ДОДАТОК А

Код програми

Файл FeatureType.java

```
package ua.nure.myronenko.facerecognitionservice.constant;
public enum FeatureType {
    PCA
}
```

Файл MainController.java

```
package ua.nure.myronenko.facerecognitionservice.controller;
import ua.nure.myronenko.facerecognitionservice.training.Trainer;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import java.awt.*;
import java.util.Optional;
@Controller
@RequestMapping("/api")
public class MainController {
    private final Trainer trainer;
    public MainController(Trainer trainer) {
        this.trainer = trainer;
    }
    @SuppressWarnings("rawtypes")
    @PostMapping(name = "/", produces = "application/json")
    public ResponseEntity login(@RequestParam Image image) {
        return  ResponseEntity.of(Optional.of(trainer.getModel()));
    }
    @SuppressWarnings({"rawtypes", "unchecked"})
    @Autowired
    @PutMapping(name = "/trainer/face", produces = "application/json")
    public ResponseEntity addImageToTrainer(@RequestParam Image image) {
        return  ResponseEntity.of(trainer.add(image));
    }
    @SuppressWarnings({"rawtypes", "unchecked"})
    @Autowired
    @PostMapping(name = "/trainer/train", produces = "application/json")
    public ResponseEntity train(@RequestParam Long userId) {
        return  ResponseEntity.of(trainer.train(userId));
    }
}
```

Файл Maths.java

```
package ua.nure.myronenko.facerecognitionservice.decomposition.util;
public class Maths {
   public static double hypot(double a, double b) {
      double r;
      if (Math.abs(a) > Math.abs(b)) {
         r = b/a;
         r = Math.abs(a)*Math.sqrt(1+r*r);
      } else if (b != 0) {
         r = a/b;
         r = Math.abs(b)*Math.sqrt(1+r*r);
      } else {
         r = 0.0;
      }
      return r;
   }
}
```

Файл CholeskyDecomposition.java

```
package ua.nure.myronenko.facerecognitionservice.decomposition;
public class CholeskyDecomposition implements java.io.Serializable {
   private double[][] L;
   private int n;
   private boolean isspd;
   public CholeskyDecomposition (Matrix Arg) {
      double[][] A = Arg.getArray();
      n = Arg.getRowDimension();
      L = new double[n][n];
      isspd = (Arg.getColumnDimension() == n);
      for (int j = 0; j < n; j++) {
         double[] Lrowj = L[j];
```

```
         double d = 0.0;
         for (int k = 0; k < j; k++) {
            double[] Lrowk = L[k];
            double s = 0.0;
            for (int i = 0; i < k; i++) {
               s += Lrowk[i]*Lrowj[i];
            }
            Lrowj[k] = s = (A[j][k] - s)/L[k][k];
            d = d + s*s;
            isspd = isspd & (A[k][j] == A[j][k]);
         }
         d = A[j][j] - d;
         isspd = isspd & (d > 0.0);
         L[j][j] = Math.sqrt(Math.max(d,0.0));
         for (int k = j+1; k < n; k++) {
            L[j][k] = 0.0;
         }
      }
   }
   public boolean isSPD () {
      return isspd;
   }
   public Matrix getL () {
      return new Matrix(L,n,n);
   }
   public Matrix solve (Matrix B) {
      if (B.getRowDimension() != n) {
         throw new IllegalArgumentException("Matrix row dimensions must agree.");
      }
      if (!isspd) {
         throw new RuntimeException("Matrix is not symmetric positive definite.");
      }
      double[][] X = B.getArrayCopy();
      int nx = B.getColumnDimension();
                           for (int k = 0; k < n; k++) {
                             for (int j = 0; j < nx; j++) {
                                for (int i = 0; i < k ; i++) {
                                    X[k][j] -= X[i][j]*L[k][i];
                                }
                                X[k][j] /= L[k][k];
                             }
                           }
                           for (int k = n-1; k >= 0; k--) {
                             for (int j = 0; j < nx; j++) {
                                for (int i = k+1; i < n ; i++) {
                                    X[k][j] -= X[i][j]*L[i][k];
                                }
                                X[k][j] /= L[k][k];
                             }
                           }
      return new Matrix(X,n,nx);
   }
  private static final long serialVersionUID = 1;
}
```

