```java
package dk.binfo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.
SpringBootApplication;

@SpringBootApplication
public class BinfoApplication {

    public static void main(String[] args) {
        SpringApplication.run(BinfoApplication.class, args)
;
    }
}

```

```java
package dk.binfo.models;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;
/**
 *  @author Patrick Klæbel
 *  @author Jens Bäckvall
 *  @author Steen Petersen
 *  @author Morten Olsen
 *
 */
@Entity
@Table(name = "role")
public class Role {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name="role_id")
    private int id;
    @Column(name="role")
    private String role;

    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getRole() {
        return role;
    }
    public void setRole(String role) {
        this.role = role;
    }


}

```

```java
package dk.binfo.models;

import java.util.Set;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.ManyToMany;
import javax.persistence.Table;

import org.springframework.data.annotation.Transient;
/**
 *  @author Patrick Klæbel
 *  @author Jens Bäckvall
 *  @author Steen Petersen
 *  @author Morten Olsen
 *
 */
@Entity
@Table(name = "user")
public class User {

    @Id
    @Column(name = "email")
    private String email;

    @Column(name = "password")
    @Transient
    private String password;

    @Column(name = "name")
    private String name;

    @Column(name = "last_name")
    private String lastName;

    @Column(name = "active")
    private boolean active;

    @ManyToMany(cascade = CascadeType.ALL)
    @JoinTable(name = "user_role", joinColumns = @
JoinColumn(name = "user_id"), inverseJoinColumns = @
JoinColumn(name = "role_id"))
    private Set<Role> roles;

    @Column(name ="phone_number")
    private String phoneNumber;

    @Column(name = "my_apartment")
    private int myApartment;

    public int getMyApartment() {
        return myApartment;
    }

    public void setMyApartment(int myApartment) {
        this.myApartment = myApartment;
    }

    public String getPhoneNumber() {
        return phoneNumber;
    }

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

    public String getPassword() {
        return password;
    }

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

    public String getName() {
        return name;
    }

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

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public String getEmail() {
        return email;
    }

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

    public boolean isActive() {
        return active;
    }

    public void setActive(boolean active) {
        this.active = active;
    }

    public Set<Role> getRoles() {
        return roles;
    }

    public void setRoles(Set<Role> roles) {
        this.roles = roles;
    }

}

```

```java
package dk.binfo.models;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;
/**
 *  @author Patrick Klæbel
 *  @author Jens Bäckvall
 *  @author Steen Petersen
 *  @author Morten Olsen
 *
 */
@Entity
@Table(name = "apartment")
public class Apartment {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "id")
    private int id;
    @Column(name = "address")
    private String address;
    @Column(name = "number")
    private String number;
    @Column(name = "rooms")
    private Integer rooms;
    @Column(name = "garden")
    private boolean garden;
    @Column(name = "size")
    private Integer size;
    @Column(name = "floor")
    private Integer floor;
    @Column(name = "floors")
    private Integer floors;

    public int getId() {
        return id;
    }

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

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public String getNumber() {
        return number;
    }

    public void setNumber(String number) {
        this.number = number;
    }

    public Integer getRooms() {
        return rooms;
    }

    public void setRooms(Integer rooms) {
        this.rooms = rooms;
    }

    public boolean isGarden() {
        return garden;
    }

    public void setGarden(boolean garden) {
        this.garden = garden;
    }

    public Integer getSize() {
        return size;
    }

    public void setSize(Integer size) {
        this.size = size;
    }

    public Integer getFloor() {
        return floor;
    }

    public void setFloor(Integer floor) {
        this.floor = floor;
    }

    public Integer getFloors() {
        return floors;
    }

    public void setFloors(Integer floors) {
        this.floors = floors;
    }
```

```java
}

```

```java
package dk.binfo.services;

import dk.binfo.models.User;

import java.util.ArrayList;
import java.util.List;
import com.itextpdf.text.*;

/**
 * The Interface for creating waitinglists for displaying on screen
 * or in a PDF
 *
 * @author jensbackvall
 */

public interface ListService {
    void generateSingleApartmentPDF(int listLength, int apartmentNumber, String filePath);
    void generateCompleteListPDF(int listLength, int priority, String filePath);
    List<User> generateList(int length, int priority);
    List<User> generateSingleApartmentList(int length, int ApartmentId);
    void loopThroughEmailList(ArrayList<String> emailList, Document theList, String filePath);
}
```

```java
package dk.binfo.services;

import dk.binfo.models.User;

import java.util.List;
/**
 *  @author Patrick Klæbel
 *  @author Jens Bäckvall
 *  @author Steen Petersen
 *  @author Morten Olsen
 *
 */
public interface UserService {
    User findUserByEmail(String email);
    User updateUserSettings(User user);
    User deleteUser(String email);
    User update(User user);
    List<User> findAll();
    void register(User user);
    void adminRegisterUser(User user);
}

