CIS ARGE

Internship Project

Project Report

July 27, 2022

Developed By:

Arman Dehgani

Tuna Ozcan

Contents

1 Models

1.1 User.java

This model represents a user in the project that has attributes such as first name, last name, username, and so on.

```
@Entity
@Table(name = "users")
public class User {
// Attributes and Getters & Setters
}
```

One of the attributes of the `User` class is of type `Verification` which in the database holds the foreign key of the `verification` instance. In other words, in the database `User` does not have an attribute of type `Verification`, but instead, it has the foreign key of `verification`.

```
@OneToOne(cascade = CascadeType.ALL)
@JoinColumn(name = "verification_fk")
private Verification verification;
```

1.2 Verification.java

Every `user` has a `verification` attribute. The `Verification` class is a model that represents attributes that are related to verification of a user. For example, is the user verified, does the user's face match with their id card, and so on.

```
@Entity
@Table(name = "verifications")
public class Verification {
// Attributes and Getters & Setters
}
```

Two of Verification class attributes are Face and IdCard defined as:

```
@OneToOne(cascade = CascadeType.ALL)
@JoinColumn(name = "face_fk")
private Face face;

@OneToOne(cascade = CascadeType.ALL)
@JoinColumn(name = "idCard_fk")
private IdCard idCard;
```

Another attribute of Verification is `trainedYML` which is used to store the path of the trained.yml file. This is a file produced by OpenCV when we try to compare the user's face with their id card. This variable is defined as:

```
@Column(name = "trained_yml")
private String trainedYML;
```

1.3 Face.java

This class represents the Face of a user. It has the `imagePath` attribute which holds the path of the user's face image input.

```
@Table(name = "faces")
@Entity
public class Face {
// Attributes and Getters & Setters
}
```

1.4 IdCard.java

Just like the Face class, the IdCard class represents the id card of a user. It has the `imagePath` attribute which holds the path of the user's id card image input.

```
@Table(name = "id_cards")
@Entity
public class IdCard {
// Attributes and Getters & Setters
}
```

In addition, it has `idCardReadText` which holds the text that was read from the id card image, and `tcNumberRead` that has the id number read from the id card image.

```
@Column(name = "read_text", nullable = true, length = 5000)
private String idCardReadText;

@Column(name = "read_tc_number", nullable = true)
private String tcNumberRead;
```

2 Repositories

These repositories store the User, Verification, Face and IdCard objects to the database. The interfaces extend from the `JpaRepository`, and therefore, there is no need to implement the query for the defined methods below:

2.1 UserRepository.java

```
@Repository
public interface UserRepository extends JpaRepository<User, Long> {

Optional<User> findByTcNumber(String tcNumber);
Optional<User> findByUsername(String username);
Optional<User> findByUsernameAndPassword(String username, String password);

}
```

2.2 VerificationRepository.java

```
@Repository
public interface VerificationRepository
                              extends JpaRepository<Verification, Long> {

}
```

2.3 FaceRepository.java

```
@Repository
public interface FaceRepository extends JpaRepository<Face, Long> {

}
```

2.4 IdCardRepository.java

```
@Repository
public interface IdCardRepository extends JpaRepository<IdCard, Long> {

}
```

3 Services

3.1 SignupService.java

This class is responsible for signing up a new user. It makes sure that the new user can sign up (e.g., it checks if there is no other user with the same username, etc.)

The following attributes specify the length and patterns of the input during signup. For example, the length of the username, the characters that must be used in password, etc.

Please note that these current values and patterns are set as arbitrary values and must be changed according to your requirements.

```
@Service
public class SignupService {

/* First Name */
private static final int MINIMUM_FIRSTNAME_LENGTH = 2;
private static final int MAXIMUM_FIRSTNAME_LENGTH = 100;
private static final String FIRSTNAME_PATTERN
                                        = "^[a-zA-ZİiIıÖöÜüŞşÇçĞğ ,.'-]+$";

/* Last Name */
private static final int MINIMUM_LASTNAME_LENGTH = 2;
private static final int MAXIMUM_LASTNAME_LENGTH = 100;
private static final String LASTNAME_PATTERN
                                        = "^[a-zA-ZİiIıÖöÜüŞşÇçĞğ ,.'-]+$";
/* ID Number */
private static final int ID_LENGTH = 11;
private static final String ID_PATTERN = "^[0-9]+$";

/* Username */
private static final int MINIMUM_USERNAME_LENGTH = 2;
private static final int MAXIMUM_USERNAME_LENGTH = 100;
/* TODO TO BE CHANGED LATER */
private static final String USERNAME_PATTERN = ".*";

/* Password */
private static final int MINIMUM_PASSWORD_LENGTH = 2;
private static final int MAXIMUM_PASSWORD_LENGTH = 100;
/* TODO TO BE CHANGED LATER */
private static final String PASSWORD_NOT_ALLOWED_CHARACTERS_PATTERN = "";
private static final String PASSWORD_MUST_USE_CHARACTERS_PATTERN = "";
```