Файл EigenvalueDecomposition.java

```
package ua.nure.myronenko.facerecognitionservice.decomposition;
import ua.nure.myronenko.facerecognitionservice.decomposition.util.Maths;
public class EigenvalueDecomposition implements java.io.Serializable {
   private int n;
   private boolean issymmetric;
   private double[] d, e;
   private double[][] V;
   private double[][] H;
   private double[] ort;
   private void tred2 () {
      for (int j = 0; j < n; j++) {
         d[j] = V[n-1][j];
      }
      for (int i = n-1; i > 0; i--) {
         double scale = 0.0;
         double h = 0.0;
         for (int k = 0; k < i; k++) {
            scale = scale + Math.abs(d[k]);
```

```
      }
      if (scale == 0.0) {
         e[i] = d[i-1];
         for (int j = 0; j < i; j++) {
            d[j] = V[i-1][j];
            V[i][j] = 0.0;
            V[j][i] = 0.0;
         }
      } else {
         for (int k = 0; k < i; k++) {
            d[k] /= scale;
            h += d[k] * d[k];
         }
         double f = d[i-1];
         double g = Math.sqrt(h);
         if (f > 0) {
            g = -g;
         }
         e[i] = scale * g;
         h = h - f * g;
         d[i-1] = f - g;
         for (int j = 0; j < i; j++) {
            e[j] = 0.0;
         }
         for (int j = 0; j < i; j++) {
            f = d[j];
            V[j][i] = f;
            g = e[j] + V[j][j] * f;
            for (int k = j+1; k <= i-1; k++) {
               g += V[k][j] * d[k];
               e[k] += V[k][j] * f;
            }
            e[j] = g;
         }
         f = 0.0;
         for (int j = 0; j < i; j++) {
            e[j] /= h;
            f += e[j] * d[j];
         }
         double hh = f / (h + h);
         for (int j = 0; j < i; j++) {
            e[j] -= hh * d[j];
         }
         for (int j = 0; j < i; j++) {
            f = d[j];
            g = e[j];
            for (int k = j; k <= i-1; k++) {
               V[k][j] -= (f * e[k] + g * d[k]);
            }
            d[j] = V[i-1][j];
            V[i][j] = 0.0;
         }
      }
      d[i] = h;
   }
   for (int i = 0; i < n-1; i++) {
      V[n-1][i] = V[i][i];
      V[i][i] = 1.0;
      double h = d[i+1];
      if (h != 0.0) {
         for (int k = 0; k <= i; k++) {
            d[k] = V[k][i+1] / h;
         }
         for (int j = 0; j <= i; j++) {
            double g = 0.0;
            for (int k = 0; k <= i; k++) {
               g += V[k][i+1] * V[k][j];
            }
            for (int k = 0; k <= i; k++) {
               V[k][j] -= g * d[k];
            }
         }
      }
      for (int k = 0; k <= i; k++) {
         V[k][i+1] = 0.0;
      }
   }
   for (int j = 0; j < n; j++) {
      d[j] = V[n-1][j];
      V[n-1][j] = 0.0;
```

```
      }
      V[n-1][n-1] = 1.0;
      e[0] = 0.0;
   }
   private void tql2 () {
      for (int i = 1; i < n; i++) {
         e[i-1] = e[i];
      }
      e[n-1] = 0.0;
      double f = 0.0;
      double tst1 = 0.0;
      double eps = Math.pow(2.0,-52.0);
      for (int l = 0; l < n; l++) {
         tst1 = Math.max(tst1,Math.abs(d[l]) + Math.abs(e[l]));
         int m = l;
         while (m < n) {
            if (Math.abs(e[m]) <= eps*tst1) {
               break;
            }
            m++;
         }
         if (m > l) {
            int iter = 0;
            do {
               iter = iter + 1;
               double g = d[l];
               double p = (d[l+1] - g) / (2.0 * e[l]);
               double r = Maths.hypot(p, 1.0);
               if (p < 0) {
                  r = -r;
               }
               d[l] = e[l] / (p + r);
               d[l+1] = e[l] * (p + r);
               double dl1 = d[l+1];
               double h = g - d[l];
               for (int i = l+2; i < n; i++) {
                  d[i] -= h;
               }
               f = f + h;
               p = d[m];
               double c = 1.0;
               double c2 = c;
               double c3 = c;
               double el1 = e[l+1];
               double s = 0.0;
               double s2 = 0.0;
               for (int i = m-1; i >= l; i--) {
                  c3 = c2;
                  c2 = c;
                  s2 = s;
                  g = c * e[i];
                  h = c * p;
                  r = Maths.hypot(p,e[i]);
                  e[i+1] = s * r;
                  s = e[i] / r;
                  c = p / r;
                  p = c * d[i] - s * g;
                  d[i+1] = h + s * (c * g + s * d[i]);
                  for (int k = 0; k < n; k++) {
                     h = V[k][i+1];
                     V[k][i+1] = s * V[k][i] + c * h;
                     V[k][i] = c * V[k][i] - s * h;
                  }
               }
               p = -s * s2 * c3 * el1 * e[l] / dl1;
               e[l] = s * p;
               d[l] = c * p;
            } while (Math.abs(e[l]) > eps*tst1);
         }
         d[l] = d[l] + f;
         e[l] = 0.0;
      }
      for (int i = 0; i < n-1; i++) {
         int k = i;
         double p = d[i];
         for (int j = i+1; j < n; j++) {
            if (d[j] < p) {
               k = j;
               p = d[j];
            }
```

```
      }
      if (k != i) {
         d[k] = d[i];
         d[i] = p;
         for (int j = 0; j < n; j++) {
            p = V[j][i];
            V[j][i] = V[j][k];
            V[j][k] = p;
         }
      }
   }
}
private void orthes () {
   int low = 0;
   int high = n-1;
   for (int m = low+1; m <= high-1; m++) {
      double scale = 0.0;
      for (int i = m; i <= high; i++) {
         scale = scale + Math.abs(H[i][m-1]);
      }
      if (scale != 0.0) {
         double h = 0.0;
         for (int i = high; i >= m; i--) {
            ort[i] = H[i][m-1]/scale;
            h += ort[i] * ort[i];
         }
         double g = Math.sqrt(h);
         if (ort[m] > 0) {
            g = -g;
         }
         h = h - ort[m] * g;
         ort[m] = ort[m] - g;
         for (int j = m; j < n; j++) {
            double f = 0.0;
            for (int i = high; i >= m; i--) {
               f += ort[i]*H[i][j];
            }
            f = f/h;
            for (int i = m; i <= high; i++) {
               H[i][j] -= f*ort[i];
            }
          }
          for (int i = 0; i <= high; i++) {
            double f = 0.0;
            for (int j = high; j >= m; j--) {
               f += ort[j]*H[i][j];
            }
            f = f/h;
            for (int j = m; j <= high; j++) {
               H[i][j] -= f*ort[j];
            }
          }
          ort[m] = scale*ort[m];
          H[m][m-1] = scale*g;
      }
   }
   for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
         V[i][j] = (i == j ? 1.0 : 0.0);
      }
   }
   for (int m = high-1; m >= low+1; m--) {
      if (H[m][m-1] != 0.0) {
         for (int i = m+1; i <= high; i++) {
            ort[i] = H[i][m-1];
         }
         for (int j = m; j <= high; j++) {
            double g = 0.0;
            for (int i = m; i <= high; i++) {
               g += ort[i] * V[i][j];
            }
            g = (g / ort[m]) / H[m][m-1];
            for (int i = m; i <= high; i++) {
               V[i][j] += g * ort[i];
            }
         }
      }
   }
}
private transient double cdivr, cdivi;
```

```
private void cdiv(double xr, double xi, double yr, double yi) {
   double r,d;
   if (Math.abs(yr) > Math.abs(yi)) {
      r = yi/yr;
      d = yr + r*yi;
      cdivr = (xr + r*xi)/d;
      cdivi = (xi - r*xr)/d;
   } else {
      r = yr/yi;
      d = yi + r*yr;
      cdivr = (r*xr + xi)/d;
      cdivi = (r*xi - xr)/d;
   }
}
private void hqr2 () {
   int nn = this.n;
   int n = nn-1;
   int low = 0;
   int high = nn-1;
   double eps = Math.pow(2.0,-52.0);
   double exshift = 0.0;
   double p=0,q=0,r=0,s=0,z=0,t,w,x,y;
   double norm = 0.0;
   for (int i = 0; i < nn; i++) {
      if (i < low | i > high) {
         d[i] = H[i][i];
         e[i] = 0.0;
      }
      for (int j = Math.max(i-1,0); j < nn; j++) {
         norm = norm + Math.abs(H[i][j]);
      }
   }
   int iter = 0;
   while (n >= low) {
      int l = n;
      while (l > low) {
         s = Math.abs(H[l-1][l-1]) + Math.abs(H[l][l]);
         if (s == 0.0) {
            s = norm;
         }
         if (Math.abs(H[l][l-1]) < eps * s) {
            break;
         }
         l--;
      }
      if (l == n) {
         H[n][n] = H[n][n] + exshift;
         d[n] = H[n][n];
         e[n] = 0.0;
         n--;
         iter = 0;
      } else if (l == n-1) {
         w = H[n][n-1] * H[n-1][n];
         p = (H[n-1][n-1] - H[n][n]) / 2.0;
         q = p * p + w;
         z = Math.sqrt(Math.abs(q));
         H[n][n] = H[n][n] + exshift;
         H[n-1][n-1] = H[n-1][n-1] + exshift;
         x = H[n][n];
         if (q >= 0) {
            if (p >= 0) {
               z = p + z;
            } else {
               z = p - z;
            }
            d[n-1] = x + z;
            d[n] = d[n-1];
            if (z != 0.0) {
               d[n] = x - w / z;
            }
            e[n-1] = 0.0;
            e[n] = 0.0;
            x = H[n][n-1];
            s = Math.abs(x) + Math.abs(z);
            p = x / s;
            q = z / s;
            r = Math.sqrt(p * p+q * q);
            p = p / r;
            q = q / r;
            for (int j = n-1; j < nn; j++) {
```

```
            z = H[n-1][j];
            H[n-1][j] = q * z + p * H[n][j];
            H[n][j] = q * H[n][j] - p * z;
         }
         for (int i = 0; i <= n; i++) {
            z = H[i][n-1];
            H[i][n-1] = q * z + p * H[i][n];
            H[i][n] = q * H[i][n] - p * z;
         }
         for (int i = low; i <= high; i++) {
            z = V[i][n-1];
            V[i][n-1] = q * z + p * V[i][n];
            V[i][n] = q * V[i][n] - p * z;
         }
      } else {
         d[n-1] = x + p;
         d[n] = x + p;
         e[n-1] = z;
         e[n] = -z;
      }
      n = n - 2;
      iter = 0;
   } else {
      x = H[n][n];
      y = 0.0;
      w = 0.0;
      if (l < n) {
         y = H[n-1][n-1];
         w = H[n][n-1] * H[n-1][n];
      }
      if (iter == 10) {
         exshift += x;
         for (int i = low; i <= n; i++) {
            H[i][i] -= x;
         }
         s = Math.abs(H[n][n-1]) + Math.abs(H[n-1][n-2]);
         x = y = 0.75 * s;
         w = -0.4375 * s * s;
      }
      if (iter == 30) {
          s = (y - x) / 2.0;
          s = s * s + w;
          if (s > 0) {
              s = Math.sqrt(s);
              if (y < x) {
                 s = -s;
              }
              s = x - w / ((y - x) / 2.0 + s);
              for (int i = low; i <= n; i++) {
                 H[i][i] -= s;
              }
              exshift += s;
              x = y = w = 0.964;
          }
      }
      iter = iter + 1;
      int m = n-2;
      while (m >= l) {
         z = H[m][m];
         r = x - z;
         s = y - z;
         p = (r * s - w) / H[m+1][m] + H[m][m+1];
         q = H[m+1][m+1] - z - r - s;
         r = H[m+2][m+1];
         s = Math.abs(p) + Math.abs(q) + Math.abs(r);
         p = p / s;
         q = q / s;
         r = r / s;
         if (m == l) {
            break;
         }
         if (Math.abs(H[m][m-1]) * (Math.abs(q) + Math.abs(r)) <
            eps * (Math.abs(p) * (Math.abs(H[m-1][m-1]) + Math.abs(z) +
            Math.abs(H[m+1][m+1])))) {
               break;
         }
         m--;
      }

      for (int i = m+2; i <= n; i++) {
```

```
            H[i][i-2] = 0.0;
            if (i > m+2) {
               H[i][i-3] = 0.0;
            }
         }
         for (int k = m; k <= n-1; k++) {
            boolean notlast = (k != n-1);
            if (k != m) {
               p = H[k][k-1];
               q = H[k+1][k-1];
               r = (notlast ? H[k+2][k-1] : 0.0);
               x = Math.abs(p) + Math.abs(q) + Math.abs(r);
               if (x == 0.0) {
                   continue;
               }
               p = p / x;
               q = q / x;
               r = r / x;
            }

            s = Math.sqrt(p * p + q * q + r * r);
            if (p < 0) {
               s = -s;
            }
            if (s != 0) {
               if (k != m) {
                  H[k][k-1] = -s * x;
               } else if (l != m) {
                  H[k][k-1] = -H[k][k-1];
               }
               p = p + s;
               x = p / s;
               y = q / s;
               z = r / s;
               q = q / p;
               r = r / p;
            }
         }
      }
   }
   if (norm == 0.0) {
      return;
   }
   for (n = nn-1; n >= 0; n--) {
      p = d[n];
      q = e[n];
      if (q == 0) {
         int l = n;
         H[n][n] = 1.0;
         for (int i = n-1; i >= 0; i--) {
            w = H[i][i] - p;
            r = 0.0;
            for (int j = l; j <= n; j++) {
               r = r + H[i][j] * H[j][n];
            }
            if (e[i] < 0.0) {
               z = w;
               s = r;
            } else {
               l = i;
               t = Math.abs(H[i][n]);
               if ((eps * t) * t > 1) {
                  for (int j = i; j <= n; j++) {
                     H[j][n] = H[j][n] / t;
                  }
               }
            }
         }
      } else if (q < 0) {
         int l = n-1;
         if (Math.abs(H[n][n-1]) > Math.abs(H[n-1][n])) {
            H[n-1][n-1] = q / H[n][n-1];
            H[n-1][n] = -(H[n][n] - p) / H[n][n-1];
         } else {
            cdiv(0.0,-H[n-1][n],H[n-1][n-1]-p,q);
            H[n-1][n-1] = cdivr;
            H[n-1][n] = cdivi;
         }
         H[n][n-1] = 0.0;
         H[n][n] = 1.0;
```

```
            for (int i = n-2; i >= 0; i--) {
               double ra,sa,vr,vi;
               ra = 0.0;
               sa = 0.0;
               for (int j = l; j <= n; j++) {
                  ra = ra + H[i][j] * H[j][n-1];
                  sa = sa + H[i][j] * H[j][n];
               }
               w = H[i][i] - p;
               if (e[i] < 0.0) {
                  z = w;
                  r = ra;
                  s = sa;
               } else {
                  l = i;
                  t = Math.max(Math.abs(H[i][n-1]),Math.abs(H[i][n]));
                  if ((eps * t) * t > 1) {
                     for (int j = i; j <= n; j++) {
                        H[j][n-1] = H[j][n-1] / t;
                        H[j][n] = H[j][n] / t;
                     }
                  }
               }
            }
         }
      }
      for (int j = nn-1; j >= low; j--) {
         for (int i = low; i <= high; i++) {
            z = 0.0;
            for (int k = low; k <= Math.min(j,high); k++) {
               z = z + V[i][k] * H[k][j];
            }
            V[i][j] = z;
         }
      }
   }
   public EigenvalueDecomposition (Matrix Arg) {
      double[][] A = Arg.getArray();
      n = Arg.getColumnDimension();
      V = new double[n][n];
      d = new double[n];
      e = new double[n];
      issymmetric = true;
      for (int j = 0; (j < n) & issymmetric; j++) {
         for (int i = 0; (i < n) & issymmetric; i++) {
            issymmetric = (A[i][j] == A[j][i]);
         }
      }
      if (issymmetric) {
         for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
               V[i][j] = A[i][j];
            }
         }
         tred2();
         tql2();
      } else {
         H = new double[n][n];
         ort = new double[n];

         for (int j = 0; j < n; j++) {
            for (int i = 0; i < n; i++) {
               H[i][j] = A[i][j];
            }
         }
         orthes();
         hqr2();
      }
   }
   public Matrix getV () {
      return new Matrix(V,n,n);
   }
   public double[] getRealEigenvalues () {
      return d;
   }
   public double[] getImagEigenvalues () {
      return e;
   }
   public Matrix getD () {
      Matrix X = new Matrix(n,n);
```

```
      double[][] D = X.getArray();
      for (int i = 0; i < n; i++) {
         for (int j = 0; j < n; j++) {
            D[i][j] = 0.0;
         }
         D[i][i] = d[i];
         if (e[i] > 0) {
            D[i][i+1] = e[i];
         } else if (e[i] < 0) {
            D[i][i-1] = e[i];
         }
      }
      return X;
   }
   public double[] getd(){
                           return d;
   }
  private static final long serialVersionUID = 1;
}
```