```

```java
package dk.binfo.services;

import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.ArrayList;

import org.codehaus.groovy.runtime.powerassert.SourceText;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Service;

/**
 * A service we can use to make a waitinglist
 * the waitinglist generated can be based on
 * a priority or an apartment
 *
 * The waitinglist length can also be based on a
 * specific length input
 *
 * @author      Stonie
 */
@Service("waitinglist")
public class Waitinglist {

    /**
     * A pointer we can use to refer to springs JdbcTemplate
     * so we can use the same database connection and can avoid
     * creating multiple connections that overlap eachother
     *
     * @author      Stonie
     */
    private final JdbcTemplate jdbcTemplate;

    /**
     * An autowired method that gets the jdbcTemplate
     * from spring so we can use the same database
     * connection and can avoid creating multiple
     * connections that overlap eachother
     *
     * @param       jdbcTemplate The length of the waitinglist to be generated
     * @author      Stonie
     */
    @Autowired
    public Waitinglist(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

    /**
     * The main method, used to get an Arraylist
     * containing sorted results based on both
     * length and ApartmentId.
     * <p>
     * It uses almost all methods in the class to
     * do this in the most optimized way.
     *
     * @param       length The length of the waitinglist to be generated
     * @param       ApartmentId The id of the apartment
     * @return      ArrayList containing emails on the waitinglist
     * @author      Stonie
     */
    public ArrayList<String> getWaitinglist(int length,int ApartmentId){
        if(getPreferences(length,ApartmentId)==null){
            return null;
        }
        return checkPriority(length,getPreferences(length,ApartmentId),ApartmentId);
    }

    /**
     * Returns the emails of neighbours of the ApartmentId
     * using sql to get the apartments around the apartment
     * and then uses sql again to get the emails of said
     * apartments.
     *
     * @param       ApartmentId The id of the apartment
     * @return      ArrayList containing emails of neighbours
     * @author      Stonie
     */
    public ArrayList<String> getNeighbourEmails(int ApartmentId){
        ArrayList<Integer> neighboursID = new ArrayList<Integer>();
        try {
            PreparedStatement sql = jdbcTemplate.getDataSource().getConnection().prepareStatement("SELECT neighbour FROM `apartment_neighbours` WHERE id_apartment=?;");
            sql.setInt(1, ApartmentId);
            ResultSet result = sql.executeQuery();
            while (result.next()) {
                int neighbour = result.getInt("neighbour");
                neighboursID.add(neighbour);
```

```java
      }
      sql.close();
      result.close();
    } catch (Exception e) {
      e.printStackTrace();
    }
    ArrayList<String> emailsunsorted = new ArrayList<String>();
    if (neighboursID.isEmpty()){
      return emailsunsorted;
    }
    for (int neighbourID : neighboursID){
      try {
        PreparedStatement sql = jdbcTemplate.getDataSource().getConnection().prepareStatement("SELECT email FROM `user` WHERE my_apartment=?;");
        sql.setInt(1, neighbourID);
        ResultSet result = sql.executeQuery();
        if (result.next()){
          String email = result.getString("email");
          emailsunsorted.add(email);
        }
        sql.close();
        result.close();
      } catch (Exception e) {
        e.printStackTrace();
      }
    }
    if (emailsunsorted.isEmpty()){
      return emailsunsorted;
    }
    ArrayList<String> emailssorted = new ArrayList<String>();
      try {
        String SQLString = "SELECT email FROM `list_and_seniority` WHERE email=? AND list_priority=1";
        for (int i = 1;i<emailsunsorted.size();i++){
          SQLString += " OR email=? AND list_priority=1";
        }
        String SQLEND = " ORDER BY seniority ASC;";
        SQLString += SQLEND;
        PreparedStatement sql = jdbcTemplate.getDataSource().getConnection().prepareStatement(SQLString);
        for (int i = 0;i<emailsunsorted.size();i++){
```

```java
                    sql.setString(i+1, emailsunsorted.get(i));
                }
                ResultSet result = sql.executeQuery();
                while (result.next()){
                    String email = result.getString("email");
                    emailssorted.add(email);
                }
                sql.close();
                result.close();
                return emailssorted;
            } catch (Exception e){
                    e.printStackTrace();
            }
        return null;
    }

    /**
     * Returns the emails on waitinglist for the ApartmentId
     * using sql to get them.
     * The list will not be sorted.
     * <p>
     * It will NOT contain the emails from the neighbour
     * waitinglist, use the getNeighbourEmails method for that
     *
     * @param       length The length of the waitinglist to be generated
     * @param       ApartmentId The id of the apartment
     * @return      ArrayList containing emails on the waitinglist
     * @author      Stonie
     */
    public ArrayList<String> getPreferences(int length,int ApartmentId){
        ArrayList<String> pref = new ArrayList<String>();
        try {
                PreparedStatement sql = jdbcTemplate.getDataSource().getConnection()
                    .prepareStatement("SELECT email FROM `user_preferences` WHERE id_apartment=?;");
                sql.setInt(1, ApartmentId);
                ResultSet result = sql.executeQuery();
                if(!result.next()){
                    sql.close();
                    result.close();
                    return null;
                }
```

```java

                    do {
                        String email = result.getString("email");
                        pref.add(email);
                        if (pref.size() >= length){
                            sql.close();
                            result.close();
                            return pref;
                        }
                    } while (result.next());

                    sql.close();
                    result.close();
                    return pref;
            } catch (Exception e) {
                e.printStackTrace();
            }

            return null;
    }

    /**
     * Sorts the emails based on seniority
     * and returns the sorted list as an ArrayList
     *
     * @param       length The length of the waitinglist to be generated
     * @param       emails An arraylist containing the emails to be sorted
     * @param       ApartmentId The id of the apartment
     * @return      ArrayList containing emails sorted by seniority
     * @author      Stonie
     */
    public ArrayList<String> checkPriority(int length, ArrayList<String> emails, int ApartmentId){


                    ArrayList<String> emailssorted = getNeighbourEmails(ApartmentId);
                    try {
                        String SQLString = "SELECT email FROM `list_and_seniority` WHERE email=? AND list_priority=2";
                        for (int i = 1;i<emails.size();i++){
                            SQLString += " OR email=? AND list_priority=2";
                        }
                        String SQLEND = " ORDER BY seniority ASC;";
```

```java
          SQLString += SQLEND;
          PreparedStatement sql = jdbcTemplate.getDataSource().getConnection().prepareStatement(SQLString);
          for (int i = 0;i<emails.size();i++){
            sql.setString((i+1), emails.get(i));
          }
          ResultSet result = sql.executeQuery();
          while (result.next()){
            String email = result.getString("email");
            emailssorted.add(email);
          }
          sql.close();
          result.close();
        } catch (Exception e){
          e.printStackTrace();
        }
    try {
      String SQLString = "SELECT email FROM `list_and_seniority` WHERE email=? AND list_priority=3";
      for (int i = 1;i<emails.size();i++){
        SQLString += " OR email=? AND list_priority=3";
      }
      String SQLEND = " ORDER BY seniority ASC;";
      SQLString += SQLEND;
      PreparedStatement sql = jdbcTemplate.getDataSource().getConnection().prepareStatement(SQLString);
      for (int i = 0;i<emails.size();i++){
        sql.setString((i+1), emails.get(i));
      }
      ResultSet result = sql.executeQuery();
      while (result.next()){
        String email = result.getString("email");
        emailssorted.add(email);
      }
      sql.close();
      result.close();
    } catch (Exception e){
      e.printStackTrace();
    }
    try {
      String SQLString = "SELECT email FROM `list_and_seniority` WHERE email=? AND list_priority=4";
      for (int i = 1;i<emails.size();i++){
        SQLString += " OR email=? AND list_priority=4";
```

```java
            }
            String SQLEND = " ORDER BY seniority ASC;";
            SQLString += SQLEND;
            PreparedStatement sql = jdbcTemplate.
getDataSource().getConnection().prepareStatement(SQLString
);
            for (int i = 0;i<emails.size();i++){
                sql.setString((i+1), emails.get(i));
            }
            ResultSet result = sql.executeQuery();
            while (result.next()){
                String email = result.getString("email");
                emailssorted.add(email);
            }
            sql.close();
            result.close();
        } catch (Exception e){
            e.printStackTrace();
        }
        int size = emailssorted.size();
        if (length<size){
            emailssorted.subList(length, size).clear();
        }
        return emailssorted;
    }

    /**
     * Returns an ArrayList containing all the emails
     * on a specific waitinglist based on priority and
     * sorted by seniority.
     *
     * @param       length The length of the waitinglist
to be generated
     * @param       priority the priority of the list to
be generated
     * @return      ArrayList containing emails on the
priority list sorted by seniority
     * @author      Stonie
     */
    public ArrayList<String> getSingleWaitinglist(int
length,int priority){
        if (priority>4||priority<1){
            return null;
        }
        ArrayList<String> emailssorted = new ArrayList<
String>();
        try {
            PreparedStatement sql = jdbcTemplate.
getDataSource().getConnection().prepareStatement("SELECT
email FROM `list_and_seniority` WHERE list_priority="+
priority+" ORDER BY seniority ASC;");
            ResultSet result = sql.executeQuery();
            while (result.next()){
                String email = result.getString("email");
                emailssorted.add(email);
            }
            sql.close();
            result.close();
            return emailssorted;
        } catch (Exception e){
                e.printStackTrace();
        }
        return emailssorted;
    }
}