When a user is trying to sign up, the `signup()` method is called. This method will first call `isValid()` to check if the input values are valid or not. If valid, it creates a new user and saves the user in the `UserRepository`.

```
@Autowired
private UserRepository userRepository;


public User signup(String firstName, String lastName,
           String tcNumber, String username, String password) {

    /* This makes sure that the input values are valid for sign up */
    boolean isValid = validate(firstName, lastName,
                          tcNumber, username, password);

    /* If the input values are valid, sign up the user */
    if (isValid) {
        User user = new User();
        user.setFirstName(firstName);
        user.setLastName(lastName);
        user.setTcNumber(tcNumber);
        user.setUsername(username);
        user.setPassword(password);

        userRepository.save(user);
        return user;
    }

    /*
    Currently, when signup is successful, the signed up user is returned. If
    not successful, null is returned.
    However, signupStatus (defined in this class) can be returned instead, if
    necessary.
    */

    return null;
}
```

When `isValid()` is called, it will call specific methods for each input value to check if they are valid, as shown in the code below.

If any of the input values are not valid, `false` is returned. However, if all values are valid, `true` is returned.

```
66 public boolean validate(String firstName, String lastName,
67                String tcNumber, String username, String password) {
68
69     if (!isFirstNameCorrect(firstName)) {
70         return false;
71     }
72
73     if (!isLastNameCorrect(lastName)) {
74         return false;
75     }
76
77     if (!isTcNumberCorrect(tcNumber)) {
78         return false;
79     }
80
81     if (!isUsernameCorrect(username)) {
82         return false;
83     }
84
85     if (!isPasswordCorrect(password)) {
86         return false;
87     }
88
89     signupStatus = SignupStatus.SIGNED_UP_SUCCESSFULLY;
90     return true;
91 }
```

The following methods make sure that the first name and last name are both within the specified length, and that they have the pattern specified at the top of this class.

```
94  private boolean isFirstNameCorrect(String firstName) {
95
96      if (firstName.length() > MAXIMUM_FIRSTNAME_LENGTH) {
97          signupStatus = SignupStatus.INCORRECT_FIRSTNAME_TOO_LONG;
98          return false;
99      }
100     if (firstName.length() < MINIMUM_FIRSTNAME_LENGTH) {
101         signupStatus = SignupStatus.INCORRECT_FIRSTNAME_TOO_SHORT;
102         return false;
103     }
104
105     Pattern pattern = Pattern.compile(FIRSTNAME_PATTERN);
106     Matcher matcher = pattern.matcher(firstName);
107
108     if (!matcher.find()) {
109         signupStatus = SignupStatus.INCORRECT_NAME_CHARACTERS;
110         return false;
111     }
112     return true
113 }
114
115 private boolean isLastNameCorrect(String lastName) {
116
117     if (lastName.length() > MAXIMUM_LASTNAME_LENGTH) {
118         signupStatus = SignupStatus.INCORRECT_LASTNAME_TOO_LONG;
119         return false;
120     }
121     if (lastName.length() < MINIMUM_LASTNAME_LENGTH) {
122         signupStatus = SignupStatus.INCORRECT_LASTNAME_TOO_SHORT;
123         return false;
124     }
125
126     Pattern pattern = Pattern.compile(LASTNAME_PATTERN);
127     Matcher matcher = pattern.matcher(lastName);
128
129     if (!matcher.find()) {
130         signupStatus = SignupStatus.INCORRECT_LASTNAME_CHARACTERS;
131         return false;
132     }
133
134     return true;
135 }
```

