null;
64         }
65         return checkPriority(length,getPreferences(length,
        ApartmentId),ApartmentId);
66     }
67
68     /**
69      * Returns the emails of neighbours of the ApartmentId
70      * using sql to get the apartments around the apartment
71      * and then uses sql again to get the emails of said
72      * apartments.
73      *
74      * @param      ApartmentId The id of the apartment
75      * @return     ArrayList containing emails of
        neighbours
76      * @author     Stonie
77      */
78     public ArrayList<String> getNeighbourEmails(int
        ApartmentId){
79         ArrayList<Integer> neighboursID = new ArrayList<
        Integer>();
80         try {
81             PreparedStatement sql = jdbcTemplate.
        getDataSource().getConnection().prepareStatement("SELECT
        neighbour FROM `apartment_neighbours` WHERE id_apartment
        =?;");
82             sql.setInt(1, ApartmentId);
83             ResultSet result = sql.executeQuery();
84             while (result.next()) {
85                 int neighbour = result.getInt("neighbour");
86                 neighboursID.add(neighbour);

```

```

87         }
88         sql.close();
89         result.close();
90     } catch (Exception e) {
91         e.printStackTrace();
92     }
93     ArrayList<String> emailsunsorted = new ArrayList<
String>();
94     if (neighboursID.isEmpty()){
95         return emailsunsorted;
96     }
97     for (int neighbourID : neighboursID){
98         try {
99             PreparedStatement sql = jdbcTemplate.
getDataSource().getConnection().prepareStatement("SELECT
email FROM `user` WHERE my_apartment=?");
100             sql.setInt(1, neighbourID);
101             ResultSet result = sql.executeQuery();
102             if (result.next()){
103                 String email = result.getString("email
");
104                 emailsunsorted.add(email);
105             }
106             sql.close();
107             result.close();
108         } catch (Exception e) {
109             e.printStackTrace();
110         }
111     }
112     if (emailsunsorted.isEmpty()){
113         return emailsunsorted;
114     }
115     ArrayList<String> emailssorted = new ArrayList<
String>();
116     try {
117         String SQLString = "SELECT email FROM `
list_and_seniority` WHERE email=? AND list_priority=1";
118         for (int i = 1; i < emailsunsorted.size(); i++)
119         ){
120             SQLString += " OR email=? AND
list_priority=1";
121         }
122         String SQLEND = " ORDER BY seniority ASC;"
;
123         SQLString += SQLEND;
124         PreparedStatement sql = jdbcTemplate.
getDataSource().getConnection().prepareStatement(SQLString
);
125         for (int i = 0; i < emailsunsorted.size(); i++)
126         ){

```

```

125         sql.setString(i+1, emailsunsorted.get(
126             i));
127     }
128     ResultSet result = sql.executeQuery();
129     while (result.next()){
130         String email = result.getString("email
131     ");
132         emailsorted.add(email);
133     }
134     sql.close();
135     result.close();
136     return emailsorted;
137 } catch (Exception e){
138     e.printStackTrace();
139 }
140 return null;
141 }
142 /**
143  * Returns the emails on waitinglist for the
144  * ApartmentId
145  * using sql to get them.
146  * The list will not be sorted.
147  * <p>
148  * It will NOT contain the emails from the neighbour
149  * waitinglist, use the getNeighbourEmails method for
150  * that
151  *
152  * @param length The length of the waitinglist
153  * to be generated
154  * @param ApartmentId The id of the apartment
155  * @return ArrayList containing emails on the
156  * waitinglist
157  * @author Stonie
158  */
159 public ArrayList<String> getPreferences(int length,int
160     ApartmentId){
161     ArrayList<String> pref = new ArrayList<String>();
162     try {
163         PreparedStatement sql = jdbcTemplate.
164         getDataSource().getConnection()
165         .prepareStatement("SELECT email FROM `
166         user_preferences` WHERE id_apartment=?;");
167         sql.setInt(1, ApartmentId);
168         ResultSet result = sql.executeQuery();
169         if(!result.next()){
170             sql.close();
171             result.close();
172             return null;
173         }

```

```

166
167         do {
168             String email = result.getString("email
169         ");
170             pref.add(email);
171             if (pref.size() >= length){
172                 sql.close();
173                 result.close();
174                 return pref;
175             }
176         } while (result.next());
177
178         sql.close();
179         result.close();
180         return pref;
181     } catch (Exception e) {
182         e.printStackTrace();
183     }
184
185     return null;
186 }
187
188 /**
189  * Sorts the emails based on seniority
190  * and returns the sorted list as an ArrayList
191  *
192  * @param length The length of the waitinglist
193  * to be generated
194  * @param emails An arraylist containing the
195  * emails to be sorted
196  * @param ApartmentId The id of the apartment
197  * @return ArrayList containing emails sorted by
198  * seniority
199  * @author Stonie
200  */
201 public ArrayList<String> checkPriority(int length,
202 ArrayList<String> emails, int ApartmentId){
203
204     ArrayList<String> emailssorted =
205     getNeighbourEmails(ApartmentId);
206     try {
207         String SQLString = "SELECT email FROM
208         `list_and_seniority` WHERE email=? AND list_priority=2";
209         for (int i = 1;i<emails.size();i++){
210             SQLString += " OR email=? AND
211             list_priority=2";
212         }
213         String SQLEND = " ORDER BY seniority
214         ASC;";

```

```

207         SQLString += SQLEND;
208         PreparedStatement sql = jdbcTemplate.
        getDataSource().getConnection().prepareStatement(SQLString
        );
209         for (int i = 0; i < emails.size(); i++) {
210             sql.setString((i+1), emails.get(i)
        );
211         }
212         ResultSet result = sql.executeQuery();
213         while (result.next()) {
214             String email = result.getString("
        email");
215             emailsorted.add(email);
216         }
217         sql.close();
218         result.close();
219     } catch (Exception e) {
220         e.printStackTrace();
221     }
222     try {
223         String SQLString = "SELECT email FROM `
        list_and_seniority` WHERE email=? AND list_priority=3";
224         for (int i = 1; i < emails.size(); i++) {
225             SQLString += " OR email=? AND
        list_priority=3";
226         }
227         String SQLEND = " ORDER BY seniority ASC;";
228         SQLString += SQLEND;
229         PreparedStatement sql = jdbcTemplate.
        getDataSource().getConnection().prepareStatement(SQLString
        );
230         for (int i = 0; i < emails.size(); i++) {
231             sql.setString((i+1), emails.get(i));
232         }
233         ResultSet result = sql.executeQuery();
234         while (result.next()) {
235             String email = result.getString("email");
236             emailsorted.add(email);
237         }
238         sql.close();
239         result.close();
240     } catch (Exception e) {
241         e.printStackTrace();
242     }
243     try {
244         String SQLString = "SELECT email FROM `
        list_and_seniority` WHERE email=? AND list_priority=4";
245         for (int i = 1; i < emails.size(); i++) {
246             SQLString += " OR email=? AND
        list_priority=4";

```



```

247     }
248     String SQLEND = " ORDER BY seniority ASC;";
249     SQLString += SQLEND;
250     PreparedStatement sql = jdbcTemplate.
getDataSource().getConnection().prepareStatement(SQLString
);
251     for (int i = 0; i < emails.size(); i++) {
252         sql.setString((i+1), emails.get(i));
253     }
254     ResultSet result = sql.executeQuery();
255     while (result.next()) {
256         String email = result.getString("email");
257         emailsorted.add(email);
258     }
259     sql.close();
260     result.close();
261 } catch (Exception e) {
262     e.printStackTrace();
263 }
264 int size = emailsorted.size();
265 if (length < size) {
266     emailsorted.subList(length, size).clear();
267 }
268 return emailsorted;
269 }
270
271 /**
272  * Returns an ArrayList containing all the emails
273  * on a specific waitinglist based on priority and
274  * sorted by seniority.
275  *
276  * @param length The length of the waitinglist
to be generated
277  * @param priority the priority of the list to
be generated
278  * @return ArrayList containing emails on the
priority list sorted by seniority
279  * @author Stonie
280  */
281 public ArrayList<String> getSingleWaitinglist(int
length, int priority) {
282     if (priority > 4 || priority < 1) {
283         return null;
284     }
285     ArrayList<String> emailsorted = new ArrayList<
String>();
286     try {
287         PreparedStatement sql = jdbcTemplate.
getDataSource().getConnection().prepareStatement("SELECT
email FROM `list_and_seniority` WHERE list_priority="+

```