```

```java
package dk.binfo.services;

import org.springframework.stereotype.Service;

import java.util.Properties;
import javax.mail.Message;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;

/**
 * A class with methods to handle emails
 * so far it can only handle gmail.
 *
 * @author Stonie
 */
@Service("emailService")
public class EmailService {

    /**
     * A simple method to send emails using Gmail
     * using param user and password to login
     *
     * @param To The email we should send to
     * @param Subject The subject of the email
     * @param Body The body of the email, you can use html tags like <br>
     * @author Stonie
     */
    public void generateAndSendEmail(String To,String Subject,String Body){
    try {
        Properties mailServerProperties;
        Session getMailSession;
        MimeMessage generateMailMessage;
        // setup Mail Server Properties
        mailServerProperties = System.getProperties();
        mailServerProperties.put("mail.smtp.port", "587");
        mailServerProperties.put("mail.smtp.auth", "true");
        mailServerProperties.put("mail.smtp.starttls.enable", "true");
        mailServerProperties.put("mail.smtp.ssl.trust", "smtp.gmail.com");

        // get Mail Session
        getMailSession = Session.getDefaultInstance(mailServerProperties, null);
        generateMailMessage = new MimeMessage(getMailSession);
        generateMailMessage.addRecipient(Message.RecipientType.TO, new InternetAddress(To));
        generateMailMessage.setSubject(Subject);
        String emailBody = Body;
        generateMailMessage.setContent(emailBody, "text/html; charset=UTF-8");

        // Get Session and Send mail
        Transport transport = getMailSession.getTransport("smtp");

        // Enter your correct gmail UserID and Password
        transport.connect("smtp.gmail.com", "testbinfo1234", "testbinfo1234testbinfo1234");
        transport.sendMessage(generateMailMessage, generateMailMessage.getAllRecipients());
        transport.close();
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}

```

```java
package dk.binfo.services;

import dk.binfo.models.Apartment;
import dk.binfo.repositories.ApartmentRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;
/**
 *  @author Patrick Klæbel
 *  @author Jens Bäckvall
 *  @author Steen Petersen
 *  @author Morten Olsen
 *
 */
@Service("apartmentService")
public class ApartmentImpl implements ApartmentService {

    @Autowired
    private ApartmentRepository apartmentRepository;

    @Override
    public void saveApartment(Apartment apartment) {
        apartmentRepository.save(apartment);
    }

    @Override
    @Transactional
    public Apartment findById(int id) {
        return apartmentRepository.findOne(id);
    }

    @Override
    @Transactional
    public Apartment delete(int id) {
        Apartment deletedApartment = apartmentRepository.findOne(id);

        apartmentRepository.delete(deletedApartment);
        return deletedApartment;
    }

    @Override
    @Transactional
    public List<Apartment> findAll() {
        return apartmentRepository.findAll();
    }

    @Override
    @Transactional()
    public Apartment update(Apartment apartment){
        Apartment updateApartment = apartmentRepository.
findOne(apartment.getId());

        updateApartment.setAddress(apartment.getAddress());
        updateApartment.setNumber(apartment.getNumber());
        updateApartment.setRooms(apartment.getRooms());
        updateApartment.setGarden(apartment.isGarden());
        updateApartment.setSize(apartment.getSize());
        updateApartment.setFloor(apartment.getFloor());
        updateApartment.setFloors(apartment.getFloors());
        return updateApartment;
    }
}

```

```java
package dk.binfo.services;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.File;
import java.util.ArrayList;
import java.util.List;

import com.itextpdf.text.*;
import com.itextpdf.text.pdf.PdfWriter;
import dk.binfo.models.User;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import static java.lang.Integer.MAX_VALUE;


/**
 * A service used for generating waitinglists for displaying on screen
 * in a table or for generating a PDF of a list for a specific
 * apartment.
 *
 * The Waitinglist service is used, and just as we can do it in that
 * service, we use length, i.e. number of users in list and priority,
 * i.e. which list we want to display, as parameters.
 *
 * The priorities are:
 * 1. Connect list, for connecting apartments into larger apartments
 * 2. Internal list, for users who already live in the building
 * 3. Family list, for users who are related to Internal users
 * 4. External list, for users who have no relation to the building.
 *
 * @author jensbackvall
 */

@Service("listService")
public class ListServiceImpl implements ListService {

    @Autowired
    private UserService userService;

    @Autowired
    private Waitinglist waitinglist;

    private Font theFont = new Font(Font.FontFamily.TIMES_ROMAN, 16, Font.BOLD);
    private Font theSmallFont = new Font(Font.FontFamily.TIMES_ROMAN, 12, Font.BOLD);

    /**
     * The generatePDF method generates a PDF using the itext API
     * for creating a pdf. It also uses getWaitinglist from the
     * waitinglist service, and therefore takes in the same parameters:
     *
     * @param listLength
     * @param apartmentNumber
     *
     */

    @Override
    public void generateSingleApartmentPDF(int listLength, int apartmentNumber, String filePath) {

        Document theList = new Document();

        ArrayList <String> emailList = waitinglist.getWaitinglist(listLength, apartmentNumber);

        loopThroughEmailList(emailList, theList, filePath);
    }

    @Override
    public void generateCompleteListPDF(int listLength, int priority, String filePath) {

        Document theList = new Document();

        ArrayList <String> emailList = waitinglist.getSingleWaitinglist(listLength, priority);

        loopThroughEmailList(emailList, theList, filePath);
    }


    /**
     * The generateList method uses the getSingleWaitinglist method
     * from the waitinglist service to generate a waiting
```

```
list for a
     * certain priority, as described above. It takes in
the
     * parameters needed, being the length, or number of
users, in
     * the list as well as the priority or type of waiting
 list.
     *
     * It returns a list of the desired type, for
displaying in a
     * table on screen.
     *
     * @param length
     * @param priority
     * @return
     */

    public List<User> generateList(int length, int
priority) {
        List<User> generatedList = new ArrayList<>();
        ArrayList<String> emailList = waitinglist.
getSingleWaitinglist(Integer.MAX_VALUE, priority);
        for (String email: emailList) {
            User listUser = userService.findUserByEmail(
email);
            generatedList.add(listUser);
        }
        return generatedList;
    }

    /**
     * The generateSingleApartmentList method uses the
getWaitinglist method
     * from the waitinglist service to generate a waiting
list for a
     * certain apartment. It takes in the
     * parameters needed, being the length, or number of
users, in
     * the list as well as the id of the apartment that is
 being sold.
     *
     * It returns a list of the desired type, for
displaying in a
     * table on screen.
     *
     * @param length
     * @param ApartmentId
     * @return
     */

```

```java
    public List<User> generateSingleApartmentList(int length, int ApartmentId) {
        List<User> generatedApartmentList = new ArrayList<>();
        ArrayList<String> emailList = waitinglist.getWaitinglist(Integer.MAX_VALUE, ApartmentId);
        if (emailList == null) {
            return null;
        }
        for (String email: emailList) {
            User listUser = userService.findUserByEmail(email);
            generatedApartmentList.add(listUser);
        }
        return generatedApartmentList;
    }

    @Override
    public void loopThroughEmailList(ArrayList <String> emailList, Document theList, String filePath) {
        try {
            PdfWriter.getInstance(theList, new FileOutputStream(new File(filePath))); // filePath ?!?!?!?!?!?!?!

            theList.open();

            Header h = new Header("listHeader","New Apartment List"); // evt dags dato + listetype + lejlighed, eks. "010217_intern_ for lejlighed_23"

            theList.add(h);

            for (String email: emailList) {
                User listUser = userService.findUserByEmail(email);
                Paragraph p = new Paragraph();
                Chunk seniority = new Chunk("\nAncienittet: " + (emailList.indexOf(email) + 1) + "\n", theFont);
                Chunk name = new Chunk("Navn: " + listUser.getName() + " " + listUser.getLastName() + "\n", theSmallFont);
                Chunk phoneNumber = new Chunk("Telefonnummer: " + listUser.getPhoneNumber() + "\n", theSmallFont);
                Chunk user_email = new Chunk("E-mail: " + listUser.getEmail() + "\n", theSmallFont);
                p.add(seniority);
                p.add(name);
                p.add(phoneNumber);
                p.add(user_email);
                p.setAlignment(Element.ALIGN_CENTER);
                theList.add(p);
                Paragraph p2 = new Paragraph();
                p2.add("___________________________________________________");
                p2.setAlignment(Element.ALIGN_CENTER);
                theList.add(p2);
            }

            theList.close();

        } catch (FileNotFoundException | DocumentException e) {
            e.printStackTrace();
        }
    }

}