This method makes sure that the input id number has the specified number of digits, and that it has the pattern specified above this class (i.e., only digits). Moreover, it checks if there is already a user with this id number or not.

```
94  private boolean isTcNumberCorrect(String tcNumber) {
95
96      if (tcNumber.length() > ID_LENGTH) {
97          signupStatus = SignupStatus.INCORRECT_ID_LENGTH_TOO_LONG;
98          return false;
99      }
100
101     if (tcNumber.length() < ID_LENGTH) {
102         signupStatus = SignupStatus.INCORRECT_ID_LENGTH_TOO_SHORT;
103         return false;
104     }
105
106     /* Tries to find a user with the given tcNumber */
107     Optional<User> userOptional = userRepository.findByTcNumber(tcNumber);
108
109     /*
110     If there is already a user with the given tcNumber, returns false and
111     sets the signupStatus to INCORRECT_ID_EXISTS.
112     */
113     if (userOptional.isPresent()) {
114         signupStatus = SignupStatus.INCORRECT_ID_EXISTS;
115         return false;
116     }
117
118     Pattern pattern = Pattern.compile(ID_PATTERN);
119     Matcher matcher = pattern.matcher(tcNumber);
120
121     if (!matcher.find()) {
122         signupStatus = SignupStatus.INCORRECT_ID_CHARACTERS;
123         return false;
124     }
125     return true;
126 }
```

The other two methods `isUsernameCorrect(String username)` and `isPasswordCorrect(String password)` are implemented in a similar way.

3.2 LoginService.java

The login service will try to find a user the given username and password. If a user is found, it is returned. If not found, `null` is returned instead.

```
@Service
public class LoginService {

    @Autowired
    private UserRepository userRepository;

    public User authenticate(String username, String password) {

        return userRepository.findByUsernameAndPassword(username, password)
                .orElse(null);

    }

}
```

3.3 IdCardService.java

This service is responsible for reading text from the id card image. To do so, it will first process the image to make it more readable and then passes the processed image to Tesseract so that it reads the text in the image. Then, it will store this read text as a string to the `idCardReadText` attribute of the `IdCard` class.

```
@Service
public class IdCardService {

/* Pattern for an 11 digit number */
private static final String TC_NUMBER_PATTERN = "([0-9]{11})";

/* Path for processed images */
private static final String processedImagesPath =
                                "VerificationFiles/ProcessedImages/";

/* Path for Tessdata (used by Tesseract) */
private static final String tessDataPath = "tessdata";

/* Language for Tesseract */
private static final String tessLanguage = "tur";

/* Used when saving image */
private static final String imageType = ".png";
```

```
public void readTextFromTcImage(IdCard idCard, String username) throws
TesseractException {

    String processedIdImagePath = processedImagesPath + username + imageType;

    /* Makes input image black & white */
    Mat gray = opencv_imgcodecs.imread(idCard.getImagePath(),
                                       opencv_imgcodecs.IMREAD_GRAYSCALE);

    /* Blurs image */
    Mat blurred = new Mat();
    opencv_imgproc.GaussianBlur(gray, blurred, new Size(3, 3), 1);

    /* Binarizes image */
    Mat binarized = new Mat();
    opencv_imgproc.adaptiveThreshold(
                                  blurred, binarized, 255,
                                  opencv_imgproc.ADAPTIVE_THRESH_MEAN_C,
                                  opencv_imgproc.THRESH_BINARY, 15, 40);

    /*
    Saves processed image (which has now become black & white,
    blurred and binarized).
    */
    opencv_imgcodecs.imwrite(processedIdImagePath, binarized);

    Tesseract tesseract = new Tesseract();
    tesseract.setDatapath(tessDataPath);
    tesseract.setLanguage(tessLanguage);

    /* Reads text from processed image */
    idCard.setIdCardReadText(tesseract.doOCR(new File(processedIdImagePath)));

    /* Finds the tc number from the read text */
    readTcNumber(idCard);
}
```

The following method will try to find an 11 digit number in the `idCardReadText` attribute of the `IdCard` class. After this value is found, it saves it to the `tcNumberRead` variable of the `idCard`.

```
60 private void readTcNumber(IdCard idCard) {
61
62     Pattern pattern = Pattern.compile(TC_NUMBER_PATTERN, Pattern.MULTILINE);
63     Matcher matcher = pattern.matcher(idCard.getIdCardReadText());
64
65     if (matcher.find())
66         idCard.setTcNumberRead(matcher.group());
67
68 }
```

3.4 VerificationService.java

This service is responsible for verifying a user. To do so, it will compare the signed up name and id number of a user with their id card. Moreover, the service will also compare the user input face image with their face on their id card.