```
287 priority+" ORDER BY seniority ASC;");
288         ResultSet result = sql.executeQuery();
289         while (result.next()){
290             String email = result.getString("email");
291             emailsorted.add(email);
292         }
293         sql.close();
294         result.close();
295         return emailsorted;
296     } catch (Exception e){
297         e.printStackTrace();
298     }
299     return emailsorted;
300 }
301 }
302
```

```

1 package dk.binfo.services;
2
3 import org.springframework.stereotype.Service;
4
5 import java.util.Properties;
6 import javax.mail.Message;
7 import javax.mail.Session;
8 import javax.mail.Transport;
9 import javax.mail.internet.InternetAddress;
10 import javax.mail.internet.MimeMessage;
11
12 /**
13  * A class with methods to handle emails
14  * so far it can only handle gmail.
15  *
16  * @author Stonie
17  */
18 @Service("emailService")
19 public class EmailService {
20
21     /**
22      * A simple method to send emails using Gmail
23      * using param user and password to login
24      *
25      * @param To The email we should send to
26      * @param Subject The subject of the email
27      * @param Body The body of the email, you can use html
28      * tags like <br>
29      * @author Stonie
30      */
31     public void generateAndSendEmail(String To, String
32 Subject, String Body){
33         try {
34             Properties mailServerProperties;
35             Session getMailSession;
36             MimeMessage generateMailMessage;
37             // setup Mail Server Properties
38             mailServerProperties = System.getProperties();
39             mailServerProperties.put("mail.smtp.port", "587");
40             mailServerProperties.put("mail.smtp.auth", "true");
41             mailServerProperties.put("mail.smtp.starttls.enable
42 ", "true");
43             mailServerProperties.put("mail.smtp.ssl.trust", "
44 smtp.gmail.com");
45
46             // get Mail Session
47             getMailSession = Session.getDefaultInstance(
48 mailServerProperties, null);
49             generateMailMessage = new MimeMessage(
50 getMailSession);

```

```
45         generateMailMessage.addRecipient(Message.  
RecipientType.TO, new InternetAddress(To));  
46         generateMailMessage.setSubject(Subject);  
47         String emailBody = Body;  
48         generateMailMessage.setContent(emailBody, "text/  
html; charset=UTF-8");  
49  
50         // Get Session and Send mail  
51         Transport transport = getMailSession.getTransport("  
smtp");  
52  
53         // Enter your correct gmail UserID and Password  
54         transport.connect("smtp.gmail.com", "testbinfo1234"  
, "testbinfo1234testbinfo1234");  
55         transport.sendMessage(generateMailMessage,  
generateMailMessage.getAllRecipients());  
56         transport.close();  
57         }catch(Exception e){  
58             e.printStackTrace();  
59         }  
60     }  
61 }  
62
```

```
1 package dk.binfo.services;
2
3 import dk.binfo.models.Apartment;
4 import dk.binfo.repositories.ApartmentRepository;
5 import org.springframework.beans.factory.annotation.
    Autowired;
6 import org.springframework.stereotype.Service;
7 import org.springframework.transaction.annotation.
    Transactional;
8
9 import java.util.List;
10 /**
11  * @author Patrick Klæbel
12  * @author Jens Bäckvall
13  * @author Steen Petersen
14  * @author Morten Olsen
15  *
16  */
17 @Service("apartmentService")
18 public class ApartmentImpl implements ApartmentService {
19
20     @Autowired
21     private ApartmentRepository apartmentRepository;
22
23     @Override
24     public void saveApartment(Apartment apartment) {
25         apartmentRepository.save(apartment);
26     }
27
28     @Override
29     @Transactional
30     public Apartment findById(int id) {
31         return apartmentRepository.findOne(id);
32     }
33
34     @Override
35     @Transactional
36     public Apartment delete(int id) {
37         Apartment deletedApartment = apartmentRepository.
    findOne(id);
38
39         apartmentRepository.delete(deletedApartment);
40         return deletedApartment;
41     }
42
43     @Override
44     @Transactional
45     public List<Apartment> findAll() {
46         return apartmentRepository.findAll();
47     }
48 }
```

```
48
49     @Override
50     @Transactional()
51     public Apartment update(Apartment apartment){
52         Apartment updateApartment = apartmentRepository.
findOne(apartment.getId());
53
54         updateApartment.setAddress(apartment.getAddress());
55         updateApartment.setNumber(apartment.getNumber());
56         updateApartment.setRooms(apartment.getRooms());
57         updateApartment.setGarden(apartment.isGarden());
58         updateApartment.setSize(apartment.getSize());
59         updateApartment.setFloor(apartment.getFloor());
60         updateApartment.setFloors(apartment.getFloors());
61         return updateApartment;
62     }
63 }
64
```

```

1 package dk.binfo.services;
2
3 import java.io.FileNotFoundException;
4 import java.io.FileOutputStream;
5 import java.io.File;
6 import java.util.ArrayList;
7 import java.util.List;
8
9 import com.itextpdf.text.*;
10 import com.itextpdf.text.pdf.PdfWriter;
11 import dk.binfo.models.User;
12 import org.springframework.beans.factory.annotation.
    Autowired;
13 import org.springframework.stereotype.Service;
14
15 import static java.lang.Integer.MAX_VALUE;
16
17
18 /**
19  * A service used for generating waitinglists for
20  * displaying on screen
21  * in a table or for generating a PDF of a list for a
22  * specific
23  * apartment.
24  *
25  * The Waitinglist service is used, and just as we can do
26  * it in that
27  * service, we use length, i.e. number of users in list and
28  * priority,
29  * i.e. which list we want to display, as parameters.
30  *
31  * The priorities are:
32  * 1. Connect list, for connecting apartments into larger
33  * apartments
34  * 2. Internal list, for users who already live in the
35  * building
36  * 3. Family list, for users who are related to Internal
37  * users
38  * 4. External list, for users who have no relation to the
39  * building.
40  *
41  * @author jensbackvall
42  */
43
44 @Service("listService")
45 public class ListServiceImpl implements ListService {
46
47     @Autowired
48     private UserService userService;
49
50 }

```

```

42     @Autowired
43     private Waitinglist waitinglist;
44
45     private Font theFont = new Font(Font.FontFamily.
TIMES_ROMAN, 16, Font.BOLD);
46     private Font theSmallFont = new Font(Font.FontFamily.
TIMES_ROMAN, 12, Font.BOLD);
47
48     /**
49      * The generatePDF method generates a PDF using the
itext API
50      * for creating a pdf. It also uses getWaitinglist from
the
51      * waitinglist service, and therefore takes in the same
parameters:
52      *
53      * @param listLength
54      * @param apartmentNumber
55      *
56      */
57
58     @Override
59     public void generateSingleApartmentPDF(int listLength,
int apartmentNumber, String filePath) {
60
61         Document theList = new Document();
62
63         ArrayList <String> emailList = waitinglist.
getWaitinglist(listLength, apartmentNumber);
64
65         loopThroughEmailList(emailList, theList, filePath);
66     }
67
68     @Override
69     public void generateCompleteListPDF(int listLength, int
priority, String filePath) {
70
71         Document theList = new Document();
72
73         ArrayList <String> emailList = waitinglist.
getSingleWaitinglist(listLength, priority);
74
75         loopThroughEmailList(emailList, theList, filePath);
76     }
77
78
79     /**
80      * The generateList method uses the
getSingleWaitinglist method
81      * from the waitinglist service to generate a waiting

```



```

81 list for a
82 * certain priority, as described above. It takes in
83 the
84 * parameters needed, being the length, or number of
85 users, in
86 * the list as well as the priority or type of waiting
87 list.
88 * It returns a list of the desired type, for
89 displaying in a
90 * table on screen.
91 * @param length
92 * @param priority
93 * @return
94 */
95
96 public List<User> generateList(int length, int
priority) {
97     List<User> generatedList = new ArrayList<>();
98     ArrayList<String> emailList = waitinglist.
getSingleWaitinglist(Integer.MAX_VALUE, priority);
99     for (String email: emailList) {
100         User listUser = userService.findUserByEmail(
email);
101         generatedList.add(listUser);
102     }
103     return generatedList;
104 }
105
106 /**
107 * The generateSingleApartmentList method uses the
108 getWaitinglist method
109 * from the waitinglist service to generate a waiting
110 list for a
111 * certain apartment. It takes in the
112 * parameters needed, being the length, or number of
113 users, in
114 * the list as well as the id of the apartment that is
115 being sold.
116 * It returns a list of the desired type, for
117 displaying in a
118 * table on screen.
119 * @param length
120 * @param ApartmentId
121 * @return
122 */

```