Файл Matrix.java

```
package ua.nure.myronenko.facerecognitionservice.decomposition;
import java.io.BufferedReader;
import java.io.StreamTokenizer;
public class Matrix implements Cloneable, java.io.Serializable {
   private double[][] A;
   private int m, n;
   public Matrix (int m, int n) {
      this.m = m;
      this.n = n;
      A = new double[m][n];
   }
   public Matrix (int m, int n, double s) {
      this.m = m;
      this.n = n;
      A = new double[m][n];
      for (int i = 0; i < m; i++) {
         for (int j = 0; j < n; j++) {
            A[i][j] = s;
         }
      }
   }
   public Matrix (double[][] A) {
      m = A.length;
      n = A[0].length;
      for (int i = 0; i < m; i++) {
         if (A[i].length != n) {
            throw new IllegalArgumentException("All rows must have the same length.");
         }
      }
      this.A = A;
   }
   public Matrix (double[][] A, int m, int n) {
      this.A = A;
      this.m = m;
      this.n = n;
   }
   public Matrix (double vals[], int m) {
      this.m = m;
      n = (m != 0 ? vals.length/m : 0);
      if (m*n != vals.length) {
         throw new IllegalArgumentException("Array length must be a multiple of m.");
      }
      A = new double[m][n];
      for (int i = 0; i < m; i++) {
         for (int j = 0; j < n; j++) {
            A[i][j] = vals[i+j*m];
         }
      }
   }
   public Matrix copy () {
      Matrix X = new Matrix(m,n);
      double[][] C = X.getArray();
      for (int i = 0; i < m; i++) {
         for (int j = 0; j < n; j++) {
            C[i][j] = A[i][j];
         }
      }
      return X;
   }
   public Object clone () {
```

```
   return this.copy();
}
public double[][] getArray () {
   return A;
}
public double[][] getArrayCopy () {
   double[][] C = new double[m][n];
   for (int i = 0; i < m; i++) {
      for (int j = 0; j < n; j++) {
         C[i][j] = A[i][j];
      }
   }
   return C;
}
public int getRowDimension () {
   return m;
}
public int getColumnDimension () {
   return n;
}
public double get (int i, int j) {
   return A[i][j];
}
public Matrix getMatrix (int i0, int i1, int j0, int j1) {
   Matrix X = new Matrix(i1-i0+1,j1-j0+1);
   double[][] B = X.getArray();
   try {
      for (int i = i0; i <= i1; i++) {
         for (int j = j0; j <= j1; j++) {
            B[i-i0][j-j0] = A[i][j];
         }
      }
   } catch(ArrayIndexOutOfBoundsException e) {
      throw new ArrayIndexOutOfBoundsException("Submatrix indices");
   }
   return X;
}
public Matrix getMatrix (int i0, int i1, int[] c) {
   Matrix X = new Matrix(i1-i0+1,c.length);
   double[][] B = X.getArray();
   try {
      for (int i = i0; i <= i1; i++) {
         for (int j = 0; j < c.length; j++) {
            B[i-i0][j] = A[i][c[j]];
         }
      }
   } catch(ArrayIndexOutOfBoundsException e) {
      throw new ArrayIndexOutOfBoundsException("Submatrix indices");
   }
   return X;
}
public Matrix getMatrix (int[] r, int j0, int j1) {
   Matrix X = new Matrix(r.length,j1-j0+1);
   double[][] B = X.getArray();
   try {
      for (int i = 0; i < r.length; i++) {
         for (int j = j0; j <= j1; j++) {
            B[i][j-j0] = A[r[i]][j];
         }
      }
   } catch(ArrayIndexOutOfBoundsException e) {
      throw new ArrayIndexOutOfBoundsException("Submatrix indices");
   }
   return X;
}
public void set (int i, int j, double s) {
   A[i][j] = s;
}
public void setMatrix (int i0, int i1, int j0, int j1, Matrix X) {
   try {
      for (int i = i0; i <= i1; i++) {
         for (int j = j0; j <= j1; j++) {
            A[i][j] = X.get(i-i0,j-j0);
         }
      }
   } catch(ArrayIndexOutOfBoundsException e) {
      throw new ArrayIndexOutOfBoundsException("Submatrix indices");
   }
}
public Matrix transpose () {
```

```
   Matrix X = new Matrix(n,m);
   double[][] C = X.getArray();
   for (int i = 0; i < m; i++) {
      for (int j = 0; j < n; j++) {
         C[j][i] = A[i][j];
      }
   }
   return X;
}
public Matrix plusEquals (Matrix B) {
   checkMatrixDimensions(B);
   for (int i = 0; i < m; i++) {
      for (int j = 0; j < n; j++) {
         A[i][j] = A[i][j] + B.A[i][j];
      }
   }
   return this;
}
public Matrix minus (Matrix B) {
   checkMatrixDimensions(B);
   Matrix X = new Matrix(m,n);
   double[][] C = X.getArray();
   for (int i = 0; i < m; i++) {
      for (int j = 0; j < n; j++) {
         C[i][j] = A[i][j] - B.A[i][j];
      }
   }
   return X;
}
public Matrix times (double s) {
   Matrix X = new Matrix(m,n);
   double[][] C = X.getArray();
   for (int i = 0; i < m; i++) {
      for (int j = 0; j < n; j++) {
         C[i][j] = s*A[i][j];
      }
   }
   return X;
}
public Matrix times (Matrix B) {
   if (B.m != n) {
      throw new IllegalArgumentException("Matrix inner dimensions must agree.");
   }
   Matrix X = new Matrix(m,B.n);
   double[][] C = X.getArray();
   double[] Bcolj = new double[n];
   for (int j = 0; j < B.n; j++) {
      for (int k = 0; k < n; k++) {
         Bcolj[k] = B.A[k][j];
      }
      for (int i = 0; i < m; i++) {
         double[] Arowi = A[i];
         double s = 0;
         for (int k = 0; k < n; k++) {
            s += Arowi[k]*Bcolj[k];
         }
         C[i][j] = s;
      }
   }
   return X;
}
public EigenvalueDecomposition eig () {
   return new EigenvalueDecomposition(this);
}
public Matrix solve (Matrix B) {
   return (m == n ? (new LUDecomposition(this)).solve(B) :
                    (new QRDecomposition(this)).solve(B));
}
public Matrix solveTranspose (Matrix B) {
   return transpose().solve(B.transpose());
}
public Matrix inverse () {
   return solve(identity(m,m));
}
public double det () {
   return new LUDecomposition(this).det();
}
public double trace () {
   double t = 0;
   for (int i = 0; i < Math.min(m,n); i++) {
```

```
            t += A[i][i];
         }
         return t;
      }
      public static Matrix identity (int m, int n) {
         Matrix A = new Matrix(m,n);
         double[][] X = A.getArray();
         for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
               X[i][j] = (i == j ? 1.0 : 0.0);
            }
         }
         return A;
      }

      public static Matrix read (BufferedReader input) throws java.io.IOException {
         StreamTokenizer tokenizer= new StreamTokenizer(input);
         tokenizer.resetSyntax();
         tokenizer.wordChars(0,255);
         tokenizer.whitespaceChars(0, ' ');
         tokenizer.eolIsSignificant(true);
         java.util.Vector<Double> vD = new java.util.Vector<Double>();
         while (tokenizer.nextToken() == StreamTokenizer.TT_EOL);
         if (tokenizer.ttype == StreamTokenizer.TT_EOF)
                           throw new java.io.IOException("Unexpected EOF on matrix read.");
         do {
            vD.addElement(Double.valueOf(tokenizer.sval)); // Read & store 1st row.
         } while (tokenizer.nextToken() == StreamTokenizer.TT_WORD);
         int n = vD.size();
         double row[] = new double[n];
         for (int j=0; j<n; j++)
            row[j]=vD.elementAt(j).doubleValue();
         java.util.Vector<double[]> v = new java.util.Vector<double[]>();
         v.addElement(row);
         while (tokenizer.nextToken() == StreamTokenizer.TT_WORD) {
            v.addElement(row = new double[n]);
            int j = 0;
            do {
               if (j >= n) throw new java.io.IOException
                  ("Row " + v.size() + " is too long.");
               row[j++] = Double.valueOf(tokenizer.sval).doubleValue();
            } while (tokenizer.nextToken() == StreamTokenizer.TT_WORD);
            if (j < n) throw new java.io.IOException
               ("Row " + v.size() + " is too short.");
         }
         int m = v.size();
         double[][] A = new double[m][];
         v.copyInto(A);
         return new Matrix(A);
      }
      private void checkMatrixDimensions (Matrix B) {
         if (B.m != m || B.n != n) {
            throw new IllegalArgumentException("Matrix dimensions must agree.");
         }
      }
     private static final long serialVersionUID = 1;
}
```