The packages used in this class are the following.

```
1  import com.example.internship.model.Face;
2  import com.example.internship.model.IdCard;
3  import com.example.internship.model.User;
4  import com.example.internship.model.Verification;
5  import com.example.internship.repository.UserRepository;
6  import com.example.internship.status.VerificationStatus;
7  import net.sourceforge.tess4j.TesseractException;
8  import org.bytedeco.opencv.global.opencv_imgcodecs;
9  import org.bytedeco.opencv.global.opencv_imgproc;
10 import org.bytedeco.opencv.opencv_core.*;
11 import org.bytedeco.opencv.opencv_face.FaceRecognizer;
12 import org.bytedeco.opencv.opencv_face.LBPHFaceRecognizer;
13 import org.bytedeco.opencv.opencv_objdetect.CascadeClassifier;
14 import org.springframework.beans.factory.annotation.Autowired;
15 import org.springframework.stereotype.Service;
16 import org.springframework.transaction.annotation.Transactional;
17 import org.springframework.web.multipart.MultipartFile;
18 import java.io.File;
19 import java.io.IOException;
20 import java.util.ArrayList;
21 import java.util.Arrays;
22 import java.util.Locale;
```

Attributes of this class are given below and described next to each initialization.

```
25 @Service
26 public class VerificationService {
27
28     /*
29     Used to change strings with Turkish characters to English characters (for
30     comparison) E.g., if signed up name is Cagdas but name on the ID Card is
31     Çağdaş, the program will match the names.
32     */
33     private static final ArrayList<Character> turkishCharacters = new
34                 ArrayList<>(Arrays.asList('Ğ', 'Ü', 'Ş', 'İ', 'Ö', 'Ç'));
35     private static final ArrayList<Character> englishCharacters = new
36                 ArrayList<>(Arrays.asList('G', 'U', 'S', 'I', 'O', 'C'));
37
38     /* Path for the CascadeClassifier */
39     private static final String cascadeClassifierPath =
40     "CascadeClassifier/haarcascade_frontalface_default.xml";
41
42     /* Path to save the trained files */
43     private static final String trainedPath = "VerificationFiles/Trained/";
44
45     /* Name of id card images to save */
46     private static final String inputImagesPath =
47                                             "VerificationFiles/InputImages/";
48
49     /* Path to save the input images */
50     private static final String idCardImageName = "idCard";
51
52     /* Name of face images to save */
53     private static final String faceImageName = "face";
54
55     private static final String imageType = ".png";
56
57     /* Delimiter used in names of save images */
58     private static final String delimiter = "_";
59
60     /*
61     Language used when using the method toUpperCase() for strings
62     E.g., "ismail" becomes "İSMAİL" and not "ISMAIL", to make sure comparison
63     works as expected.
64     */
65     private static final String languageCode = "tr-TR";
66
67
```

```
68      /*
69      When there is a perfect match between faces, i.e., when
70      the same image is used.
71      */
72      private static final int PERFECT_MATCH_LEVEL = 5;
73      /*
74      Values between PERFECT_MATCH_LEVEL and MATCH_LEVEL
75      are considered as match.
76      */
77      private static final int MATCH_LEVEL = 65;
78
79      /* Used to resize images for openCV. */
80      private static final double MAX_IMAGE_HEIGHT = 500;
81
82
83      /*
84      USED TO DEBUG.
85      Path for faces using which OpenCV was trained.
86      */
87      private static final String trainedWithImagesPath =
88      "VerificationFiles/TrainedWithImages/";
89
90      /*
91      USED TO DEBUG.
92      Path for faces which were detected in the face image input
93      */
94      private static final String facesDetectedPath =
95      "VerificationFiles/FacesDetected/";
```