```

119     public List<User> generateSingleApartmentList(int
length, int ApartmentId) {
120         List<User> generatedApartmentList = new ArrayList<
>();
121         ArrayList<String> emailList = waitinglist.
getWaitinglist(Integer.MAX_VALUE, ApartmentId);
122         if (emailList == null) {
123             return null;
124         }
125         for (String email: emailList) {
126             User listUser = userService.findUserByEmail(
email);
127             generatedApartmentList.add(listUser);
128         }
129         return generatedApartmentList;
130     }
131
132     @Override
133     public void loopThroughEmailList(ArrayList <String>
emailList, Document theList, String filePath) {
134         try {
135             PdfWriter.getInstance(theList, new
FileOutputStream(new File(filePath))); // filePath
136             // ???
137             theList.open();
138
139             Header h = new Header("listHeader", "New
Apartment List"); // evt dags dato + listetype + lejliged
, eks. "010217_intern_ for lejliged_23"
140
141             theList.add(h);
142
143             for (String email: emailList) {
144                 User listUser = userService.
findUserByEmail(email);
145                 Paragraph p = new Paragraph();
146                 Chunk seniority = new Chunk("\nAncienittet
: " + (emailList.indexOf(email) + 1) + "\n", theFont);
147                 Chunk name = new Chunk("Navn: " + listUser
.getName() + " " + listUser.getLastName() + "\n",
theSmallFont);
148                 Chunk phoneNumber = new Chunk("
Telefonnummer: " + listUser.getPhoneNumber() + "\n",
theSmallFont);
149                 Chunk user_email = new Chunk("E-mail: " +
listUser.getEmail() + "\n", theSmallFont);
150                 p.add(seniority);
151                 p.add(name);
152                 p.add(phoneNumber);

```

```
153         p.add(user_email);
154         p.setAlignment(Element.ALIGN_CENTER);
155         theList.add(p);
156         Paragraph p2 = new Paragraph();
157         p2.add("
158         p2.setAlignment(Element.ALIGN_CENTER);
159         theList.add(p2);
160     }
161
162     theList.close();
163
164     } catch (FileNotFoundException | DocumentException
165     e) {
166         e.printStackTrace();
167     }
168
169 }
170
171
172
```

```
1 package dk.binfo.services;
2
3 import java.util.*;
4
5 import javax.transaction.Transactional;
6
7 import dk.binfo.models.Role;
8 import dk.binfo.models.User;
9 import dk.binfo.repositories.RoleRepository;
10 import dk.binfo.repositories.UserRepository;
11 import org.springframework.beans.factory.annotation.
    Autowired;
12 import org.springframework.security.core.GrantedAuthority;
13 import org.springframework.security.core.authority.
    SimpleGrantedAuthority;
14 import org.springframework.security.core.userdetails.
    UserDetails;
15 import org.springframework.security.core.userdetails.
    UserDetailsService;
16 import org.springframework.security.core.userdetails.
    UsernameNotFoundException;
17 import org.springframework.security.crypto.bcrypt.
    BCryptPasswordEncoder;
18 import org.springframework.stereotype.Service;
19
20 /**
21  * @author Patrick Klæbel
22  * @author Jens Bäckvall
23  * @author Steen Petersen
24  * @author Morten Olsen
25  *
26  */
27 @Service("userService")
28 public class UserServiceImpl implements UserService,
    UserDetailsService {
29
30     @Autowired
31     private UserRepository userRepository;
32     @Autowired
33     private RoleRepository roleRepository;
34     @Autowired
35     private BCryptPasswordEncoder bCryptPasswordEncoder;
36
37     @Override
38     public User findUserByEmail(String email) {
39
40         return userRepository.findByEmail(email);
41     }
42
43 }
```

```

44     @Override
45     @Transactional
46     public List<User> findAll() {
47         return userRepository.findAll();
48     }
49
50     @Override
51     public void register(User user) {
52         user.setPassword(bCryptPasswordEncoder.encode(user.
53             getPassword()));
54         user.setActive(true);
55         user.setPhoneNumber(user.getPhoneNumber());
56         Role userRole = roleRepository.findByRole("user");
57         user.setRoles(new HashSet<Role>(Arrays.asList(
58             userRole)));
59         userRepository.save(user);
60     }
61
62     @Override
63     @Transactional
64     public User deleteUser(String email) {
65         User deletedUser = userRepository.findByEmail(email
66             );
67         deletedUser.setRoles(null); // Deletes this user in
68         user_role table
69         userRepository.delete(deletedUser);
70         return deletedUser;
71     }
72
73     @Override
74     @Transactional
75     public User update(User user){
76         User updatedUser = userRepository.findByEmail(user.
77             getEmail());
78         updatedUser.setName(user.getName());
79         updatedUser.setLastName(user.getLastName());
80         updatedUser.setPhoneNumber(user.getPhoneNumber());
81         updatedUser.setActive(user.isActive());
82
83         return updatedUser;
84     }
85
86     @Override
87     public void adminRegisterUser(User user) {
88         Random ran = new Random();
89         String Password = generateString(ran,"
90             qwertyuiopasdfghjklzxcvbnm",8);
91         user.setPassword(bCryptPasswordEncoder.encode(
92             Password));
93         user.setActive(user.isActive());

```

```

87         user.setPhoneNumber(user.getPhoneNumber());
88         Role userRole = roleRepository.findByRole("user");
89         user.setRoles(new HashSet<Role>(Arrays.asList(
    userRole)));
90         userRepository.save(user);
91     }
92
93     @Override
94     @Transactional
95     public User updateUserSettings(User user){
96         User updateUser = userRepository.findByEmail(user.
    getEmail());
97         if(user.getPassword().equalsIgnoreCase("") && user
    .getPhoneNumber() != null) {
98             updateUser.setPhoneNumber(user.getPhoneNumber(
    ));
99         }
100        if(user.getPhoneNumber() == null || user.
    getPhoneNumber().equalsIgnoreCase("") && user.getPassword(
    ) != null) {
101            updateUser.setPassword(bCryptPasswordEncoder.
    encode(user.getPassword()));
102        }
103        if(user.getPhoneNumber() != null && user.
    getPassword() != null) {
104            updateUser.setPassword(bCryptPasswordEncoder.
    encode(user.getPassword()));
105            updateUser.setPhoneNumber(user.getPhoneNumber(
    ));
106        }
107        return updateUser;
108    }
109
110    @Override
111    @Transactional
112    public UserDetails loadUserByUsername(String userName)
    throws UsernameNotFoundException {
113        User user = userRepository.findByEmail(userName);
114        List<GrantedAuthority> authorities =
    getUserAuthority(user.getRoles());
115        return buildUserForAuthentication(user,
    authorities);
116    }
117
118    private List<GrantedAuthority> getUserAuthority(Set<
    Role> userRoles) {
119        Set<GrantedAuthority> roles = new HashSet<
    GrantedAuthority>();
120        for (Role role : userRoles) {
121            roles.add(new SimpleGrantedAuthority(role.

```

```
121 getRole()));  
122     }  
123  
124     List<GrantedAuthority> grantedAuthorities = new  
    ArrayList<GrantedAuthority>(roles);  
125     return grantedAuthorities;  
126 }  
127  
128 private UserDetails buildUserForAuthentication(User  
user, List<GrantedAuthority> authorities) {  
129     return new org.springframework.security.core.  
   .userdetails.User(user.getEmail(), user.getPassword(), user  
    .isActive(), true, true, true, authorities);  
130 }  
131  
132  
133 public static String generateString(Random rng, String  
    characters, int length)  
134 {  
135     char[] text = new char[length];  
136     for (int i = 0; i < length; i++)  
137     {  
138         text[i] = characters.charAt(rng.nextInt(  
            characters.length()));  
139     }  
140     return new String(text);  
141 }  
142 }  
143
```