Файл QRDecomposition.java

```
package ua.nure.myronenko.facerecognitionservice.decomposition;
import ua.nure.myronenko.facerecognitionservice.decomposition.util.Maths;
public class QRDecomposition implements java.io.Serializable {
    private double[][] QR;
    private int m, n;
    private double[] Rdiag;
    public QRDecomposition(Matrix A) {
        QR = A.getArrayCopy();
        m = A.getRowDimension();
        n = A.getColumnDimension();
        Rdiag = new double[n];
        for (int k = 0; k < n; k++) {
            double nrm = 0;
            for (int i = k; i < m; i++) {
                nrm = Maths.hypot(nrm, QR[i][k]);
            }
            if (nrm != 0.0) {
                if (QR[k][k] < 0) {
                    nrm = -nrm;
                }
                for (int i = k; i < m; i++) {
                    QR[i][k] /= nrm;
```

```
                }
                QR[k][k] += 1.0;
            }
            Rdiag[k] = -nrm;
        }
    }

    public boolean isFullRank() {
        for (int j = 0; j < n; j++) {
            if (Rdiag[j] == 0)
                return false;
        }
        return true;
    }
    public Matrix solve(Matrix B) {
        if (B.getRowDimension() != m) {
            throw new IllegalArgumentException("Matrix row dimensions must agree.");
        }
        if (!this.isFullRank()) {
            throw new RuntimeException("Matrix is rank deficient.");
        }

        int nx = B.getColumnDimension();
        double[][] X = B.getArrayCopy();

        for (int k = 0; k < n; k++) {
            for (int j = 0; j < nx; j++) {
                double s = 0.0;
                for (int i = k; i < m; i++) {
                    s += QR[i][k] * X[i][j];
                }
                s = -s / QR[k][k];
                for (int i = k; i < m; i++) {
                    X[i][j] += s * QR[i][k];
                }
            }
        }
        for (int k = n - 1; k >= 0; k--) {
            for (int j = 0; j < nx; j++) {
                X[k][j] /= Rdiag[k];
            }
            for (int i = 0; i < k; i++) {
                for (int j = 0; j < nx; j++) {
                    X[i][j] -= X[k][j] * QR[i][k];
                }
            }
        }
        return (new Matrix(X, n, nx).getMatrix(0, n - 1, 0, nx - 1));
    }
    private static final long serialVersionUID = 1;
}
```