To verify a user, the `verify(User, idCardMultipartFile, faceMultipartFile)` method of the `VerificationService` is called. This method first converts MultipartFile to png, and then tries to verify idCard text and the face.

```
98      @Autowired
99      private IdCardService idCardService;
100
101     @Autowired
102     private UserRepository userRepository;
103
104     /*
105     idCardMultipartFile: multipart file of the idCard image.
106     faceMultipartFile: multipart file of the face image
107     */
108     public void verify(User user,
109                        MultipartFile idCardMultipartFile,
110                        MultipartFile faceMultipartFile
111                        ) throws IOException, TesseractException {
112
113     /*
114     These two lines convert MultipartFile to PNG so that
115     openCV and Tesseract can work.
116     */
117     String idCardImagePath = multipartFileToPng(idCardMultipartFile,
118                             user.getTcNumber() + delimiter + idCardImageName);
119     String faceImagePath = multipartFileToPng(faceMultipartFile,
120                             user.getTcNumber() + delimiter + faceImageName);
121
122     /* If user's verification is null, it instantiate ones */
123     if (user.getVerification() == null) {
124         user.setVerification(new Verification());
125     }
126     Verification verification = user.getVerification();
127
128     /* If verification's face is null, it instantiates one */
129     if (verification.getFace() == null) {
130         verification.setFace(new Face());
131     }
132     Face face = verification.getFace();
133
134     /* If verification's idCard is null, it instantiates one */
135     if (verification.getIdCard() == null) {
136         verification.setIdCard(new IdCard());
137     }
138     IdCard idCard = verification.getIdCard();
139
```

```
140     /*
141     With these two lines, face and idCard now have the faceImagePath
142     and idCardImagePath
143     */
144     face.setImagePath(faceImagePath);
145     idCard.setImagePath(idCardImagePath);
146 
147     /*
148     With these two lines, verification now has face and idCard
149     */
150     verification.setFace(face);
151     verification.setIdCard(idCard);
152 
153     /*
154     Verify methods are called
155     */
156     verifyIdCardText(user);
157     verifyFaces(user);
158 
159     /* The isVerified attribute of verification is updated */
160     verification.setVerified(verification.isFaceVerified() &&
161     verification.isIdCardVerified());
162 
163     /* The isVerified attribute of user is updated */
164     user.setVerified(verification.isVerified());
165 
166     /* User is updated in the database */
167     update(user);
168 }
```

The `verifyIdCardText(user)` method compares the signed up name, surname, and id number with the input idCard. To do so, it will first read the text from the id card, and later checks if the signed up name and surname are in the read text or not. For the id, it tries to find an 11 digit number in the read text, and then compares it to the signed up id number.

```
170 private void verifyIdCardText(User user) throws TesseractException {
171
172     IdCard idCard = user.getVerification().getIdCard();
173
174     /* This will read the text from the idCard */
175     idCardService.readTextFromTcImage(idCard, user.getUsername());
176
177     String signedUpFirstName =
178         user.getFirstName().toUpperCase(Locale.forLanguageTag(languageCode));
179     String signedUpLastName =
180         user.getLastName().toUpperCase(Locale.forLanguageTag(languageCode));
181
182     /*
183     The Turkish characters of firstName and lastName are changed
184     to English characters for comparison.
185     */
186     String FirstNameWithEnglishCharacters =
187             changeTurkishCharactersToEnglish(signedUpFirstName);
188     String LastNameWithEnglishCharacters =
189             changeTurkishCharactersToEnglish(signedUpLastName);
190
191     /* The text read from the idCard is assigned to idCardText */
192     String idCardText =
193     idCard.getIdCardReadText()
194                 .toUpperCase(Locale.forLanguageTag(languageCode));
195
196     /*
197     idCardTextWithEnglishCharacters is the idCardText, however, the Turkish
198     characters are changed to English for comparison
199     */
200     String idCardTextWithEnglishCharacters =
201     changeTurkishCharactersToEnglish(idCardText).toUpperCase();
202
203     /* Checks if signed up first and last name are in the idCardText or not */
204     boolean doNamesMatchWithTurkishCharacters =
205                     idCardText.contains(signedUpFirstName) &&
206                     idCardText.contains(signedUpLastName);
207
208
209
```

```
    /*
    Checks if signed up first and last name (with English characters) are
    in the idCard text (with English characters) or not
    */
    boolean doNamesMatchWithEnglishCharacters =
    idCardTextWithEnglishCharacters.contains(FirstNameWithEnglishCharacters)
    &&
    idCardTextWithEnglishCharacters.contains(LastNameWithEnglishCharacters);

    /* If either of the two boolean variables is true, then names match */
    boolean doNamesMatch = doNamesMatchWithEnglishCharacters ||
                                    doNamesMatchWithTurkishCharacters;

    /*
    Checks if the signed up id number is equal to the id number
    read from the idCard or not
    */
    boolean doTcNumbersMatch =
    idCard.getTcNumberRead().equals(user.getTcNumber());

    VerificationStatus status;
    boolean isVerified;

    if (doNamesMatch && doTcNumbersMatch) {
    status = VerificationStatus.ID_VERIFIED;
    isVerified = true;
    } else {
    status = VerificationStatus.ID_NOT_VERIFIED;
    isVerified = false;
    }

    /* isVerified attribute of user's verification is updated */
    user.getVerification().setIdCardVerified(isVerified);

    /* User's verification status is updated */
    user.getVerification().setIdCardVerificationStatus(status);

}
```