```

```java
package dk.binfo.services;

import java.util.*;

import javax.transaction.Transactional;

import dk.binfo.models.Role;
import dk.binfo.models.User;
import dk.binfo.repositories.RoleRepository;
import dk.binfo.repositories.UserRepository;
import org.springframework.beans.factory.annotation.
Autowired;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.
SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.
UserDetails;
import org.springframework.security.core.userdetails.
UserDetailsService;
import org.springframework.security.core.userdetails.
UsernameNotFoundException;
import org.springframework.security.crypto.bcrypt.
BCryptPasswordEncoder;
import org.springframework.stereotype.Service;

/**
 *  @author Patrick Klæbel
 *  @author Jens Bäckvall
 *  @author Steen Petersen
 *  @author Morten Olsen
 *
 */
@Service("userService")
public class UserServiceImpl implements UserService,
UserDetailsService {

    @Autowired
    private UserRepository userRepository;
    @Autowired
    private RoleRepository roleRepository;
    @Autowired
    private BCryptPasswordEncoder bCryptPasswordEncoder;

    @Override
    public User findUserByEmail(String email) {

        return userRepository.findByEmail(email);
    }


    @Override
    @Transactional
    public List<User> findAll() {
        return userRepository.findAll();
    }

    @Override
    public void register(User user) {
        user.setPassword(bCryptPasswordEncoder.encode(user.getPassword()));
        user.setActive(true);
        user.setPhoneNumber(user.getPhoneNumber());
        Role userRole = roleRepository.findByRole("user");
        user.setRoles(new HashSet<Role>(Arrays.asList(userRole)));
        userRepository.save(user);
    }

    @Override
    @Transactional
    public User deleteUser(String email) {
        User deletedUser = userRepository.findByEmail(email);
        deletedUser.setRoles(null); // Deletes this user in user_role table
        userRepository.delete(deletedUser);
        return deletedUser;
    }

    @Override
    @Transactional
    public User update(User user){
        User updatedUser = userRepository.findByEmail(user.getEmail());
        updatedUser.setName(user.getName());
        updatedUser.setLastName(user.getLastName());
        updatedUser.setPhoneNumber(user.getPhoneNumber());
        updatedUser.setActive(user.isActive());

        return updatedUser;
    }

    @Override
    public void adminRegisterUser(User user) {
        Random ran = new Random();
        String Password = generateString(ran,"qwertyuiopasdfghjklzxcvbnm",8);
        user.setPassword(bCryptPasswordEncoder.encode(Password));
        user.setActive(user.isActive());
        user.setPhoneNumber(user.getPhoneNumber());
        Role userRole = roleRepository.findByRole("user");
        user.setRoles(new HashSet<Role>(Arrays.asList(userRole)));
        userRepository.save(user);
    }

    @Override
    @Transactional
    public User updateUserSettings(User user){
        User updateUser = userRepository.findByEmail(user.getEmail());
        if(user.getPassword().equalsIgnoreCase("") && user.getPhoneNumber() != null) {
            updateUser.setPhoneNumber(user.getPhoneNumber());
        }
        if(user.getPhoneNumber() == null || user.getPhoneNumber().equalsIgnoreCase("") && user.getPassword() != null) {
            updateUser.setPassword(bCryptPasswordEncoder.encode(user.getPassword()));
        }
        if(user.getPhoneNumber() != null && user.getPassword() != null) {
            updateUser.setPassword(bCryptPasswordEncoder.encode(user.getPassword()));
            updateUser.setPhoneNumber(user.getPhoneNumber());
        }
        return updateUser;
    }

    @Override
    @Transactional
    public UserDetails loadUserByUsername(String userName) throws UsernameNotFoundException {
        User user = userRepository.findByEmail(userName);
        List<GrantedAuthority> authorities = getUserAuthority(user.getRoles());
        return buildUserForAuthentication(user, authorities);
    }

    private List<GrantedAuthority> getUserAuthority(Set<Role> userRoles) {
        Set<GrantedAuthority> roles = new HashSet<GrantedAuthority>();
        for (Role role : userRoles) {
            roles.add(new SimpleGrantedAuthority(role.
getRole()));
        }

        List<GrantedAuthority> grantedAuthorities = new ArrayList<GrantedAuthority>(roles);
        return grantedAuthorities;
    }

    private UserDetails buildUserForAuthentication(User user, List<GrantedAuthority> authorities) {
        return new org.springframework.security.core.userdetails.User(user.getEmail(), user.getPassword(), user.isActive(), true, true, true, authorities);
    }


    public static String generateString(Random rng, String characters, int length)
    {
        char[] text = new char[length];
        for (int i = 0; i < length; i++)
        {
            text[i] = characters.charAt(rng.nextInt(characters.length()));
        }
        return new String(text);
    }
}

```

```java
package dk.binfo.services;

import dk.binfo.models.Apartment;

import java.util.List;
/**
 *  @author Patrick Klæbel
 *  @author Jens Bäckvall
 *  @author Steen Petersen
 *  @author Morten Olsen
 *
 */
public interface ApartmentService {
    Apartment delete(int id);
    List<Apartment> findAll();
    Apartment update(Apartment apartment);
    Apartment findById(int id);

    //Saves the Apartment setters.
    void saveApartment(Apartment apartment);
}