Файл CosineDissimilarity.java

```
package ua.nure.myronenko.facerecognitionservice.training;
import ua.nure.myronenko.facerecognitionservice.decomposition.Matrix;
public class CosineDissimilarity implements Metric {
                        @Override
                        public double getDistance(Matrix a, Matrix b) {
                                assert a.getRowDimension() == b.getRowDimension();
                                int size = a.getRowDimension();
                                double cosine, sNorm, eNorm, se;
                                int i;
                                se = 0;
                                for (i = 0; i < size; i++) {
                                      se += a.get(i, 0) * b.get(i, 0);
                                }
                                sNorm = 0;
                                for (i = 0; i < size; i++) {
                                      sNorm += Math.pow(a.get(i, 0), 2);
                                }
                                sNorm = Math.sqrt(sNorm);
                                eNorm = 0;
                                for (i = 0; i < size; i++) {
                                      eNorm += Math.pow(b.get(i, 0), 2);
                                }
                                eNorm = Math.sqrt(eNorm);
                                if(se < 0)
                                      se = 0 - se;
                                cosine = se / (eNorm * sNorm);
                                if (cosine == 0.0)
                                      return Double.MAX_VALUE;
```

```
                                return 1 / cosine;
                        }
}
```

Файл FeatureExtraction.java

```
package ua.nure.myronenko.facerecognitionservice.training;
import ua.nure.myronenko.facerecognitionservice.decomposition.Matrix;
import java.util.ArrayList;
public abstract class FeatureExtraction {
                        ArrayList<Matrix> trainingSet;
                        ArrayList<String> labels;
                        int numOfComponents;
                        Matrix meanMatrix;
                        Matrix W;
                        ArrayList<ProjectedTrainingMatrix> projectedTrainingSet;
                        public abstract Matrix getW();
                        public abstract ArrayList<ProjectedTrainingMatrix>
getProjectedTrainingSet();
                        public abstract Matrix getMeanMatrix();
                        public abstract int addFace(Matrix face, String label);
}
```