```
1 package dk.binfo.services;
2
3 import dk.binfo.models.Apartment;
4
5 import java.util.List;
6 /**
7  * @author Patrick Klæbel
8  * @author Jens Bäckvall
9  * @author Steen Petersen
10 * @author Morten Olsen
11 *
12 */
13 public interface ApartmentService {
14     Apartment delete(int id);
15     List<Apartment> findAll();
16     Apartment update(Apartment apartment);
17     Apartment findById(int id);
18
19     //Saves the Apartment setters.
20     void saveApartment(Apartment apartment);
21 }
22
```



```

1 package dk.binfo.controllers;
2
3 import dk.binfo.models.User;
4 import dk.binfo.services.ListService;
5 import dk.binfo.services.UserService;
6 import org.springframework.beans.factory.annotation.
    Autowired;
7 import org.springframework.security.core.Authentication;
8 import org.springframework.security.core.context.
    SecurityContextHolder;
9 import org.springframework.stereotype.Controller;
10 import org.springframework.web.bind.annotation.PathVariable
    ;
11 import org.springframework.web.bind.annotation.
    RequestMapping;
12 import org.springframework.web.bind.annotation.
    RequestMethod;
13 import org.springframework.web.servlet.ModelAndView;
14
15 import javax.servlet.ServletContext;
16 import javax.servlet.http.HttpServletRequest;
17 import javax.servlet.http.HttpServletResponse;
18 import java.io.File;
19 import java.io.IOException;
20
21 /**
22  * @author Jens Bäckvall
23  * @author Steen Petersen
24  */
25 @Controller
26 public class PDFController {
27
28     @Autowired
29     private UserService userService;
30
31     @Autowired
32     private ListService listService;
33
34     @RequestMapping(value={"/lists/connect/pdf"})
35     public ModelAndView generateConnectListPDF() {
36         ModelAndView modelAndView = new ModelAndView("/
    lists/connect", "list", listService
37             .generateList(Integer.MAX_VALUE, 1));
38         String filePath = "/Users/jensbackvall/Desktop/
    PDF_TEST/Sammenlaegningsliste.pdf"; // filePath
39         listService.generateCompleteListPDF(Integer.
    MAX_VALUE, 1, filePath);
40         Authentication auth = SecurityContextHolder.
    getContext().getAuthentication();
41         User user = userService.findUserByEmail(auth.

```

```

41 getName());
42     modelAndView.addObject("adminMessage","Du er logget
    ind som spadmin");
43     modelAndView.addObject("PDFMessage","PDF er gemt på
    /Users/jensbackvall/Desktop/PDF_TEST/");
44     modelAndView.addObject(user);
45     return modelAndView;
46 }
47
48 @RequestMapping(value={"/lists/internal/pdf"})
49 public ModelAndView generateInternListPDF() {
50     ModelAndView modelAndView = new ModelAndView("/
    lists/internal", "list", listService
51         .generateList(Integer.MAX_VALUE, 2));
52     String filePath = "/Users/jensbackvall/Desktop/
    PDF_TEST/Intern_liste.pdf"; // filePath
53     listService.generateCompleteListPDF(Integer.
    MAX_VALUE, 2, filePath);
54     Authentication auth = SecurityContextHolder.
    getContext().getAuthentication();
55     User user = userService.findUserByEmail(auth.
    getName());
56     modelAndView.addObject("adminMessage","Du er logget
    ind som spadmin");
57     modelAndView.addObject("PDFMessage","PDF er gemt på
    /Users/jensbackvall/Desktop/PDF_TEST/");
58     modelAndView.addObject(user);
59     return modelAndView;
60 }
61
62 @RequestMapping(value={"/lists/family/pdf"})
63 public ModelAndView generateFamilyListPDF() {
64     ModelAndView modelAndView = new ModelAndView("/
    lists/family", "list", listService
65         .generateList(Integer.MAX_VALUE, 3));
66     String filePath = "/Users/jensbackvall/Desktop/
    PDF_TEST/Familieliste.pdf"; // filePath
67     listService.generateCompleteListPDF(Integer.
    MAX_VALUE, 3, filePath);
68     Authentication auth = SecurityContextHolder.
    getContext().getAuthentication();
69     User user = userService.findUserByEmail(auth.
    getName());
70     modelAndView.addObject("adminMessage","Du er logget
    ind som spadmin");
71     modelAndView.addObject("PDFMessage","PDF er gemt på
    /Users/jensbackvall/Desktop/PDF_TEST/");
72     modelAndView.addObject(user);
73     return modelAndView;
74 }

```

```

75
76     @RequestMapping(value={"/lists/external/pdf"})
77     public ModelAndView generateExternalListPDF() {
78         ModelAndView modelAndView = new ModelAndView("/
lists/external", "list", listService.generateList(Integer.
MAX_VALUE, 4));
79         String filePath = "/Users/jensbackvall/Desktop/
PDF_TEST/Ekstern_liste.pdf"; // filePath
80         listService.generateCompleteListPDF(Integer.
MAX_VALUE, 4, filePath);
81         Authentication auth = SecurityContextHolder.
getContext().getAuthentication();
82         User user = userService.findUserByEmail(auth.
getName());
83         modelAndView.addObject("adminMessage", "Du er
logget ind som spadmin");
84         modelAndView.addObject("PDFMessage", "PDF er gemt
på /Users/jensbackvall/Desktop/PDF_TEST/");
85         modelAndView.addObject(user);
86         return modelAndView;
87     }
88
89     @RequestMapping(value={"/lists/listapartmentPDF/{id}"}, method = RequestMethod.GET)
90     public ModelAndView showSingleApartmentList(@
PathVariable("id") Integer id) {
91         ModelAndView modelAndView = new ModelAndView("/
lists/listapartment", "list", listService.
generateSingleApartmentList(Integer.MAX_VALUE, id));
92         String filePath = "/Users/jensbackvall/Desktop/
PDF_TEST/Liste_for_lejlighed_" + id + ".pdf"; // filePath
93         listService.generateSingleApartmentPDF(Integer.
MAX_VALUE, id, filePath);
94         Authentication auth = SecurityContextHolder.
getContext().getAuthentication();
95         User user = userService.findUserByEmail(auth.
getName());
96         modelAndView.addObject("adminMessage", "Du er
logget ind som spadmin");
97         modelAndView.addObject("PDFMessage", "PDF er gemt
på /Users/jensbackvall/Desktop/PDF_TEST/");
98         modelAndView.addObject("HeaderMessage", "Viser
Liste for Lejlighed nr. " + id);
99         modelAndView.addObject(user);
100        return modelAndView;
101    }
102 }
103

```

```
1 package dk.binfo.controllers;
2
3 import dk.binfo.models.User;
4 import dk.binfo.services.UserService;
5 import org.springframework.beans.factory.annotation.
    Autowired;
6 import org.springframework.security.core.Authentication;
7 import org.springframework.security.core.context.
    SecurityContextHolder;
8 import org.springframework.stereotype.Controller;
9 import org.springframework.web.bind.annotation.
    RequestMapping;
10 import org.springframework.web.bind.annotation.
    RequestMethod;
11 import org.springframework.web.servlet.ModelAndView;
12
13 /**
14  * @author Morten Olsen
15  */
16 @Controller
17 public class HomeController {
18
19     @Autowired
20     private UserService userService;
21
22     @RequestMapping(value= "/home", method = RequestMethod.
    GET)
23     public ModelAndView userHome(){
24         ModelAndView modelAndView = new ModelAndView();
25         Authentication auth = SecurityContextHolder.
    getContext().getAuthentication();
26         User user = userService.findUserByEmail(auth.
    getName());
27         modelAndView.addObject("user", user);
28         modelAndView.addObject("userMessage","Du er logget
    ind");
29         modelAndView.setViewName("/home");
30         return modelAndView;
31     }
32 }
33
```

```

1 package dk.binfo.controllers;
2
3 import dk.binfo.models.User;
4 import dk.binfo.services.ListService;
5 import dk.binfo.services.UserService;
6 import org.springframework.beans.factory.annotation.
    Autowired;
7 import org.springframework.security.core.Authentication;
8 import org.springframework.security.core.context.
    SecurityContextHolder;
9 import org.springframework.stereotype.Controller;
10 import org.springframework.web.bind.annotation.PathVariable
    ;
11 import org.springframework.web.bind.annotation.
    RequestMapping;
12 import org.springframework.web.bind.annotation.
    RequestMethod;
13 import org.springframework.web.servlet.ModelAndView;
14
15
16 /**
17  * @author Patrick Klæbel
18  * @author Jens Bäckvall
19  * @author Steen Petersen
20  * @author Morten Olsen
21  *
22  */
23 @Controller
24 public class ListController {
25
26     @Autowired
27     private UserService userService;
28
29     @Autowired
30     private ListService listService;
31
32
33     @RequestMapping(value={"/lists/connect"})
34     public ModelAndView showConnectList() {
35         ModelAndView modelAndView = new ModelAndView("/
    lists/connect", "list", listService
36             .generateList(Integer.MAX_VALUE, 1));
37         Authentication auth = SecurityContextHolder.
    getContext().getAuthentication();
38         User user = userService.findUserByEmail(auth.
    getName());
39         modelAndView.addObject(user);
40         modelAndView.addObject("adminMessage","Du er logget
    ind som spadmin");
41         return modelAndView;

```