The `changeTurkishCharactersToEnglish(String string)` method changes the Turkish characters (e.g. 'Ğ', Ş' and 'Ü') in the string to English characters (e.g., 'G', 'S', 'U') for comparison.

```
250 private String changeTurkishCharactersToEnglish(String string) {
251     String output = "";
252 
253     for (int i = 0; i < string.length(); i++) {
254         char character = string.charAt(i);
255         if (turkishCharacters.contains(character))
256             character =
257                 englishCharacters.get(turkishCharacters.indexOf(character));
258 
259         output += character;
260     }
261 
262     return output;
263 }
```

Moreover, the `update(User user)` updates a user in the database.

```
265 @Transactional
266 public void update(User user) {
267 
268     userRepository
269 
270     /* Tries to find a user with the given username */
271     .findByUsername(user.getUsername())
272 
273         /* If user exists, updates the user */
274         .ifPresent(user1 -> {
275             user1.setVerification(user.getVerification());
276             userRepository.save(user1);
277     });
278 }
```

The `verifyFaces(User user)` method, compares the user's face image input with the user's idCard. To do so, it first trains the program with user's face on the idCard. Then, it tries to recognize user's face image input. Then, it outputs a confidence level starting from 0, where the lower the number is, the closer the two faces are. For example, a confidence level of 25 is a good match, whereas 195 is not a good match.

To train and recognize faces, first the faces should be detected. To do so, OpenCV is used with CascadeClassifier for face detection. Using these two, the program is able to detect any face in the input image.

```
280 private void verifyFaces(User user) {
281
282     Verification verification = user.getVerification();
283     Face inputFace = verification.getFace();
284
285     /* This trains with the face on the idCard image */
286     trainWithIdCardFace(user);
287
288      /* Path for the trained file */
289     String trained = verification.getTrainedYML();
290
291     /* This makes the face image black and white for openCV to work better */
292     Mat inputFaceImageGray = opencv_imgcodecs.imread(inputFace.getImagePath(),
293                              opencv_imgcodecs.IMREAD_GRAYSCALE);
294
295     FaceRecognizer faceRecognizer = LBPHFaceRecognizer.create();
296
297     /* Recognizer uses the trained file mentioned above */
298     faceRecognizer.read(trained);
299
300     /* This cascadeClassifier can be used to detect faces */
301     CascadeClassifier cascadeClassifier = new
302                          CascadeClassifier(cascadeClassifierPath);
303
304
305     /* Used for faces detected in the input face image */
306     RectVector rectVector = new RectVector();
307
308     /*
309     This detects faces in the face image and stores rectangles
310     for each detected face to the rectVector
311     */
312     cascadeClassifier.detectMultiScale(inputFaceImageGray, rectVector);
313
314
```

```
315     /* Used to store labels for each recognized face */
316     int[] labels = new int[(int) rectVector.size()];
317
318     /* Used to store the confidence levels of each recognized face */
319     double[] confidenceLevels = new double[(int) rectVector.size()];
320
321     /* For every detected face, this takes the face from the image, resizes it
322     so that its height is equal to the MAX_IMAGE_HEIGHT, and then
323     uses the faceRecognizer to recognize the face. */
324     int j = 0;
325     for (Rect rect : rectVector.get()) {
326         Mat face = new Mat(inputFaceImageGray, rect);
327
328         double factor = MAX_IMAGE_HEIGHT / face.size().height();
329
330         opencv_imgproc.resize(face, face, new Size(), factor, factor, 0);
331         faceRecognizer.predict(face, labels, confidenceLevels);
332
333         /* USED TO DEBUG */
334         opencv_imgcodecs.imwrite(facesDetectedPath + user.getUsername() +
335                                     delimiter + j++ + imageType, face);
336     }
337     int outputLabel = 0;
338     double outputConfidenceLevel = -1;
339     for (int i = 0; i < rectVector.size(); i++) {
340
341         /*
342         It is assumed that label 0 is only used when face is not recognized
343         So we only want to get the outputs for which a face was recognized
344         */
345         if (labels[i] != 0) {
346             /*
347             We also want to get the output which
348             had a better match, thus lower confidence level
349             */
350             if (outputConfidenceLevel == -1
351                     || confidenceLevels[i] < outputConfidenceLevel)
352
353                 outputLabel = labels[i];
354                 outputConfidenceLevel = confidenceLevels[i];
355         }
356     }
357     // USED TO DEBUG
358     System.out.println("Face Recognition Confidence Level: " +
359                                                 outputConfidenceLevel);
360
```

```
361
362     VerificationStatus status;
363     boolean isVerified = false;
364
365     if (rectVector.size() == 0) {
366     status = VerificationStatus.NO_FACE_DETECTED;
367     }
368     else if (outputLabel == 0 || outputConfidenceLevel > MATCH_LEVEL) {
369     status = VerificationStatus.FACES_DO_NOT_MATCH;
370     } else if (outputConfidenceLevel <= PERFECT_MATCH_LEVEL) {
371     status = VerificationStatus.FACES_MATCH_PERFECTLY;
372     } else {
373
374     status = VerificationStatus.FACES_MATCH;
375     isVerified = true;
376     }
377
378     /* verificaiton's status is updated */
379     verification.setFaceVerificationStatus(status);
380
381     /* verificaiton's isVerified attribute is updated */
382     verification.setFaceVerified(isVerified);
383 }
```