```

```java
package dk.binfo.controllers;

import dk.binfo.models.User;
import dk.binfo.services.ListService;
import dk.binfo.services.UserService;
import org.springframework.beans.factory.annotation.
Autowired;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.
SecurityContextHolder;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.PathVariable
;
import org.springframework.web.bind.annotation.
RequestMapping;
import org.springframework.web.bind.annotation.
RequestMethod;
import org.springframework.web.servlet.ModelAndView;

import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.File;
import java.io.IOException;

/**
 *  @author Jens Bäckvall
 *  @author Steen Petersen
 */
@Controller
public class PDFController {

    @Autowired
    private UserService userService;

    @Autowired
    private ListService listService;

    @RequestMapping(value={"/lists/connect/pdf"})
    public ModelAndView generateConnectListPDF() {
        ModelAndView modelAndView = new ModelAndView("/
lists/connect", "list", listService
                .generateList(Integer.MAX_VALUE, 1));
        String filePath = "/Users/jensbackvall/Desktop/
PDF_TEST/Sammenlaegningsliste.pdf"; // filePath
        listService.generateCompleteListPDF(Integer.
MAX_VALUE, 1, filePath);
        Authentication auth = SecurityContextHolder.
getContext().getAuthentication();
        User user = userService.findUserByEmail(auth.
getName());
        modelAndView.addObject("adminMessage","Du er logget
 ind som spadmin");
        modelAndView.addObject("PDFMessage","PDF er gemt på
 /Users/jensbackvall/Desktop/PDF_TEST/");
        modelAndView.addObject(user);
        return modelAndView;
    }

    @RequestMapping(value={"/lists/internal/pdf"})
    public ModelAndView generateInternListPDF() {
        ModelAndView modelAndView = new ModelAndView("/
lists/internal", "list", listService
                .generateList(Integer.MAX_VALUE, 2));
        String filePath = "/Users/jensbackvall/Desktop/
PDF_TEST/Intern_liste.pdf"; // filePath
        listService.generateCompleteListPDF(Integer.
MAX_VALUE, 2, filePath);
        Authentication auth = SecurityContextHolder.
getContext().getAuthentication();
        User user = userService.findUserByEmail(auth.
getName());
        modelAndView.addObject("adminMessage","Du er logget
 ind som spadmin");
        modelAndView.addObject("PDFMessage","PDF er gemt på
 /Users/jensbackvall/Desktop/PDF_TEST/");
        modelAndView.addObject(user);
        return modelAndView;
    }

    @RequestMapping(value={"/lists/family/pdf"})
    public ModelAndView generateFamilyListPDF() {
        ModelAndView modelAndView = new ModelAndView("/
lists/family", "list", listService
                .generateList(Integer.MAX_VALUE, 3));
        String filePath = "/Users/jensbackvall/Desktop/
PDF_TEST/Familieliste.pdf"; // filePath
        listService.generateCompleteListPDF(Integer.
MAX_VALUE, 3, filePath);
        Authentication auth = SecurityContextHolder.
getContext().getAuthentication();
        User user = userService.findUserByEmail(auth.
getName());
        modelAndView.addObject("adminMessage","Du er logget
 ind som spadmin");
        modelAndView.addObject("PDFMessage","PDF er gemt på
 /Users/jensbackvall/Desktop/PDF_TEST/");
        modelAndView.addObject(user);
        return modelAndView;
    }
```

```java

    @RequestMapping(value={"/lists/external/pdf"})
    public ModelAndView generateExternalListPDF() {
        ModelAndView modelAndView = new ModelAndView("/lists/external", "list", listService.generateList(Integer.MAX_VALUE, 4));
        String filePath = "/Users/jensbackvall/Desktop/PDF_TEST/Ekstern_liste.pdf"; // filePath
        listService.generateCompleteListPDF(Integer.MAX_VALUE, 4, filePath);
        Authentication auth = SecurityContextHolder.getContext().getAuthentication();
        User user = userService.findUserByEmail(auth.getName());
        modelAndView.addObject("adminMessage","Du er logget ind som spadmin");
        modelAndView.addObject("PDFMessage","PDF er gemt på /Users/jensbackvall/Desktop/PDF_TEST/");
        modelAndView.addObject(user);
        return modelAndView;
    }

    @RequestMapping(value={"/lists/listapartmentPDF/{id}"}, method = RequestMethod.GET)
    public ModelAndView showSingleApartmentList(@PathVariable("id") Integer id) {
        ModelAndView modelAndView = new ModelAndView("/lists/listapartment", "list", listService.generateSingleApartmentList(Integer.MAX_VALUE, id));
        String filePath = "/Users/jensbackvall/Desktop/PDF_TEST/Liste_for_lejlighed_" + id + ".pdf"; // filePath
        listService.generateSingleApartmentPDF(Integer.MAX_VALUE, id, filePath);
        Authentication auth = SecurityContextHolder.getContext().getAuthentication();
        User user = userService.findUserByEmail(auth.getName());
        modelAndView.addObject("adminMessage","Du er logget ind som spadmin");
        modelAndView.addObject("PDFMessage","PDF er gemt på /Users/jensbackvall/Desktop/PDF_TEST/");
        modelAndView.addObject("HeaderMessage","Viser Liste for Lejlighed nr. " + id);
        modelAndView.addObject(user);
        return modelAndView;
    }
}

```

```java
package dk.binfo.controllers;

import dk.binfo.models.User;
import dk.binfo.services.UserService;
import org.springframework.beans.factory.annotation.
Autowired;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.
SecurityContextHolder;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.
RequestMapping;
import org.springframework.web.bind.annotation.
RequestMethod;
import org.springframework.web.servlet.ModelAndView;

/**
 *  @author Morten Olsen
 */
@Controller
public class HomeController {

    @Autowired
    private UserService userService;

    @RequestMapping(value= "/home", method = RequestMethod.
GET)
    public ModelAndView userHome(){
        ModelAndView modelAndView = new ModelAndView();
        Authentication auth = SecurityContextHolder.
getContext().getAuthentication();
        User user = userService.findUserByEmail(auth.
getName());
        modelAndView.addObject("user", user);
        modelAndView.addObject("userMessage","Du er logget
ind");
        modelAndView.setViewName("/home");
        return modelAndView;
    }
}

```

```java
package dk.binfo.controllers;

import dk.binfo.models.User;
import dk.binfo.services.ListService;
import dk.binfo.services.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;


/**
 *  @author Patrick Klæbel
 *  @author Jens Bäckvall
 *  @author Steen Petersen
 *  @author Morten Olsen
 *
 */
@Controller
public class ListController {

    @Autowired
    private UserService userService;

    @Autowired
    private ListService listService;


    @RequestMapping(value={"/lists/connect"})
    public ModelAndView showConnectList() {
        ModelAndView modelAndView = new ModelAndView("/lists/connect", "list", listService
                .generateList(Integer.MAX_VALUE, 1));
        Authentication auth = SecurityContextHolder.getContext().getAuthentication();
        User user = userService.findUserByEmail(auth.getName());
        modelAndView.addObject(user);
        modelAndView.addObject("adminMessage","Du er logget ind som spadmin");
        return modelAndView;
    }


    @RequestMapping(value={"/lists/internal"})
    public ModelAndView showInternList() {
        ModelAndView modelAndView = new ModelAndView("/lists/internal", "list", listService
                .generateList(20, 2));
        Authentication auth = SecurityContextHolder.getContext().getAuthentication();
        User user = userService.findUserByEmail(auth.getName());
        modelAndView.addObject(user);
        modelAndView.addObject("adminMessage","Du er logget ind som spadmin");
        return modelAndView;
    }

    @RequestMapping(value={"/lists/family"})
    public ModelAndView showFamilyList() {
        ModelAndView modelAndView = new ModelAndView("/lists/family", "list", listService
                .generateList(Integer.MAX_VALUE, 3));
        Authentication auth = SecurityContextHolder.getContext().getAuthentication();
        User user = userService.findUserByEmail(auth.getName());
        modelAndView.addObject(user);
        modelAndView.addObject("adminMessage","Du er logget ind som spadmin");
        return modelAndView;
    }

    @RequestMapping(value={"/lists/external"})
    public ModelAndView showExternalList() {
        ModelAndView modelAndView = new ModelAndView("/lists/external", "list", listService.generateList(Integer.MAX_VALUE, 4));
        Authentication auth = SecurityContextHolder.getContext().getAuthentication();
        User user = userService.findUserByEmail(auth.getName());
        modelAndView.addObject(user);
        modelAndView.addObject("adminMessage","Du er logget ind som spadmin");
        return modelAndView;
    }

    @RequestMapping(value={"/lists/listapartment/{id}"}, method = RequestMethod.GET)
    public ModelAndView showSingleApartmentList(@PathVariable("id") Integer id) {
        ModelAndView modelAndView = new ModelAndView("/lists/listapartment", "list", listService.generateSingleApartmentList(Integer.MAX_VALUE, id));
        Authentication auth = SecurityContextHolder.getContext().getAuthentication();
        User user = userService.findUserByEmail(auth.getName());
        modelAndView.addObject(user);
        modelAndView.addObject("adminMessage","Du er logget ind som admin");
        modelAndView.addObject("HeaderMessage","Viser Liste for Lejlighed nr. " + id);
        modelAndView.setViewName("/lists/listapartment");
        return modelAndView;
    }

}