```

42     }
43
44
45     @RequestMapping(value={"/lists/internal"})
46     public ModelAndView showInternList() {
47         ModelAndView modelAndView = new ModelAndView("/
48 lists/internal", "list", listService
49         .generateList(20, 2));
50         Authentication auth = SecurityContextHolder.
51         getContext().getAuthentication();
52         User user = userService.findUserByEmail(auth.
53         getName());
54         modelAndView.addObject(user);
55         modelAndView.addObject("adminMessage","Du er logget
56 ind som spadmin");
57         return modelAndView;
58     }
59
60     @RequestMapping(value={"/lists/family"})
61     public ModelAndView showFamilyList() {
62         ModelAndView modelAndView = new ModelAndView("/
63 lists/family", "list", listService
64         .generateList(Integer.MAX_VALUE, 3));
65         Authentication auth = SecurityContextHolder.
66         getContext().getAuthentication();
67         User user = userService.findUserByEmail(auth.
68         getName());
69         modelAndView.addObject(user);
70         modelAndView.addObject("adminMessage","Du er logget
71 ind som spadmin");
72         return modelAndView;
73     }
74
75     @RequestMapping(value={"/lists/external"})
76     public ModelAndView showExternalList() {
77         ModelAndView modelAndView = new ModelAndView("/
78 lists/external", "list", listService.generateList(Integer.
79 MAX_VALUE, 4));
80         Authentication auth = SecurityContextHolder.
81         getContext().getAuthentication();
82         User user = userService.findUserByEmail(auth.
83         getName());
84         modelAndView.addObject(user);
85         modelAndView.addObject("adminMessage","Du er logget
86 ind som spadmin");
87         return modelAndView;
88     }
89
90     @RequestMapping(value={"/lists/listapartment/{id}"},
91     method = RequestMethod.GET)

```

```
78     public ModelAndView showSingleApartmentList(@
    PathVariable("id") Integer id) {
79         ModelAndView modelAndView = new ModelAndView("/
    lists/listapartment", "list", listService.
    generateSingleApartmentList(Integer.MAX_VALUE, id));
80         Authentication auth = SecurityContextHolder.
    getContext().getAuthentication();
81         User user = userService.findUserByEmail(auth.
    getName());
82         modelAndView.addObject(user);
83         modelAndView.addObject("adminMessage", "Du er
    logget ind som admin");
84         modelAndView.addObject("HeaderMessage", "Viser
    Liste for Lejlighed nr. " + id);
85         modelAndView.setViewName("/lists/listapartment");
86         return modelAndView;
87     }
88
89 }
90
```

```
1 package dk.binfo.controllers;
2
3
4 import dk.binfo.models.User;
5 import dk.binfo.services.UserService;
6 import org.springframework.beans.factory.annotation.
    Autowired;
7 import org.springframework.security.core.Authentication;
8 import org.springframework.security.core.context.
    SecurityContextHolder;
9 import org.springframework.stereotype.Controller;
10 import org.springframework.validation.BindingResult;
11 import org.springframework.web.bind.annotation.
    ModelAttribute;
12 import org.springframework.web.bind.annotation.PathVariable
    ;
13 import org.springframework.web.bind.annotation.
    RequestMapping;
14 import org.springframework.web.bind.annotation.
    RequestMethod;
15 import org.springframework.web.servlet.ModelAndView;
16
17 import javax.validation.Valid;
18 /**
19  * @author Jens Bäckvall
20  * @author Morten Olsen
21  */
22 @Controller
23 public class UserController {
24
25     @Autowired
26     private UserService userService;
27
28     @RequestMapping(value={"/settings/{email:.*}"}, method
    = RequestMethod.GET)
29     public ModelAndView settings(@PathVariable String email
    ){
30         ModelAndView modelAndView = new ModelAndView();
31         Authentication auth = SecurityContextHolder.
    getContext().getAuthentication();
32         User user = userService.findUserByEmail(auth.
    getName());
33         modelAndView.addObject("user", user);
34         modelAndView.addObject("userForm", user);
35         modelAndView.setViewName("/settings");
36         return modelAndView;
37     }
38
39     @RequestMapping(value={"/settings/{email:.*}"}, method=
    RequestMethod.POST)
```



```
40     public ModelAndView editPhoneSettings(@ModelAttribute @
Valid User userinfo, BindingResult bindingResult, @
PathVariable String email){
41         ModelAndView modelAndView = new ModelAndView();
42         Authentication auth = SecurityContextHolder.
getContext().getAuthentication();
43         User user = userService.findUserByEmail(auth.
getName());
44         modelAndView.addObject("user", user);
45
46         if (bindingResult.hasErrors())
47         {
48             modelAndView.setViewName("/settings/{email}");
49         }
50         modelAndView.setViewName("redirect:/settings/{email
}");
51         userService.updateUserSettings(userinfo);
52         return modelAndView;
53     }
54 }
55
```

```
1 package dk.binfo.controllers;
2
3 import dk.binfo.models.User;
4 import dk.binfo.services.UserService;
5 import org.springframework.beans.factory.annotation.
    Autowired;
6 import org.springframework.security.core.Authentication;
7 import org.springframework.security.core.context.
    SecurityContextHolder;
8 import org.springframework.stereotype.Controller;
9 import org.springframework.validation.BindingResult;
10 import org.springframework.web.bind.annotation.
    ModelAttribute;
11 import org.springframework.web.bind.annotation.PathVariable
    ;
12 import org.springframework.web.bind.annotation.
    RequestMapping;
13 import org.springframework.web.bind.annotation.
    RequestMethod;
14 import org.springframework.web.servlet.ModelAndView;
15
16 import javax.validation.Valid;
17 /**
18  * @author Patrick Klæbel
19  * @author Jens Bäckvall
20  */
21 @Controller
22 public class LoginController {
23
24     @Autowired
25     private UserService userService;
26
27     @RequestMapping(value={"/login", "/"}, method =
    RequestMethod.GET)
28     public ModelAndView login(){
29         ModelAndView modelAndView = new ModelAndView();
30         modelAndView.setViewName("login");
31         return modelAndView;
32     }
33 }
34
```

```
1 package dk.binfo.controllers;
2
3 import dk.binfo.models.User;
4 import dk.binfo.services.UserService;
5 import org.springframework.beans.factory.annotation.
    Autowired;
6 import org.springframework.security.core.Authentication;
7 import org.springframework.security.core.context.
    SecurityContextHolder;
8 import org.springframework.stereotype.Controller;
9 import org.springframework.validation.BindingResult;
10 import org.springframework.web.bind.annotation.
    ModelAttribute;
11 import org.springframework.web.bind.annotation.PathVariable
    ;
12 import org.springframework.web.bind.annotation.
    RequestMapping;
13 import org.springframework.web.bind.annotation.
    RequestMethod;
14 import org.springframework.web.servlet.ModelAndView;
15
16 import javax.validation.Valid;
17 /**
18  * @author Patrick Klæbel
19  * @author Jens Bäckvall
20  * @author Steen Petersen
21  * @author Morten Olsen
22  *
23  */
24 @Controller
25 public class UsersController {
26
27     @Autowired
28     private UserService userService;
29
30     @RequestMapping("/users")
31     public ModelAndView showUsers() {
32         ModelAndView modelAndView = new ModelAndView("/
    users", "users", userService.findAll());
33         Authentication auth = SecurityContextHolder.
    getContext().getAuthentication();
34         User user = userService.findUserByEmail(auth.
    getName());
35         modelAndView.addObject("user", user);
36         modelAndView.setViewName("/users");
37         return modelAndView;
38     }
39
40     @RequestMapping("/users/add")
41     public ModelAndView adminCreateNewUser(){
```