`trainWithIdCardFace(User user)` trains the program using the face on the user's idCard, so that it recognizes user's face image input. To do so, it first tries to detect faces on the idCard image. Later, it chooses the biggest rectangle of face to train. However, it was assumed that OpenCV needs at least 2 images to train. Since there is only one image, we used the same face 2 times to train the program.

```
385 private void trainWithIdCardFace(User user) {
386
387     Verification verification = user.getVerification();
388     IdCard idCard = verification.getIdCard();
389
390     String idCardImagePath = idCard.getImagePath();
391
392     FaceRecognizer recognizer = LBPHFaceRecognizer.create();
393
394
395     /* Makes the idCard image black and white */
396     Mat gray = opencv_imgcodecs.imread(idCardImagePath,
397                                        opencv_imgcodecs.IMREAD_GRAYSCALE);
398
399     /* Used to detect faces */
400     CascadeClassifier cascadeClassifier = new
401                           CascadeClassifier(cascadeClassifierPath);
402
403     /* Used for faces detected in the image */
404     RectVector rectVector = new RectVector();
405     cascadeClassifier.detectMultiScale(gray, rectVector);
406
407
408     MatVector faces = new MatVector();
409
410
411     /*
412     We want to choose the biggest rectangle
413     (i.e., the biggest face in the image)
414     */
415     Rect mainRect = new Rect(0, 0, 0, 0);
416     for (Rect rect : rectVector.get()) {
417
418         if (rect.width() >= mainRect.width()
419                               && rect.height() >= mainRect.height())
420             mainRect = rect;
421     }
422
423
```

```
424     Mat face = new Mat(gray, mainRect);
425 
426     /*
427     Apparently, OpenCV needs at least 2 images to train.
428     Since there is only one image to train, we use the same face 2 times.
429     */
430     faces.push_back(face);
431     faces.push_back(face);
432 
433 
434     // USED TO DEBUG
435     opencv_imgcodecs.imwrite(trainedWithImagesPath +
436                              user.getUsername() + imageType, face);
437 
438     Mat labels = new Mat((int) faces.size(), 1);
439 
440     recognizer.train(faces, labels);
441 
442     String trained = trainedPath + user.getUsername();
443 
444     /*
445     Saves the trained file, so it can be used to
446     recognize face in verifyFaces(User user) method
447     */
448     recognizer.save(trained);
449     verification.setTrainedYML(trained);
450 
451 }
```