```

```java
package dk.binfo.controllers;


import dk.binfo.models.User;
import dk.binfo.services.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Controller;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;

import javax.validation.Valid;
/**
 *  @author Jens Bäckvall
 *  @author Morten Olsen
 */
@Controller
public class UserController {

    @Autowired
    private UserService userService;

    @RequestMapping(value={"/settings/{email:.+}"}, method = RequestMethod.GET)
    public ModelAndView settings(@PathVariable String email){
        ModelAndView modelAndView = new ModelAndView();
        Authentication auth = SecurityContextHolder.getContext().getAuthentication();
        User user = userService.findUserByEmail(auth.getName());
        modelAndView.addObject("user", user);
        modelAndView.addObject("userForm",user);
        modelAndView.setViewName("/settings");
        return modelAndView;
    }

    @RequestMapping(value="/settings/{email:.+}", method=RequestMethod.POST)
    public ModelAndView editPhoneSettings(@ModelAttribute @Valid User userinfo, BindingResult bindingResult, @PathVariable String email){
        ModelAndView modelAndView = new ModelAndView();
        Authentication auth = SecurityContextHolder.getContext().getAuthentication();
        User user = userService.findUserByEmail(auth.getName());
        modelAndView.addObject("user", user);

        if (bindingResult.hasErrors())
        {
            modelAndView.setViewName("/settings/{email}");
        }
        modelAndView.setViewName("redirect:/settings/{email}");
        userService.updateUserSettings(userinfo);
        return modelAndView;
    }
}

```

```java
package dk.binfo.controllers;

import dk.binfo.models.User;
import dk.binfo.services.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Controller;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;

import javax.validation.Valid;
/**
 *  @author Patrick Klæbel
 *  @author Jens Bäckvall
 */
@Controller
public class LoginController {

    @Autowired
    private UserService userService;

    @RequestMapping(value={"/login", "/"}, method = RequestMethod.GET)
    public ModelAndView login(){
        ModelAndView modelAndView = new ModelAndView();
        modelAndView.setViewName("login");
        return modelAndView;
    }
}

```

```java
package dk.binfo.controllers;

import dk.binfo.models.User;
import dk.binfo.services.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Controller;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;

import javax.validation.Valid;
/**
 *  @author Patrick Klæbel
 *  @author Jens Bäckvall
 *  @author Steen Petersen
 *  @author Morten Olsen
 *
 */
@Controller
public class UsersController {

    @Autowired
    private UserService userService;

    @RequestMapping("/users")
    public ModelAndView showUsers() {
        ModelAndView modelAndView = new ModelAndView("/users", "users", userService.findAll());
        Authentication auth = SecurityContextHolder.getContext().getAuthentication();
        User user = userService.findUserByEmail(auth.getName());
        modelAndView.addObject("user", user);
        modelAndView.setViewName("/users");
        return modelAndView;
    }

    @RequestMapping("/users/add")
    public ModelAndView adminCreateNewUser(){
        ModelAndView modelAndView = new ModelAndView();
        Authentication auth = SecurityContextHolder.getContext().getAuthentication();
        User user = userService.findUserByEmail(auth.getName());
        modelAndView.addObject("user", user);
        modelAndView.addObject("users", new User());
        modelAndView.setViewName("/users/add");
        return modelAndView;
    }

    @RequestMapping(value = "/users/add", method = RequestMethod.POST)
    public ModelAndView adminCreateUser(@Valid User users, BindingResult bindingResult) {
        ModelAndView modelAndView = new ModelAndView();
        Authentication auth = SecurityContextHolder.getContext().getAuthentication();
        User user = userService.findUserByEmail(auth.getName());
        modelAndView.addObject("user", user);
        User userExists = userService.findUserByEmail(users.getEmail());
        if (userExists != null) {
            bindingResult.rejectValue("email", "error.user", "Der eksisterer allerede en bruger med den angivne email");
        }
        else {
            userService.adminRegisterUser(users);
            modelAndView.addObject("successMessage", "SUCCES!: Du har tilføjet en ny bruger.");
            modelAndView.addObject("users", new User());
            modelAndView.setViewName("/users/add");
        }
        return modelAndView;
    }

    @RequestMapping(value="/users/delete/{email:.+}", method=RequestMethod.GET)
    public ModelAndView deleteApartment(@PathVariable String email) {
        ModelAndView modelAndView = new ModelAndView("redirect:/users");
        userService.deleteUser(email);
        return modelAndView;
    }

    @RequestMapping(value="/users/edit/{email:.+}", method=RequestMethod.GET)
    public ModelAndView editUserPage(@PathVariable String email) {
        ModelAndView modelAndView = new ModelAndView("/users/edit");
        Authentication auth = SecurityContextHolder.getContext().getAuthentication();
        User user = userService.findUserByEmail(auth.getName());
        User editedUser = userService.findUserByEmail(email);
        modelAndView.addObject("user", user);
        modelAndView.addObject("users", editedUser);
        return modelAndView;
    }

    @RequestMapping(value="/users/edit/{email:.+}", method=RequestMethod.POST)
    public ModelAndView editApartment(@ModelAttribute @Valid User editedUser, BindingResult bindingResult){
        ModelAndView modelAndView = new ModelAndView();
        Authentication auth = SecurityContextHolder.getContext().getAuthentication();
        User user = userService.findUserByEmail(auth.getName());
        modelAndView.addObject("user", user);
        if (bindingResult.hasErrors())
        {
            modelAndView.setViewName("/users/edit");
        }
        modelAndView.setViewName("redirect:/users");
        userService.update(editedUser);
        return modelAndView;
    }

}


```

```java
package dk.binfo.controllers;

import dk.binfo.models.User;
import dk.binfo.services.UserService;
import org.springframework.beans.factory.annotation.
Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.
RequestMapping;
import org.springframework.web.bind.annotation.
RequestMethod;
import org.springframework.web.servlet.ModelAndView;

import javax.validation.Valid;
/**
 *  @author Jens Bäckvall
 *  @author Morten Olsen
 */
@Controller
public class RegisterController {

    @Autowired
    private UserService userService;

    @RequestMapping(value={"/registration", "/register"},
method = RequestMethod.GET)
    public ModelAndView registration(){
        ModelAndView modelAndView = new ModelAndView();
        modelAndView.addObject("user",  new User());
        modelAndView.setViewName("registration");
        return modelAndView;
    }

    @RequestMapping(value = "/registration", method =
RequestMethod.POST)
    public ModelAndView createNewUser(@Valid User user,
BindingResult bindingResult) {
        ModelAndView modelAndView = new ModelAndView();
        User userExists = userService.findUserByEmail(user.
getEmail());
        if (userExists != null) {
            bindingResult.rejectValue("email", "error.user"
, "Der eksisterer allerede en bruger med den angivne email"
);
        }
        if (bindingResult.hasErrors()) {
            modelAndView.setViewName("registration");
        } else {
            userService.register(user);
            modelAndView.addObject("successMessage", "
SUCCES!: Du har tilføjet en ny bruger.");
            modelAndView.addObject("user", new User());
            modelAndView.setViewName("registration");
        }
        return modelAndView;
    }
}