```

42     ModelAndView modelAndView = new ModelAndView();
43     Authentication auth = SecurityContextHolder.
getContext().getAuthentication();
44     User user = userService.findUserByEmail(auth.
getName());
45     modelAndView.addObject("user", user);
46     modelAndView.addObject("users", new User());
47     modelAndView.setViewName("/users/add");
48     return modelAndView;
49 }
50
51 @RequestMapping(value = "/users/add", method =
RequestMethod.POST)
52 public ModelAndView adminCreateUser(@Valid User users,
BindingResult bindingResult) {
53     ModelAndView modelAndView = new ModelAndView();
54     Authentication auth = SecurityContextHolder.
getContext().getAuthentication();
55     User user = userService.findUserByEmail(auth.
getName());
56     modelAndView.addObject("user", user);
57     User userExists = userService.findUserByEmail(users
.getEmail());
58     if (userExists != null) {
59         bindingResult.rejectValue("email", "error.user"
, "Der eksisterer allerede en bruger med den angivne email"
);
60     }
61     else {
62         userService.adminRegisterUser(users);
63         modelAndView.addObject("successMessage", "
SUCCES!: Du har tilføjet en ny bruger.");
64         modelAndView.addObject("users", new User());
65         modelAndView.setViewName("/users/add");
66     }
67     return modelAndView;
68 }
69
70 @RequestMapping(value="/users/delete/{email:.+}",
method=RequestMethod.GET)
71 public ModelAndView deleteApartment(@PathVariable
String email) {
72     ModelAndView modelAndView = new ModelAndView("
redirect:/users");
73     userService.deleteUser(email);
74     return modelAndView;
75 }
76
77 @RequestMapping(value="/users/edit/{email:.+}", method=
RequestMethod.GET)

```

```
78     public ModelAndView editUserPage(@PathVariable String
email) {
79         ModelAndView modelAndView = new ModelAndView("/
users/edit");
80         Authentication auth = SecurityContextHolder.
getContext().getAuthentication();
81         User user = userService.findUserByEmail(auth.
getName());
82         User editedUser = userService.findUserByEmail(
email);
83         modelAndView.addObject("user", user);
84         modelAndView.addObject("users", editedUser);
85         return modelAndView;
86     }
87
88     @RequestMapping(value="/users/edit/{email:.+}", method
=RequestMethod.POST)
89     public ModelAndView editApartment(@ModelAttribute @
Valid User editedUser, BindingResult bindingResult){
90         ModelAndView modelAndView = new ModelAndView();
91         Authentication auth = SecurityContextHolder.
getContext().getAuthentication();
92         User user = userService.findUserByEmail(auth.
getName());
93         modelAndView.addObject("user", user);
94         if (bindingResult.hasErrors())
95         {
96             modelAndView.setViewName("/users/edit");
97         }
98         modelAndView.setViewName("redirect:/users");
99         userService.update(editedUser);
100        return modelAndView;
101    }
102
103 }
104
105
```

```
1 package dk.binfo.controllers;
2
3 import dk.binfo.models.User;
4 import dk.binfo.services.UserService;
5 import org.springframework.beans.factory.annotation.
    Autowired;
6 import org.springframework.stereotype.Controller;
7 import org.springframework.validation.BindingResult;
8 import org.springframework.web.bind.annotation.
    RequestMapping;
9 import org.springframework.web.bind.annotation.
    RequestMethod;
10 import org.springframework.web.servlet.ModelAndView;
11
12 import javax.validation.Valid;
13 /**
14  * @author Jens Bäckvall
15  * @author Morten Olsen
16  */
17 @Controller
18 public class RegisterController {
19
20     @Autowired
21     private UserService userService;
22
23     @RequestMapping(value={"/registration", "/register"},
    method = RequestMethod.GET)
24     public ModelAndView registration(){
25         ModelAndView modelAndView = new ModelAndView();
26         modelAndView.addObject("user", new User());
27         modelAndView.setViewName("registration");
28         return modelAndView;
29     }
30
31     @RequestMapping(value = "/registration", method =
    RequestMethod.POST)
32     public ModelAndView createUser(@Valid User user,
    BindingResult bindingResult) {
33         ModelAndView modelAndView = new ModelAndView();
34         User userExists = userService.findUserByEmail(user.
    getEmail());
35         if (userExists != null) {
36             bindingResult.rejectValue("email", "error.user"
    , "Der eksisterer allerede en bruger med den angivne email"
    );
37         }
38         if (bindingResult.hasErrors()) {
39             modelAndView.setViewName("registration");
40         } else {
41             userService.register(user);
```

```
42         modelAndView.addObject("successMessage", "
    SUCCES!: Du har tilføjet en ny bruger.");
43         modelAndView.addObject("user", new User());
44         modelAndView.setViewName("registration");
45     }
46     return modelAndView;
47 }
48 }
49
```

```
1 package dk.binfo.controllers;
2
3 import dk.binfo.models.Apartment;
4 import dk.binfo.repositories.ApartmentRepository;
5 import dk.binfo.services.ApartmentService;
6 import org.springframework.beans.factory.annotation.
    Autowired;
7 import org.springframework.stereotype.Controller;
8 import org.springframework.validation.BindingResult;
9 import org.springframework.web.bind.annotation.
    ModelAttribute;
10 import org.springframework.web.bind.annotation.PathVariable
    ;
11 import org.springframework.web.bind.annotation.
    RequestMapping;
12 import org.springframework.web.bind.annotation.
    RequestMethod;
13 import org.springframework.web.servlet.ModelAndView;
14 import dk.binfo.models.User;
15 import dk.binfo.services.UserService;
16 import org.springframework.security.core.Authentication;
17 import org.springframework.security.core.context.
    SecurityContextHolder;
18
19 import javax.validation.Valid;
20
21 @Controller
22 public class ApartmentController {
23
24     /**
25      * @author Patrick Klæbel
26      * @author Steen Petersen
27      * @author Morten Olsen
28      */
29
30     @Autowired
31     private ApartmentRepository repository;
32
33     @Autowired
34     private UserService userService;
35
36     @Autowired
37     private ApartmentService apartmentService;
38
39     @RequestMapping("/apartment")
40     public ModelAndView showApartment() {
41         ModelAndView modelAndView = new ModelAndView("/
    apartment", "apartment", repository.findAll());
42         Authentication auth = SecurityContextHolder.
    getContext().getAuthentication();
```



```

43         User user = userService.findUserByEmail(auth.
getName());
44         modelAndView.addObject("user", user);
45         modelAndView.addObject("adminMessage", "Du er logget
ind som spadmin");
46         modelAndView.setViewName("/apartment");
47
48         return modelAndView;
49     }
50
51     @RequestMapping(value = "/apartment/add", method =
RequestMethod.POST)
52     public ModelAndView createNewApartment(@Valid Apartment
apartment, BindingResult bindingResult) {
53         ModelAndView modelAndView = new ModelAndView();
54         Authentication auth = SecurityContextHolder.
getContext().getAuthentication();
55         User user = userService.findUserByEmail(auth.
getName());
56         if (bindingResult.hasErrors()) {
57             modelAndView.setViewName("/apartment/add");
58         } else {
59             apartmentService.saveApartment(apartment);
60             modelAndView.addObject("successMessage", "
Fantastisk arbejde! Du har nu tilføjet en ny lejlighed. Du
ROCKER!!!");
61             modelAndView.addObject("user", user);
62             modelAndView.addObject("apartment", new
Apartment());
63             modelAndView.setViewName("/apartment/add");
64
65         }
66         return modelAndView;
67     }
68
69     @RequestMapping("/apartment/add")
70     public ModelAndView add(){
71         ModelAndView modelAndView = new ModelAndView();
72         Authentication auth = SecurityContextHolder.
getContext().getAuthentication();
73         User user = userService.findUserByEmail(auth.
getName());
74         modelAndView.addObject("user", user);
75         modelAndView.addObject("adminMessage", "Fedt man spa
du er admin");
76         modelAndView.addObject("apartment", new Apartment()
);
77         modelAndView.setViewName("/apartment/add");
78         return modelAndView;
79     }

```