Файл FileManager.java

```
package ua.nure.myronenko.facerecognitionservice.training;
import ua.nure.myronenko.facerecognitionservice.decomposition.Matrix;
import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.awt.image.WritableRaster;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.Scanner;
public class FileManager {
    public static Matrix convertPGMtoMatrix(String address) throws IOException {
        FileInputStream fileInputStream = new FileInputStream(address);
        Scanner scan = new Scanner(fileInputStream);

        scan.nextLine();
        int picWidth = scan.nextInt();
        int picHeight = scan.nextInt();

        fileInputStream.close();

        fileInputStream = new FileInputStream(address);
        DataInputStream dis = new DataInputStream(fileInputStream);

        int numnewlines = 3;
        while (numnewlines > 0) {
            char c;
            do {
                c = (char) (dis.readUnsignedByte());
            } while (c != '\n');
            numnewlines--;
        }

        double[][] data2D = new double[picHeight][picWidth];
        for (int row = 0; row < picHeight; row++) {
            for (int col = 0; col < picWidth; col++) {
                data2D[row][col] = dis.readUnsignedByte();
            }
        }
        return new Matrix(data2D);
    }

    public static Matrix normalize(Matrix input) {
        int row = input.getRowDimension();

        for (int i = 0; i < row; i++) {
            input.set(i, 0, 0 - input.get(i, 0));

        }

        double max = input.get(0, 0);
        double min = input.get(0, 0);

        for (int i = 1; i < row; i++) {
            if (max < input.get(i, 0))
                max = input.get(i, 0);

            if (min > input.get(i, 0))
```

```
                min = input.get(i, 0);

        }

        Matrix result = new Matrix(112, 92);
        for (int p = 0; p < 92; p++) {
            for (int q = 0; q < 112; q++) {
                double value = input.get(p * 112 + q, 0);
                value = (value - min) * 255 / (max - min);
                result.set(q, p, value);
            }
        }

        return result;

    }
    public static void convertMatricetoImage(Matrix x, int featureMode) throws IOException {
        int row = x.getRowDimension();
        int column = x.getColumnDimension();

        for (int i = 0; i < column; i++) {
            Matrix eigen = normalize(x.getMatrix(0, row - 1, i, i));
            BufferedImage img = new BufferedImage(92, 112, BufferedImage.TYPE_BYTE_GRAY);
            WritableRaster raster = img.getRaster();
            for (int m = 0; m < 112; m++) {
                for (int n = 0; n < 92; n++) {
                    int value = (int) eigen.get(m, n);
                    raster.setSample(n, m, 0, value);
                }
            }
            File file = null;
            if (featureMode == 0)
                file = new File("Eigenface" + i + ".bmp");
            else if (featureMode == 1)
                file = new File("Fisherface" + i + ".bmp");
            else if (featureMode == 2)
                file = new File("Laplacianface" + i + ".bmp");

            if (!file.exists())
                file.createNewFile();

            ImageIO.write(img, "bmp", file);
        }
    }
}
```

Файл Metric.java

```
package ua.nure.myronenko.facerecognitionservice.training;
import ua.nure.myronenko.facerecognitionservice.decomposition.Matrix;
public interface Metric {
                        double getDistance(Matrix a, Matrix b);
}
```

Файл PCA.java

```
package ua.nure.myronenko.facerecognitionservice.training;
import ua.nure.myronenko.facerecognitionservice.decomposition.EigenvalueDecomposition;
import ua.nure.myronenko.facerecognitionservice.decomposition.Matrix;
import java.util.ArrayList;
import java.util.Arrays;
public class PCA extends FeatureExtraction {
                        public PCA(ArrayList<Matrix> trainingSet, ArrayList<String> labels,
                                        int numOfComponents) throws Exception {
                                if(numOfComponents >= trainingSet.size()){
                                      throw new Exception("the expected dimensions could not be
achieved!");
                                }
                                this.trainingSet = trainingSet;
                                this.labels = labels;
                                this.numOfComponents = numOfComponents;

                                this.meanMatrix = getMean(this.trainingSet);
                                this.W = getFeature(this.trainingSet, this.numOfComponents);

                                this.projectedTrainingSet = new ArrayList<>();
                                for (int i = 0; i < trainingSet.size(); i++) {
                                      ProjectedTrainingMatrix ptm = new
ProjectedTrainingMatrix(this.W

                        .transpose().times(trainingSet.get(i).minus(meanMatrix)),
                                                labels.get(i));
                                      this.projectedTrainingSet.add(ptm);
```

```
			}
		}
		private Matrix getFeature(ArrayList<Matrix> input, int K) {
			int i, j;
			int row = input.get(0).getRowDimension();
			int column = input.size();
			Matrix X = new Matrix(row, column);
			for (i = 0; i < column; i++) {
				X.setMatrix(0, row - 1, i, i,
input.get(i).minus(this.meanMatrix));
			}
			Matrix XT = X.transpose();
			Matrix XTX = XT.times(X);
			EigenvalueDecomposition feature = XTX.eig();
			double[] d = feature.getd();
			assert d.length >= K : "number of eigenvalues is less than K";
			int[] indexes = this.getIndexesOfKEigenvalues(d, K);
			Matrix eigenVectors = X.times(feature.getV());
			Matrix selectedEigenVectors = eigenVectors.getMatrix(0,
					eigenVectors.getRowDimension() - 1, indexes);
			row = selectedEigenVectors.getRowDimension();
			column = selectedEigenVectors.getColumnDimension();
			for (i = 0; i < column; i++) {
				double temp = 0;
				for (j = 0; j < row; j++)
					temp += Math.pow(selectedEigenVectors.get(j, i), 2);
				temp = Math.sqrt(temp);

				for (j = 0; j < row; j++) {
					selectedEigenVectors.set(j, i,
selectedEigenVectors.get(j, i) / temp);
				}
			}
			return selectedEigenVectors;

		}
		private class mix implements Comparable {
			int index;
			double value;

			mix(int i, double v) {
				index = i;
				value = v;
			}
			public int compareTo(Object o) {
				double target = ((mix) o).value;
				if (value > target)
					return -1;
				else if (value < target)
					return 1;

				return 0;
			}
		}
		private int[] getIndexesOfKEigenvalues(double[] d, int k) {
			mix[] mixes = new mix[d.length];
			int i;
			for (i = 0; i < d.length; i++)
				mixes[i] = new mix(i, d[i]);
			Arrays.sort(mixes);
			int[] result = new int[k];
			for (i = 0; i < k; i++)
				result[i] = mixes[i].index;
			return result;
		}

		private static Matrix getMean(ArrayList<Matrix> input) {
			int rows = input.get(0).getRowDimension();
			int length = input.size();
			Matrix all = new Matrix(rows, 1);
			for (int i = 0; i < length; i++) {
				all.plusEquals(input.get(i));
			}
			return all.times((double) 1 / length);
		}
		@Override
		public Matrix getW() {
			return this.W;
		}
		@Override
```

```
                        public ArrayList<ProjectedTrainingMatrix> getProjectedTrainingSet() {
                                return this.projectedTrainingSet;
                        }
                        @Override
                        public Matrix getMeanMatrix() {
                                return meanMatrix;
                        }
                        @Override
                        public int addFace(Matrix face, String label) {
                                return 0;
                        }
                        public ArrayList<Matrix> getTrainingSet(){
                                return this.trainingSet;
                        }
                        public Matrix reconstruct(int whichImage, int dimensions) throws Exception{
                                if(dimensions > this.numOfComponents)
                                      throw new Exception("dimensions should be smaller than the
number of components");

                                Matrix afterPCA =
this.projectedTrainingSet.get(whichImage).matrix.getMatrix(0, dimensions-1, 0, 0);
                                Matrix eigen = this.getW().getMatrix(0, 10304-1, 0, dimensions - 1);
                                return eigen.times(afterPCA).plus(this.getMeanMatrix());

                        }
}
```

