

# ALUMNI PORTAL

## SECURITY ASSESSMENT

### INTRODUCTION

This document aims to reveal vulnerabilities and recommendations for improvement of the client's system with the respect to KISS principle, of having strong security, not obscurity. The system can interact both with external and internal users. Users can input the data using the interface. This means that the front-end of the system has to be protected as well as local devices connected to the system because layering security controls and risk mitigation safeguards into software design incorporates the principle of defence in depth.

### OVERALL CONCLUSION

The residual risk, as follows, according to the DREAD method:

1. Before the implementation of the actions, listed below was close to 100%. This risk rate was because of accessibility with unverified email address, ability to check if the login exists in the database, etc.
2. The residual risk after the implementation of the mitigation actions is 2. This is because the ability to input malicious code using the login page is still available. The application works directly with the server so can become a path of impact to the server if not redirected through the firewall and DMZ.
3. The residual risk after the implementation of the recommended mitigation actions is 1. This is because there still are risks of session hijacking, the data will not be encrypted. But to perform any hacking on this level will require more resources, so make it less profitable and hard in realization.

### RECOMMENDATIONS

Because the application is not a part of a big system, and has its own database the overall risk for the client is not high and mitigation of the risks will not require high budgets. However, the application holds sensitive data that may have a negative impact on client and employees in case of disclosure.

Following 20/80 rule we recommend to evolve the application and add more security features in the following order:

1. The application should have a workflow that will pass the data through the firewall and DMZ zone.
2. Add a periodical refreshing of the passwords both for users and admins;
3. Create filters to avoid malicious code input to the text fields;
4. Add a controller that will be responsible for any mistakes, that are shown to the user;
5. Create a procedure of accessing to the applications data by the HR managers and software developers to make it authorized;
6. Add a two steps verification, for example with the passwords sent to the users on the attempt to login;
7. Bruce forcing (not allowing weak passwords).

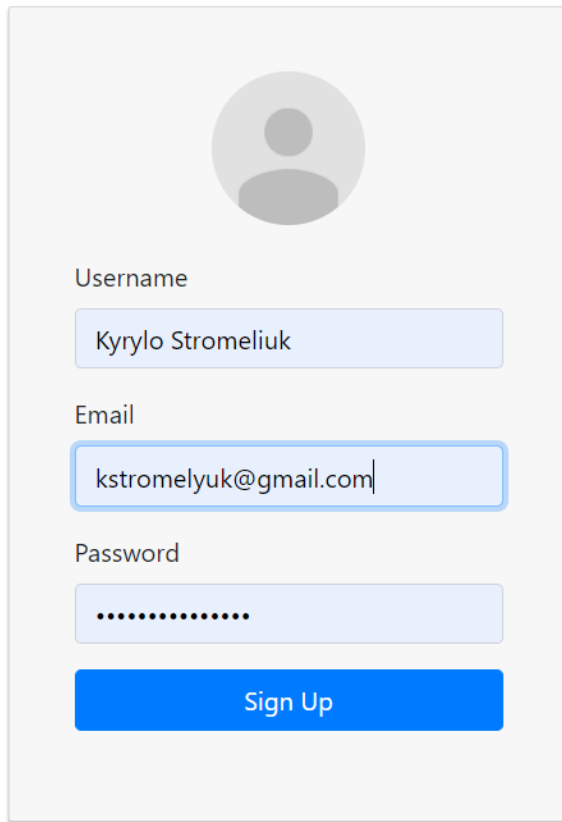
## **IMPLEMENTED SAFETY MEASURES**

Using the workflow of 20/80 rule our Samurai team has already implemented safety measures of:

1. Registration can be done with the email, that was provided by the (former) employee in advance to the authorized representative of the HR department;
2. Registration is provided with the token that can expire in 24 hours in case the user did not finish the registration. This token also helps to avoid hacker's attempts to catch user's information during the registration process as performs by the POST method;
3. Registration and login process does not provide any extra information that can help with attempts to search if the login or email already exists;
4. All the back end process is provided on the server side;
5. User's interface, login interface has no additional menu, images or other interface that can give any additional information to the hackers;
6. The roles of all users are implemented. The admin, HR and user have their own access level.