```
80
81     @RequestMapping(value="/apartment/edit/{id}", method=
RequestMethod.GET)
82     public ModelAndView editApartmentPage(@PathVariable
Integer id) {
83         ModelAndView modelAndView = new ModelAndView("/
apartment/add-edit");
84         Authentication auth = SecurityContextHolder.
getContext().getAuthentication();
85         User user = userService.findUserByEmail(auth.
getName());
86         modelAndView.addObject("user", user);
87         Apartment apartment = apartmentService.findById(id
);
88         modelAndView.addObject("apartment", apartment);
89         return modelAndView;
90     }
91
92     @RequestMapping(value="/apartment/edit/{id}", method=
RequestMethod.POST)
93     public ModelAndView editApartment(@ModelAttribute @
Valid Apartment apartment, BindingResult bindingResult, @
PathVariable Integer id){
94         ModelAndView modelAndView = new ModelAndView();
95         Authentication auth = SecurityContextHolder.
getContext().getAuthentication();
96         User user = userService.findUserByEmail(auth.
getName());
97         modelAndView.addObject("user", user);
98         if (bindingResult.hasErrors())
99         {
100             modelAndView.setViewName("/apartment/edit");
101         }
102         modelAndView.setViewName("redirect:/apartment");
103         apartmentService.update(apartment);
104         return modelAndView;
105     }
106
107     @RequestMapping(value="/apartment/delete/{id}", method
=RequestMethod.GET)
108     public ModelAndView deleteApartment(@PathVariable
Integer id) {
109         ModelAndView modelAndView = new ModelAndView("
redirect:/apartment");
110         apartmentService.delete(id);
111         return modelAndView;
112     }
113
114 }
```

```
1 package dk.binfo.controllers;
2
3
4 import dk.binfo.models.User;
5 import dk.binfo.services.UserService;
6 import org.springframework.beans.factory.annotation.
    Autowired;
7 import org.springframework.security.core.Authentication;
8 import org.springframework.security.core.context.
    SecurityContextHolder;
9 import org.springframework.stereotype.Controller;
10 import org.springframework.web.bind.annotation.
    RequestMapping;
11 import org.springframework.web.bind.annotation.
    RequestMethod;
12 import org.springframework.web.servlet.ModelAndView;
13 /**
14  * @author Patrick Klæbel
15  *
16  */
17 @Controller
18 public class AccessDeniedController {
19
20     @Autowired
21     private UserService userService;
22
23     @RequestMapping(value = {"/accessDenied"}, method =
    RequestMethod.GET)
24     public ModelAndView accessDenied() {
25         ModelAndView modelAndView = new ModelAndView();
26         Authentication auth = SecurityContextHolder.
    getContext().getAuthentication();
27         User user = userService.findUserByEmail(auth.
    getName());
28         modelAndView.addObject("user", user);
29         modelAndView.setViewName("/accessDenied");
30         return modelAndView;
31     }
32 }
33
```

```

1 package dk.binfo.controllers;
2
3 import dk.binfo.models.User;
4 import dk.binfo.repositories.UserRepository;
5 import dk.binfo.services.EmailService;
6 import dk.binfo.services.UserService;
7 import org.springframework.beans.factory.annotation.
    Autowired;
8 import org.springframework.stereotype.Controller;
9 import org.springframework.security.crypto.bcrypt.
    BCryptPasswordEncoder;
10 import org.springframework.web.bind.annotation.PathVariable
    ;
11 import org.springframework.web.bind.annotation.
    RequestMapping;
12 import org.springframework.web.bind.annotation.
    RequestMethod;
13 import org.springframework.web.bind.annotation.RequestParam
    ;
14 import org.springframework.web.servlet.ModelAndView;
15
16 /**
17  * @author Steen Petersen
18  *
19  */
20 @Controller
21 public class ForgotPasswordController {
22
23     @Autowired
24     private UserService userService;
25
26     @Autowired
27     private EmailService emailService;
28
29     @Autowired
30     private UserRepository userRepository;
31
32     @Autowired
33     private BCryptPasswordEncoder bCryptPasswordEncoder;
34
35     @RequestMapping(value={"/forgotpassword"}, method =
    RequestMethod.GET)
36     public ModelAndView forgotPassword(){
37         ModelAndView modelAndView = new ModelAndView();
38         modelAndView.setViewName("forgotpassword");
39         return modelAndView;
40     }
41
42     @RequestMapping(value={"/forgotpassword/{password}/{
    email}/newpassword"}, method = RequestMethod.GET)

```

```

43     public ModelAndView forgotPassword(@PathVariable("email"
    ") String email, @PathVariable("password") String password)
    {
44         ModelAndView modelAndView = new ModelAndView();
45         modelAndView.setViewName("forgotpasswordlink");
46         User user = userService.findUserByEmail(email);
47         if (user == null) {
48             modelAndView.setViewName("redirect:/login");
49         }else{
50             if (user.getPassword().replace("/", "").equals(
password)){
51
52                 }else{
53                     modelAndView.setViewName("redirect:/login")
;
54                 }
55             }
56             return modelAndView;
57         }
58
59         @RequestMapping(value={"/forgotpassword/{password}/{
email}/newpassword"}, method = RequestMethod.POST)
60         public ModelAndView forgotPassword(@RequestParam String
password1, @RequestParam String password2, @PathVariable
61         ("email") String email, @PathVariable("password"
") String password){
62             ModelAndView modelAndView = new ModelAndView();
63             modelAndView.setViewName("forgotpasswordlink");
64             User user = userService.findUserByEmail(email);
65             if (user == null) {
66                 modelAndView.addObject("Message", "Linket
virker ikke, prøv igen.");
67             }else{
68                 if (user.getPassword().replace("/", "").equals(
password)){
69                     if (password1.equals(password2)){
70                         modelAndView.addObject("Message", "Din
kode er blevet ændret");
71                         modelAndView.setViewName("redirect:/
login");
72                         user.setPassword(bCryptPasswordEncoder.
encode(password1));
73                         userRepository.save(user);
74                     }else{
75                         modelAndView.addObject("Message", "Du
har ik indtastet den samme kode");
76                     }
77                 }else{
78                     modelAndView.addObject("Message", "Linket
virker ikke, prøv igen.");

```

```

79         }
80     }
81     return modelAndView;
82 }
83
84 @RequestMapping(value = "/forgotpassword", method =
RequestMethod.POST)
85 public ModelAndView forgotPassword(@RequestParam
String email) {
86     ModelAndView modelAndView = new ModelAndView();
87     User user = userService.findUserByEmail(email);
88     if (user == null) {
89         modelAndView.addObject("Message", "Emailen
existere ikke, prøv igen.");
90     }else{
91         modelAndView.addObject("Message", "SUCCES!:
Der er sent et link til din email!");
92         sendResetEmail(user);
93     }
94     return modelAndView;
95 }
96
97 public void sendResetEmail(User user){
98     String body = "<B>Reset dit password</B><br><br>";
99     body += "Hvis du har bedt om at få dit password
reset,<br> beder vi dig ";
100     body += "beder vi dig klikke på nedenstående link
.<br><br>";
101     body += "<a href=\"http://localhost:8080/
forgotpassword/"+user.getPassword().replace("/", "")+"\"+
user.getEmail()+"/newpassword\">Klik her for at reset dit
password</a><br><br>";
102     body += "Har du ikke bedt om et reset skal du bare
ignore denne email.<br><br>";
103     body += "Mvh<br>";
104     body += "Jens Boligforening<br>";
105     body += "nørrebrogade 123<br>";
106     body += "2312 nørrebro";
107     emailService.generateAndSendEmail(user.getEmail(),
"BoligInfo password reset", body);
108 }
109 }
110

```

```
1 package dk.binfo.repositories;
2
3 import dk.binfo.models.Role;
4 import org.springframework.data.jpa.repository.
    JpaRepository;
5 import org.springframework.stereotype.Repository;
6 /**
7  * @author Patrick Klæbel
8  * @author Jens Bäckvall
9  * @author Steen Petersen
10 * @author Morten Olsen
11 *
12 */
13 @Repository("roleRepository")
14 public interface RoleRepository extends JpaRepository<Role,
    Integer>{
15     Role findByRole(String role);
16
17 }
18
```

```
1 package dk.binfo.repositories;
2
3 import dk.binfo.models.User;
4 import org.springframework.data.jpa.repository.
    JpaRepository;
5 import org.springframework.stereotype.Repository;
6 /**
7  * @author Patrick Klæbel
8  * @author Jens Bäckvall
9  * @author Steen Petersen
10 * @author Morten Olsen
11 *
12 */
13 @Repository("userRepository")
14 public interface UserRepository extends JpaRepository<User,
    String> {
15     User findByEmail(String email);
16 }
17
```