4 Controllers

4.1 LoginController.java

```java
@RestController
public class LoginController {

    @Autowired
    LoginService loginService;

    @PostMapping("/login")
    public boolean login(@RequestBody LoginRequest loginRequest) {

        User auth = loginService.authenticate(loginRequest.getUsername(),
        loginRequest.getPassword());

        return auth != null;
    }

}
```

4.2 SignupController.java

```java
@RestController
public class SignupController {

    @Autowired
    SignupService signupService;

    @PostMapping("/signup")
    public boolean signup(@ModelAttribute User user) {

    User signedUpUser = signupService.signup(
                                    user.getFirstName(),
                                    user.getLastName(),
                                    user.getTcNumber(),
                                    user.getUsername(),
                                    user.getPassword());
        return signedUpUser != null;
    }

}
```

4.3 VerificationController.java

```java
@RestController
@RequestMapping("/verification")
public class VerificationController {

    @Autowired
    VerificationService verificationService;

    @Autowired
    UserRepository userRepository;


    @PostMapping("/verify/{username}")
    public String verify(@PathVariable String username,
    MultipartFile idCard,
    MultipartFile faceImage) throws IOException,
    TesseractException {

        Optional<User> userOptional = userRepository.findByUsername(username);

        if (userOptional.isEmpty()) {
            return VerificationStatus.USERNAME_NOT_FOUND.name();
        }

        User user = userOptional.get();
        verificationService.verify(user, idCard, faceImage);
        Verification verification = user.getVerification();

        String output = "Face Status: " +
        verification.getFaceVerificationStatus().name() +
        "\tFace Verified: " +
        verification.isFaceVerified();

        output += "\nIdCard Status: " +
        verification.getIdCardVerificationStatus().name() +
        "\tIdCard Verified: " +
        verification.isIdCardVerified();

        output += "\nIs User Verified: " + verification.isVerified();

        return output;
    }


```

```
44      @GetMapping("/status/{username}")
45      public String status(@PathVariable String username) {
46
47          Optional<User> userOptional = userRepository.findByUsername(username);
48
49          if (userOptional.isEmpty()) {
50              return VerificationStatus.USERNAME_NOT_FOUND.name();
51          }
52
53          User user = userOptional.get();
54          Verification verification = user.getVerification();
55
56          if (verification == null) {
57              return VerificationStatus.NOT_VERIFIED_YET.name();
58          }
59
60          return verification.getFaceVerificationStatus().name() + "\n" +
61          verification.getIdCardVerificationStatus().name();
62          }
63
64
65      @GetMapping("/face-status/{username}")
66      public String faceStatus(@PathVariable String username) {
67
68          Optional<User> userOptional = userRepository.findByUsername(username);
69
70          if (userOptional.isEmpty()) {
71              return VerificationStatus.USERNAME_NOT_FOUND.name();
72          }
73
74          User user = userOptional.get();
75          Verification verification = user.getVerification();
76
77          if (verification == null) {
78              return VerificationStatus.NOT_VERIFIED_YET.name();
79          }
80
81          return verification.getFaceVerificationStatus().name();
82      }
83
84
85
86
87
88
89
```

```
90      @GetMapping("/id-status/{username}")
91      public String idCardStatus(@PathVariable String username) {
92
93          Optional<User> userOptional = userRepository.findByUsername(username);
94
95          if (userOptional.isEmpty()) {
96              return VerificationStatus.USERNAME_NOT_FOUND.name();
97          }
98
99          User user = userOptional.get();
100         Verification verification = user.getVerification();
101
102         if (verification == null) {
103             return VerificationStatus.NOT_VERIFIED_YET.name();
104         }
105
106         return verification.getIdCardVerificationStatus().name();
107     }
108
109
110     @GetMapping("/is-verified/{username}")
111     public boolean isVerified(@PathVariable String username) {
112
113         Optional<User> userOptional = userRepository.findByUsername(username);
114
115         if (userOptional.isEmpty()) {
116             return false;
117         }
118
119         User user = userOptional.get();
120         Verification verification = user.getVerification();
121
122         if (verification == null)
123             return false;
124
125         return verification.isVerified();
126     }
127 }
```

5 Request

5.1 LoginRequest.java

This class represents a login request, which has two attributes username and password.

```java
public class LoginRequest {

    private String username;
    private String password;

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

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