```

```java
package dk.binfo.controllers;

import dk.binfo.models.Apartment;
import dk.binfo.repositories.ApartmentRepository;
import dk.binfo.services.ApartmentService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;
import dk.binfo.models.User;
import dk.binfo.services.UserService;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;

import javax.validation.Valid;

@Controller
public class ApartmentController {

    /**
     *  @author Patrick Klæbel
     *  @author Steen Petersen
     *  @author Morten Olsen
     */

    @Autowired
    private ApartmentRepository repository;

    @Autowired
    private UserService userService;

    @Autowired
    private ApartmentService apartmentService;

    @RequestMapping("/apartment")
    public ModelAndView showApartment() {
        ModelAndView modelAndView = new ModelAndView("/apartment", "apartment", repository.findAll());
        Authentication auth = SecurityContextHolder.getContext().getAuthentication();
        User user = userService.findUserByEmail(auth.getName());
        modelAndView.addObject("user", user);
        modelAndView.addObject("adminMessage","Du er logget ind som spadmin");
        modelAndView.setViewName("/apartment");

        return modelAndView;
    }

    @RequestMapping(value = "/apartment/add", method = RequestMethod.POST)
    public ModelAndView createNewApartment(@Valid Apartment apartment, BindingResult bindingResult) {
        ModelAndView modelAndView = new ModelAndView();
        Authentication auth = SecurityContextHolder.getContext().getAuthentication();
        User user = userService.findUserByEmail(auth.getName());
        if (bindingResult.hasErrors()) {
            modelAndView.setViewName("/apartment/add");
        } else {
            apartmentService.saveApartment(apartment);
            modelAndView.addObject("successMessage", "Fantastisk arbejde! Du har nu tilføjet en ny lejlighed. Du ROCKER!!!");
            modelAndView.addObject("user", user);
            modelAndView.addObject("apartment", new Apartment());
            modelAndView.setViewName("/apartment/add");

        }
        return modelAndView;
    }

    @RequestMapping("/apartment/add")
    public ModelAndView add(){
        ModelAndView modelAndView = new ModelAndView();
        Authentication auth = SecurityContextHolder.getContext().getAuthentication();
        User user = userService.findUserByEmail(auth.getName());
        modelAndView.addObject("user", user);
        modelAndView.addObject("adminMessage","Fedt man spa du er admin");
        modelAndView.addObject("apartment", new Apartment());
        modelAndView.setViewName("/apartment/add");
        return modelAndView;
    }

    @RequestMapping(value="/apartment/edit/{id}", method=RequestMethod.GET)
    public ModelAndView editApartmentPage(@PathVariable Integer id) {
        ModelAndView modelAndView = new ModelAndView("/apartment/add-edit");
        Authentication auth = SecurityContextHolder.getContext().getAuthentication();
        User user = userService.findUserByEmail(auth.getName());
        modelAndView.addObject("user", user);
        Apartment apartment = apartmentService.findById(id);
        modelAndView.addObject("apartment", apartment);
        return modelAndView;
    }

    @RequestMapping(value="/apartment/edit/{id}", method=RequestMethod.POST)
    public ModelAndView editApartment(@ModelAttribute @Valid Apartment apartment, BindingResult bindingResult, @PathVariable Integer id){
        ModelAndView modelAndView = new ModelAndView();
        Authentication auth = SecurityContextHolder.getContext().getAuthentication();
        User user = userService.findUserByEmail(auth.getName());
        modelAndView.addObject("user", user);
        if (bindingResult.hasErrors())
        {
            modelAndView.setViewName("/apartment/edit");
        }
        modelAndView.setViewName("redirect:/apartment");
        apartmentService.update(apartment);
        return modelAndView;
    }

    @RequestMapping(value="/apartment/delete/{id}", method=RequestMethod.GET)
    public ModelAndView deleteApartment(@PathVariable Integer id) {
        ModelAndView modelAndView = new ModelAndView("redirect:/apartment");
        apartmentService.delete(id);
        return modelAndView;
    }

}
```

```java
package dk.binfo.controllers;


import dk.binfo.models.User;
import dk.binfo.services.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;
/**
 *  @author Patrick Klæbel
 *
 */
@Controller
public class AccessDeniedController {

    @Autowired
    private UserService userService;

    @RequestMapping(value = {"/accessDenied"}, method = RequestMethod.GET)
    public ModelAndView accessDenied() {
        ModelAndView modelAndView = new ModelAndView();
        Authentication auth = SecurityContextHolder.getContext().getAuthentication();
        User user = userService.findUserByEmail(auth.getName());
        modelAndView.addObject("user", user);
        modelAndView.setViewName("/accessDenied");
        return modelAndView;
    }
}

```

```java
package dk.binfo.controllers;

import dk.binfo.models.User;
import dk.binfo.repositories.UserRepository;
import dk.binfo.services.EmailService;
import dk.binfo.services.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.ModelAndView;

/**
 *  @author Steen Petersen
 *
 */
@Controller
public class ForgotPasswordController {

    @Autowired
    private UserService userService;

    @Autowired
    private EmailService emailService;

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private BCryptPasswordEncoder bCryptPasswordEncoder;

    @RequestMapping(value={"/forgotpassword"}, method = RequestMethod.GET)
    public ModelAndView forgotPassword(){
        ModelAndView modelAndView = new ModelAndView();
        modelAndView.setViewName("forgotpassword");
        return modelAndView;
    }

    @RequestMapping(value={"/forgotpassword/{password}/{email}/newpassword"}, method = RequestMethod.GET)
    public ModelAndView forgotPassword(@PathVariable("email") String email, @PathVariable("password") String password){
        ModelAndView modelAndView = new ModelAndView();
        modelAndView.setViewName("forgotpasswordlink");
        User user = userService.findUserByEmail(email);
        if (user == null) {
            modelAndView.setViewName("redirect:/login");
        }else{
            if (user.getPassword().replace("/","").equals(password)){

            }else{
                modelAndView.setViewName("redirect:/login");
            }
        }
        return modelAndView;
    }

    @RequestMapping(value={"/forgotpassword/{password}/{email}/newpassword"}, method = RequestMethod.POST)
    public ModelAndView forgotPassword(@RequestParam String password1,@RequestParam String password2,@PathVariable
            ("email") String email, @PathVariable("password") String password){
        ModelAndView modelAndView = new ModelAndView();
        modelAndView.setViewName("forgotpasswordlink");
        User user = userService.findUserByEmail(email);
        if (user == null) {
            modelAndView.addObject("Message", "Linket virker ikke, prøv igen.");
        }else{
            if (user.getPassword().replace("/","").equals(password)){
                if (password1.equals(password2)){
                    modelAndView.addObject("Message", "Din kode er blevet ændret");
                    modelAndView.setViewName("redirect:/login");
                    user.setPassword(bCryptPasswordEncoder.encode(password1));
                    userRepository.save(user);
                }else{
                    modelAndView.addObject("Message", "Du har ik indtastet den samme kode");
                }
            }else{
                modelAndView.addObject("Message", "Linket virker ikke, prøv igen.");
            }
        }
        return modelAndView;
    }

    @RequestMapping(value = "/forgotpassword", method = RequestMethod.POST)
    public ModelAndView forgotPassword(@RequestParam String email) {
        ModelAndView modelAndView = new ModelAndView();
        User user = userService.findUserByEmail(email);
        if (user == null) {
            modelAndView.addObject("Message", "Emailen existere ikke, prøv igen.");
        }else{
            modelAndView.addObject("Message", "SUCCES!: Der er sent et link til din email!");
            sendResetEmail(user);
        }
        return modelAndView;
    }

    public void sendResetEmail(User user){
        String body = "<B>Reset dit password</B><br><br>";
        body += "Hvis du har bedt om at få dit password resat,<br> beder vi dig ";
        body += "beder vi dig klikke på nedenstående link .<br><br>";
        body += "<a href=\"http://localhost:8080/forgotpassword/"+user.getPassword().replace("/","")+"/"+user.getEmail()+"/newpassword\">Klik her for at reset dit password</a><br><br>";
        body += "Har du ikke bedt om et reset skal du bare  ignore denne email.<br><br>";
        body += "Mvh<br>";
        body += "Jens Boligforening<br>";
        body += "nørrebrogade 123<br>";
        body += "2312 nørrebro";
        emailService.generateAndSendEmail(user.getEmail(), "BoligInfo password reset",body);
    }
}
```

```java
package dk.binfo.repositories;

import dk.binfo.models.Role;
import org.springframework.data.jpa.repository.
JpaRepository;
import org.springframework.stereotype.Repository;
/**
 *  @author Patrick Klæbel
 *  @author Jens Bäckvall
 *  @author Steen Petersen
 *  @author Morten Olsen
 *
 */
@Repository("roleRepository")
public interface RoleRepository extends JpaRepository<Role,
 Integer>{
    Role findByRole(String role);

}