Файл ProjectedTrainingMatrix.java

```
package ua.nure.myronenko.facerecognitionservice.training;
import ua.nure.myronenko.facerecognitionservice.decomposition.Matrix;
public class ProjectedTrainingMatrix {
                        Matrix matrix;
                        String label;
                        double distance = 0;
                        public ProjectedTrainingMatrix(Matrix m, String l) {
                                this.matrix = m;
                                this.label = l;
                        }
}
```

Файл Trainer.java

```
package ua.nure.myronenko.facerecognitionservice.training;
import ua.nure.myronenko.facerecognitionservice.constant.FeatureType;
import ua.nure.myronenko.facerecognitionservice.decomposition.Matrix;
import com.google.common.base.Preconditions;
import lombok.Builder;
import lombok.Getter;
import java.awt.*;
import java.util.ArrayList;
import java.util.Objects;
import java.util.Optional;
@Builder
@Getter
public class Trainer {
    Metric metric;
    FeatureType featureType;
    FeatureExtraction featureExtraction;
    int numberOfComponents;
    int k;
    ArrayList<Matrix> trainingSet;
    ArrayList<String> trainingLabels;
    ArrayList<ProjectedTrainingMatrix> model;
    public void add(Matrix matrix, String label) {
        if (Objects.isNull(trainingSet)) {
            trainingSet = new ArrayList<>();
            trainingLabels = new ArrayList<>();
        }

        trainingSet.add(matrix);
        trainingLabels.add(label);
    }
    public Optional add(Image image) {
        return  Optional.empty();
    }
    public Optional train(Long userId) {
        return  Optional.empty();
    }
    public void addFaceAfterTraining(Matrix matrix, String label) {
        featureExtraction.addFace(matrix, label);
    }
    public void train() throws Exception {
```

```
        Preconditions.checkNotNull(metric);
        Preconditions.checkNotNull(featureType);
        Preconditions.checkNotNull(numberOfComponents);
        Preconditions.checkNotNull(trainingSet);
        Preconditions.checkNotNull(trainingLabels);

        if (featureType == FeatureType.PCA) {
            featureExtraction = new PCA(trainingSet, trainingLabels, numberOfComponents);
        }

        model = featureExtraction.getProjectedTrainingSet();
    }
    public String recognize(Matrix matrix) {
        Matrix testCase =
featureExtraction.getW().transpose().times(matrix.minus(featureExtraction.getMeanMatrix()));
        String result = KNN.assignLabel(model.toArray(new ProjectedTrainingMatrix[0]), testCase, k,
metric);
        return result;
    }
}
```

Файл FaceRecognitionService.java

```
package ua.nure.myronenko.facerecognitionservice;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class FaceRecognitionService {
    public static void main(String[] args) {
        SpringApplication.run(FaceRecognitionService.class, args);
    }
}
```