```
1 package dk.binfo.repositories;
2
3 import dk.binfo.models.Apartment;
4 import org.springframework.data.jpa.repository.
    JpaRepository;
5 import org.springframework.stereotype.Repository;
6 /**
7  * @author Patrick Klæbel
8  * @author Jens Bäckvall
9  * @author Steen Petersen
10 * @author Morten Olsen
11 *
12 */
13 @Repository("apartmentRepository")
14 public interface ApartmentRepository extends JpaRepository<
    Apartment, Integer> {
15
16 }
```

```

1 package dk.binfo.configurations;
2
3 import org.springframework.beans.factory.annotation.
  Autowired;
4 import org.springframework.context.annotation.Bean;
5 import org.springframework.context.annotation.Configuration
  ;
6 import org.springframework.security.config.annotation.
  authentication.builders.AuthenticationManagerBuilder;
7 import org.springframework.security.config.annotation.web.
  builders.HttpSecurity;
8 import org.springframework.security.config.annotation.web.
  builders.WebSecurity;
9 import org.springframework.security.config.annotation.web.
  configuration.EnableWebSecurity;
10 import org.springframework.security.config.annotation.web.
  configuration.WebSecurityConfigurerAdapter;
11 import org.springframework.security.core.userdetails.
  UserDetailsService;
12 import org.springframework.security.crypto.bcrypt.
  BCryptPasswordEncoder;
13 import org.springframework.security.web.access.
  AccessDeniedHandler;
14 import org.springframework.security.web.util.matcher.
  AntPathRequestMatcher;
15 import org.thymeleaf.extras.springsecurity4.dialect.
  SpringSecurityDialect;
16 import org.thymeleaf.spring4.SpringTemplateEngine;
17 import org.thymeleaf.templateresolver.ITemplateResolver;
18
19 /**
20  * @author Patrick Klæbel
21  * @author Steen Petersen
22  *
23  */
24 @Configuration
25 @EnableWebSecurity
26 public class SecurityConfiguration extends
  WebSecurityConfigurerAdapter {
27
28     @Autowired
29     private BCryptPasswordEncoder bCryptPasswordEncoder;
30
31
32     @Autowired
33     private UserDetailsService userDetailsService;
34
35     @Override
36     protected void configure(AuthenticationManagerBuilder
  auth)

```

```

37         throws Exception {
38         auth
39             .userDetailsService(userDetailsService)
40             .passwordEncoder(bCryptPasswordEncoder);
41     }
42
43     @Override
44     protected void configure(HttpSecurity http) throws
Exception {
45
46         http.authorizeRequests()
47             .antMatchers("/", "/login", "/"
forgotpassword/**", "/registration", "/error").permitAll()
48             .antMatchers("/apartment/**", "/users/**").
hasAuthority("ADMIN")
49             .anyRequest().authenticated().and().csrf().
disable().formLogin()
50             .loginPage("/login").failureUrl("/login?
error=true").permitAll()
51             .defaultSuccessUrl("/home")
52             .usernameParameter("email").
passwordParameter("password")
53             .and().rememberMe().key("rem-me-key").
rememberMeParameter("remember-me").rememberMeCookieName("
remember-me")
54             .and().logout().logoutRequestMatcher(new
AntPathRequestMatcher("/logout")).logoutSuccessUrl("/login"
)
55             .and().exceptionHandling().
accessDeniedHandler(accessDeniedHandler());
56     }
57
58     @Override
59     public void configure(WebSecurity web) throws Exception
{
60         web.ignoring().antMatchers("/resources/**", "/"
static/**", "/fonts/**", "/css/**", "/js/**", "/img/**");
61     }
62
63     @Bean
64     public BCryptPasswordEncoder passwordEncoder() {
65         BCryptPasswordEncoder bCryptPasswordEncoder = new
BCryptPasswordEncoder();
66         return bCryptPasswordEncoder;
67     }
68
69     @Bean
70     public AccessDeniedHandler accessDeniedHandler(){
71         return new CustomAccessDeniedHandler();
72     }

```

73 }

```
1 package dk.binfo.configurations;
2
3 import java.io.IOException;
4
5 import javax.servlet.ServletException;
6 import javax.servlet.http.HttpServletRequest;
7 import javax.servlet.http.HttpServletResponse;
8
9 import org.slf4j.Logger;
10 import org.slf4j.LoggerFactory;
11
12 import org.springframework.security.access.
    AccessDeniedException;
13 import org.springframework.security.core.Authentication;
14 import org.springframework.security.core.context.
    SecurityContextHolder;
15 import org.springframework.security.web.access.
    AccessDeniedHandler;
16
17 /**
18  * Whenever a user attempts to access a page that is
19  * restricted to roles they do not have,
20  * the application will return a status code of 403, which
21  * means Access Denied.<p></p>
22  * Following Roles
23  * <ul>
24  * <li>Admin
25  * <li>User
26  * </ul><p></p>
27  *
28  * By default, Spring Security has an
29  * ExceptionTranslationFilter defined
30  * which handles exceptions of type AuthenticationException
31  * and AccessDeniedException.
32  * The latter is done through a property called
33  * accessDeniedHandler,
34  * which uses the AccessDeniedHandlerImpl class.
35  * <p>
36  * In order to customize this behavior to use our own page
37  * that we created (see <a href="/accessDenied">AccessDenied
38  * page</a>),
39  * we need to override the properties of the
40  * ExceptionTranslationFilter class.
41  * This can be done through either Java configuration or
42  * XML configuration.
43  * <br>
44  * Using Java, I customized the 403 error handling process
45  * by using the accessDeniedHandler()
46  * methods while configuring the HttpSecurity element.
47  * <br>
```

```

38  * Using an access denied handler instead of a page has the
    advantage
39  * that we can define custom logic to be executed before
    redirecting to the 403 page.
40  * For this, we created a class that implements the
    AccessDeniedHandler interface
41  * and overrides the handle() method.
42  *
43  * <p>
44  * More implementations:
45  * <ul>
46  *     <li>Add the logs to the database, with a page for
    admins to view logs</li>
47  *     <li>Add log file if web application running on
    dedicated server</li>
48  * </ul>
49  */
50  public class CustomAccessDeniedHandler implements
    AccessDeniedHandler {
51
52      /**
53       *public final class LoggerFactory
54       * extends Object
55       * <br>The LoggerFactory is a utility class producing
    Loggers for various logging APIs, most notably for log4j,
    logback and JDK 1.4 logging. Other implementations such as
    NOPLogger and SimpleLogger are also supported.
56       * LoggerFactory is essentially a wrapper around an
    ILoggerFactory instance bound with LoggerFactory at compile
    time.<p></p>
57       *
58       * Please note that all methods in LoggerFactory are
    static.
59       * <br>
60       *     Author: Alexander Dorokhine, Robert Elliot, Ceki
    Gülcü
61       */
62      public static final Logger LOG = LoggerFactory.
    getLogger(CustomAccessDeniedHandler.class);
63
64      @Override
65      /**
66       *<br>
67       * The custom logic logs a warning message for every
    access denied attempt
68       * containing the user that made the attempt and the
    protected URL they were trying to access:
69       * <p>
70       * 2017-11-30 10:42:47.236 WARN 5232 --- [nio-8080-
    exec-6] d.b.c.CustomAccessDeniedHandler

```

```

71      * <br>User: user@test.dk[user] attempted to access
    the protected URL: /apartment<br>
72      * This user tried to access /apartment which required
    admin rank.
73      *
74      * @param request Extends the ServletRequest interface
    to provide request information for HTTP servlets.
75      * @param response Extends the ServletResponse
    interface to provide HTTP-specific functionality in
    sending a response. For example, it has methods to access
    HTTP headers and cookies.
76      * @param exc Checked exception thrown when a
    operation is denied, typically due to a file permission or
    other access check.
77      *
78      */
79      public void handle(HttpServletRequest request,
    HttpServletResponse response, AccessDeniedException exc)
    throws IOException, ServletException {
80          Authentication auth = SecurityContextHolder.
    getContext().getAuthentication();
81          if (auth != null) {
82              LOG.warn("User: " + auth.getName() + auth.
    getAuthorities() + " attempted to access the protected URL
    : " + request.getRequestURI());
83          }
84          response.sendRedirect(request.getContextPath() +
    "/accessDenied");
85      }
86
87  }
88

```