```

```java
package dk.binfo.repositories;

import dk.binfo.models.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
/**
 *  @author Patrick Klæbel
 *  @author Jens Bäckvall
 *  @author Steen Petersen
 *  @author Morten Olsen
 *
 */
@Repository("userRepository")
public interface UserRepository extends JpaRepository<User, String> {
    User findByEmail(String email);
}

```

```java
package dk.binfo.repositories;

import dk.binfo.models.Apartment;
import org.springframework.data.jpa.repository.
JpaRepository;
import org.springframework.stereotype.Repository;
/**
 *  @author Patrick Klæbel
 *  @author Jens Bäckvall
 *  @author Steen Petersen
 *  @author Morten Olsen
 *
 */
@Repository("apartmentRepository")
public interface ApartmentRepository extends JpaRepository<
Apartment, Integer> {

}
```

```java
package dk.binfo.configurations;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.builders.WebSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.web.access.AccessDeniedHandler;
import org.springframework.security.web.util.matcher.AntPathRequestMatcher;
import org.thymeleaf.extras.springsecurity4.dialect.SpringSecurityDialect;
import org.thymeleaf.spring4.SpringTemplateEngine;
import org.thymeleaf.templateresolver.ITemplateResolver;

/**
 *  @author Patrick Klæbel
 *  @author Steen Petersen
 *
 */
@Configuration
@EnableWebSecurity
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

    @Autowired
    private BCryptPasswordEncoder bCryptPasswordEncoder;


    @Autowired
    private UserDetailsService userDetailsService;

    @Override
    protected void configure(AuthenticationManagerBuilder auth)
```

```java
            throws Exception {
        auth
                .userDetailsService(userDetailsService)
                .passwordEncoder(bCryptPasswordEncoder);
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {

        http.authorizeRequests()
                .antMatchers("/", "/login", "/forgotpassword/**", "/registration", "/error").permitAll()
                .antMatchers("/apartment/**", "/users/**").hasAuthority("ADMIN")
                .anyRequest().authenticated().and().csrf().disable().formLogin()
                .loginPage("/login").failureUrl("/login?error=true").permitAll()
                .defaultSuccessUrl("/home")
                .usernameParameter("email").passwordParameter("password")
                .and().rememberMe().key("rem-me-key").rememberMeParameter("remember-me").rememberMeCookieName("remember-me")
                .and().logout().logoutRequestMatcher(new AntPathRequestMatcher("/logout")).logoutSuccessUrl("/login")
                .and().exceptionHandling().accessDeniedHandler(accessDeniedHandler());
    }

    @Override
    public void configure(WebSecurity web) throws Exception {
        web.ignoring().antMatchers("/resources/**", "/static/**", "/fonts/**", "/css/**", "/js/**", "/img/**");
    }

    @Bean
    public BCryptPasswordEncoder passwordEncoder() {
        BCryptPasswordEncoder bCryptPasswordEncoder = new BCryptPasswordEncoder();
        return bCryptPasswordEncoder;
    }

    @Bean
    public AccessDeniedHandler accessDeniedHandler(){
        return new CustomAccessDeniedHandler();
    }
```

```
}
```

```java
package dk.binfo.configurations;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import org.springframework.security.access.AccessDeniedException;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.web.access.AccessDeniedHandler;

/**
 * Whenever a user attempts to access a page that is restricted to roles they do not have,
 * the application will return a status code of 403, which means Access Denied.<p></p>
 * Following Roles
 * <ul>
 * <li>Admin
 * <li>User
 * </ul><p></p>
 *
 * By default, Spring Security has an ExceptionTranslationFilter defined
 * which handles exceptions of type AuthenticationException and AccessDeniedException.
 * The latter is done through a property called accessDeniedHandler,
 * which uses the AccessDeniedHandlerImpl class.
 * <p>
 * In order to customize this behavior to use our own page that we created (see <a href="/accessDenied">AccessDenied page</a>),
 * we need to override the properties of the ExceptionTranslationFilter class.
 * This can be done through either Java configuration or XML configuration.
 *<br>
 * Using Java, I customized the 403 error handling process by using the accessDeniedHandler()
 * methods while configuring the HttpSecurity element.
 *<br>
 * Using an access denied handler instead of a page has the advantage
 * that we can define custom logic to be executed before redirecting to the 403 page.
 * For this, we created a class that implements the AccessDeniedHandler interface
 * and overrides the handle() method.
 *
 * <p>
 * More implamentations:
 * <ul>
 *     <li>Add the logs to the database, with a page for admins to view logs</li>
 *     <li>Add log file if web application running on dedicated server</li>
 * </ul>
 */
public class CustomAccessDeniedHandler implements AccessDeniedHandler {

    /**
     *public final class LoggerFactory
     * extends Object
     * <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.
     * LoggerFactory is essentially a wrapper around an ILoggerFactory instance bound with LoggerFactory at compile time.<p></p>
     *
     * Please note that all methods in LoggerFactory are static.
     * <br>
     *     Author: Alexander Dorokhine, Robert Elliot, Ceki Gülcü
     */
    public static final Logger LOG = LoggerFactory.getLogger(CustomAccessDeniedHandler.class);

    @Override
    /**
     *<br>
     * The custom logic logs a warning message for every access denied attempt
     * containing the user that made the attempt and the protected URL they were trying to access:
     * <p>
     * 2017-11-30 10:42:47.236  WARN 5232 --- [nio-8080-exec-6] d.b.c.CustomAccessDeniedHandler
     * <br>User: user@test.dk[user] attempted to access the protected URL: /apartment<br>
     * This user tried to access /apartment which requied admin rank.
     *
     * @param request Extends the ServletRequest interface to provide request information for HTTP servlets.
     * @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.
     * @param exc Checked exception thrown when a operation is denied, typically due to a file permission or other access check.
     *
     */
        public void handle(HttpServletRequest request, HttpServletResponse response, AccessDeniedException exc) throws IOException, ServletException {
        Authentication auth = SecurityContextHolder.getContext().getAuthentication();
        if (auth != null) {
            LOG.warn("User: " + auth.getName() +  auth.getAuthorities() + " attempted to access the protected URL: " + request.getRequestURI());
        }
        response.sendRedirect(request.getContextPath() + "/accessDenied");
    }

}

```