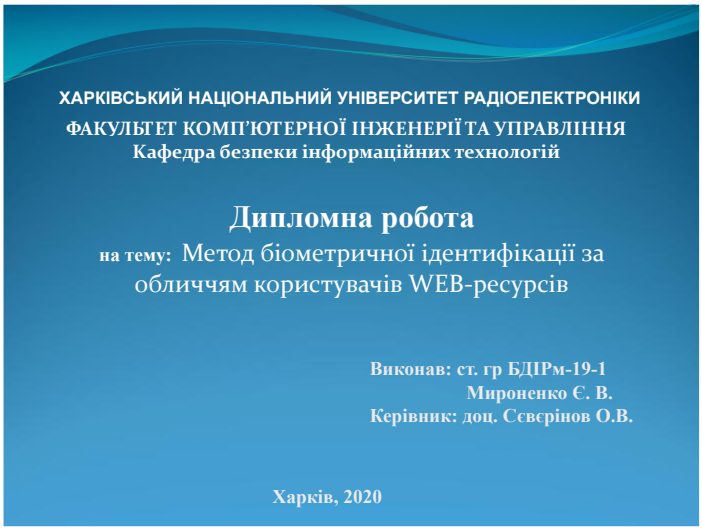
Файл Pom.xml

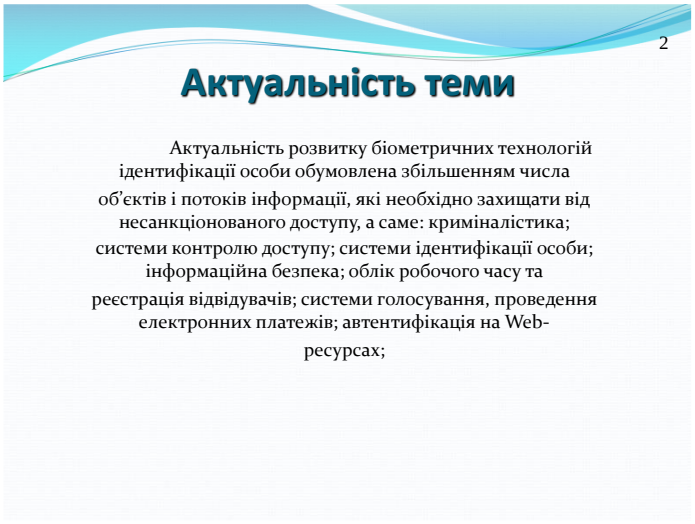
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
  <groupId>ua.nure.myronenko.facerecognitionservice</groupId>
  <artifactId>face-recognition</artifactId>
  <version>1.0</version>
  <modelVersion>4.0.0</modelVersion>
  <name>Face Recognition</name>
  <description>Implement face recognition with  Java</description>
  <distributionManagement>
    <snapshotRepository>
      <id>ossrh</id>
      <url>https://oss.sonatype.org/content/repositories/snapshots</url>
    </snapshotRepository>
    <repository>
      <id>ossrh</id>
      <url>https://oss.sonatype.org/service/local/staging/deploy/maven2/</url>
    </repository>
  </distributionManagement>
  <dependencies>
    <dependency>
      <groupId>org.projectlombok</groupId>
      <artifactId>lombok</artifactId>
      <version>1.16.10</version>
    </dependency>
    <dependency>
      <groupId>com.google.guava</groupId>
      <artifactId>guava</artifactId>
      <version>19.0</version>
    </dependency>
      <dependency>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-starter-test</artifactId>
          <version>2.4.1</version>
      </dependency>
      <dependency>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot</artifactId>
          <version>2.4.1</version>
      </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>5.3.2</version>
    </dependency>
    <dependency>
```

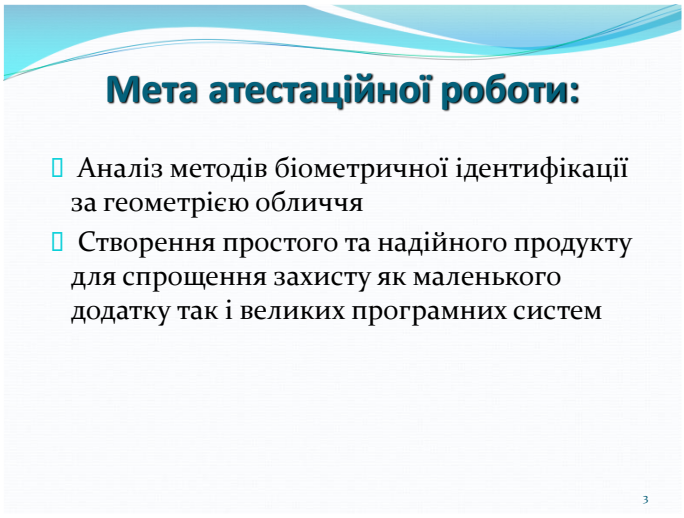
```
      <groupId>org.springframework.data</groupId>
      <artifactId>spring-data-jpa</artifactId>
      <version>2.4.2</version>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.2</version>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
        </configuration>
      </plugin>
      <plugin>
        <artifactId>maven-source-plugin</artifactId>
        <version>2.4</version>
        <executions>
          <execution>
            <id>attach-source</id>
            <goals>
              <goal>jar-no-fork</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
      <plugin>
        <artifactId>maven-javadoc-plugin</artifactId>
        <version>2.10.3</version>
        <executions>
          <execution>
            <id>attach-javadocs</id>
            <goals>
              <goal>jar</goal>
            </goals>
            <configuration>
              <additionalparam>-Xdoclint:none</additionalparam>
            </configuration>
          </execution>
        </executions>
      </plugin>

      <plugin>
        <artifactId>maven-deploy-plugin</artifactId>
        <version>2.8.2</version>
        <configuration>
          <skip>true</skip>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.sonatype.plugins</groupId>
        <artifactId>nexus-staging-maven-plugin</artifactId>
        <version>1.6.6</version>
        <executions>
          <execution>
            <id>default-deploy</id>
            <phase>deploy</phase>
            <goals>
              <goal>deploy</goal>
            </goals>
          </execution>
        </executions>
        <extensions>true</extensions>
        <configuration>
          <serverId>ossrh</serverId>
          <nexusUrl>https://oss.sonatype.org/</nexusUrl>
          <skipStagingRepositoryClose>true</skipStagingRepositoryClose>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

ДОДАТОК Б

Графічні матеріали





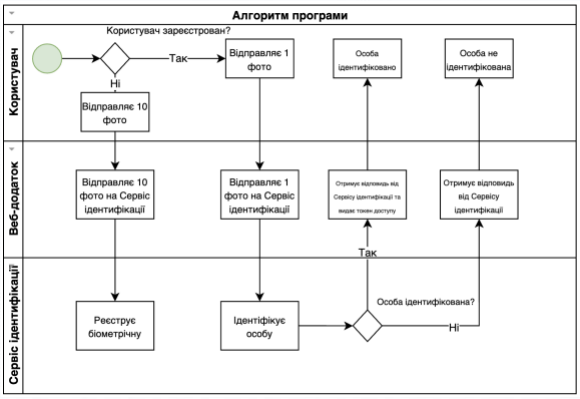


Для досягнення мети було поставлено наступні завдання:

- Ознайомитись з методами розпізнавання обличчя
- Порівняти сучасні технології ідентифікації особи за обличчям
- Проаналізувати сучасні існуючі продукти
- Розробити алгоритм створення ПЗ
- Розробити ПЗ, протестувати

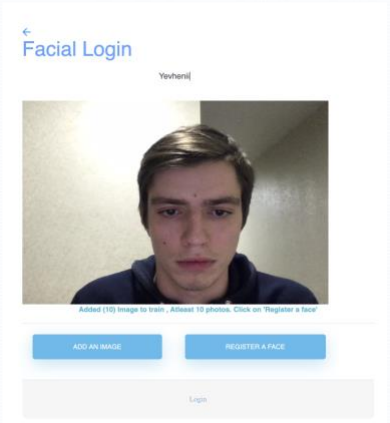
4

Алгоритм програми:



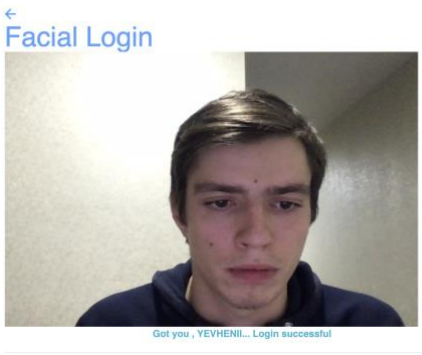
5

Реєстрація користувача



6

Вдала ідентифікація



7

Висновки:

У ході виконання атестаційної роботи були вирішені наступні питання

- Проведено аналіз методів біометричної ідентифікації за геометрією обличчя
- Проведено аналіз середовища розробки
- Проведено аналіз існуючих розробок
- Розроблено програмне забезпечення

8

Висновки:

ПРОБЛЕМИ ІНФОРМАТИЗАЦІЇ

ТЕЗИ ДОПОВІДЕЙ ВОСЬМОЇ МІЖНАРОДНОЇ НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

26 – 27 листопада 2020 року

Том 1: секції 1 – 3

БІОМЕТРИЧНА ІДЕНТИФІКАЦІЯ І АВТЕНТИФІКАЦІЯ ОСОБИ ЗА ГЕОМЕТРІЄЮ ОБЛИЧЧЯ
с.46

9

№ док.	Позначення	Найменування	Дод. відом.
1		Текстові документи	
	ГЮІК.ХХХХХ1413Ст.05ПЗ	Пояснювальна записка	70 с.
2		Графічні документи	
		Презентаційний матеріал	9 слайдів
3		Інші документи	
		Електронна версія ПЗ	1
		Відзив керівника	1 с.
		Рецензія	1 с.

Изм.	Лист	№ докум.	Підпис	Дата	ГЮІК. ХХХХХ1413Ст.05ВД		
Розроб.		Мироненко Є. В			Метод біометричної ідентифікації за обличчям користувачів WEB-	Лит.	Лист 1 / Листів 1
Перевір.		Сєвєрінов О.В.				Т	
Н.контр.		Шевченко Л.П.				ХНУРЕ Кафедра БІТ	
Затверд.		Халімов Г.З.					