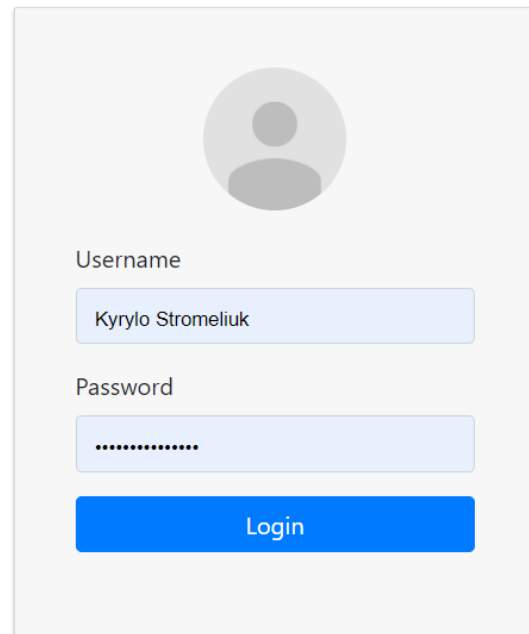
## THE EXAMPLES OF IMPLEMENTATION

General view of the registration page, nothing else existing on the page:



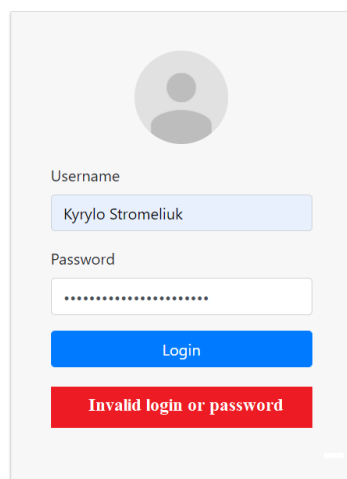
A registration form with a light gray background. At the top is a circular placeholder for a profile picture. Below it are three input fields: 'Username' with the text 'Kyrylo Stromeliuk', 'Email' with the text 'kstromelyuk@gmail.com', and 'Password' with masked characters. A blue 'Sign Up' button is at the bottom.

General view of the login page, nothing else existing on the page:



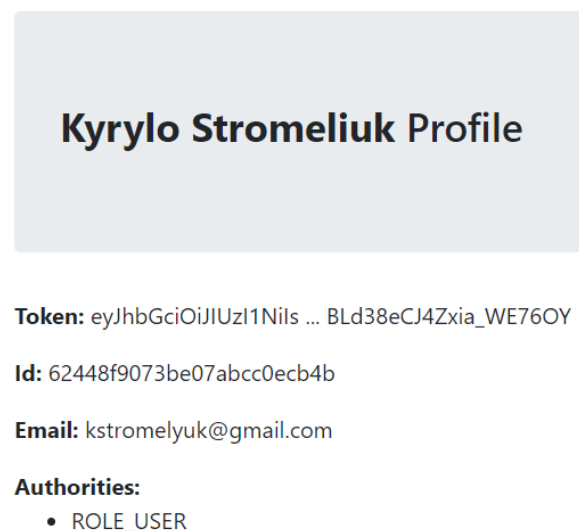
A login form with a light gray background. At the top is a circular placeholder for a profile picture. Below it are two input fields: 'Username' with the text 'Kyrylo Stromeliuk' and 'Password' with masked characters. A blue 'Login' button is at the bottom.

Not giving extra information regarding errors:



A login form with a light gray background. At the top is a circular placeholder for a profile picture. Below it are two input fields: 'Username' with the text 'Kyrylo Stromeliuk' and 'Password' with masked characters. A blue 'Login' button is at the bottom, and a red error message 'Invalid login or password' is displayed below it.

User's interface after the login, no other information except related to the user is given:



A user profile interface with a light gray background. At the top is a circular placeholder for a profile picture. Below it is the text 'Kyrylo Stromeliuk Profile'. Below this is a list of user information: 'Token: eyJhbGciOiJIUzI1NiIs ... Bld38eCj4Zxia\_WE76OY', 'Id: 62448f9073be07abcc0ecb4b', 'Email: kstromelyuk@gmail.com', and 'Authorities: ROLE\_USER'.

The code of the token, that helps to secure registration:

```
verifyToken = (req, res, next) => {
  let token = req.headers["x-access-token"];

  if (!token) {
    return res.status(403).send({ message: "No token provided!" });
  }

  jwt.verify(token, config.secret, (err, decoded) => {
    if (err) {
      return res.status(401).send({ message: "Unauthorized!" });
    }
    req.userId = decoded.id;
    next();
  });
};
```

The code. that manage output of the information according to the role ( user, admin, HR):

```
isAdmin = (req, res, next) => {
  User.findById(req.userId).exec((err, user) => {
    if (err) {
      res.status(500).send({ message: err });
      return;
    }

    Role.find(
      {
        _id: { $in: user.roles }
      },
      (err, roles) => {
        if (err) {
          res.status(500).send({ message: err });
          return;
        }
      }
    );
  });
};
```

Performing with the POST method:

```
module.exports = function(app) {
  app.use(function(req, res, next) {
    res.header(
      "Access-Control-Allow-Headers",
      "x-access-token, Origin, Content-Type, Accept"
    );
    next();
  });

  app.get("/api/admin/findAll", [authJwt.verifyToken], controller.findAll);

  app.post("/api/admin/exitEmail/create", [authJwt.verifyToken, authJwt.isAdmin], controller.create);
  app.post("/api/admin/exitEmail/update", [authJwt.verifyToken, authJwt.isAdmin], controller.update);
};
```

Token working schema:

