# Security Report for


## Booking platform

-Semester 3 Individual Project-

Plamen Peev

Student number: 4179080

- A1:2017-Injection

I could say that the risk is not completely applicable to the application, as only JPA named queries are used in this project.
String concatenation is not being used, because this is the main source of SQL injection vulnerabilities. If I would need more specific query, I would do it with a prepared statement (parametrized query) to separate the query and the provided data, therefore prevent malicious SQL queries in the parameters.

However, there could be a potential risk, because I am not using a WAF (Web application firewall) in my API to inspect incoming queries. As we cannot be sure how competent the attacker might be, we should use all resources possible to prevent an SQL Injection.

- A2:2017-Broken Authentication

The application is not yet completely secured, based on the standards of OWASP for this risk.

  - What is implemented:
    - JWT authentication on login (access and refresh tokens)
    - JWT authorization on API call
    - Hashing passwords before storing them to the database
    - Weak-password checks
  - What isn't implemented:
    - Multi-factor authentication
    - Credential recovery
    - Limit/delay failed logins
    - Log failed logging attempts/alert administrators

  As the project is yet under development, the above mentioned measures haven't been completed yet, therefore we can conclude this could be a potential risk for the application if we don't complete the unimplemented steps.

- A3:2017-Sensitive Data Exposure

The only sensitive data in the application is the passwords at this moment. For this, 'bcrypt' is used to hash the passwords before storing them in the database.

There is, however, a potential risk as the protocols are not encrypted. Therefore, the interaction between the server and the user interface is clearly visible and could be potentially used for malicious actions by an attacker who has remote access to the machine of the user.

- A4:2017-XML External Entities (XXE)

The risk is not applicable, as the only type of files being accepted are .jpeg and .png. SOAP is also not used, therefore XML upload won't be processed at all. SOAP is also not used in this project, so XXE attack is also prevented.

- A5:2017-Broken Access Control

API endpoints are restricted based on the role which is contained in the Jason Web Token. Example:

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    CustomAuthenticationFilter customAuthenticationFilter = new CustomAuthenticationFilter(authenticationManagerBean(), authenticationFailureHandler(), userService);
    customAuthenticationFilter.setFilterProcessesUrl("/api/login");
    http.csrf().disable();
    http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
    http.authorizeRequests().antMatchers( ...antPatterns: "/api/login/**","/api/register/**", "/api/token/refresh/**").permitAll();
    http.authorizeRequests().antMatchers(GET,  ...antPatterns: "/api/user/**").hasAnyAuthority( ...authorities: "ROLE_USER");
    http.authorizeRequests().antMatchers(GET,  ...antPatterns: "api/hotels-by-manager/**").hasAnyAuthority( ...authorities: "ROLE_HOTEL_MANAGER");
    http.authorizeRequests().antMatchers(PUT,  ...antPatterns: "api/hotel/update/**").hasAnyAuthority( ...authorities: "ROLE_HOTEL_MANAGER");
    http.authorizeRequests().antMatchers(POST,  ...antPatterns: "api/hotel/create/**").hasAnyAuthority( ...authorities: "ROLE_HOTEL_MANAGER");
    http.authorizeRequests().anyRequest().authenticated();
    http.addFilter(customAuthenticationFilter);
    http.addFilterBefore(new CustomAuthorizationFilter(), UsernamePasswordAuthenticationFilter.class);
```

Metadata and backup files are not present within the web roots.

Instead of CORS, a proxy is being used from the frontend.

By default, if the token doesn't contain the requested role for the API call, an error with status 403 is being send from the sever.

The only measurement which isn't implemented for this risk is the invalidation of the token on logout, but this will of course be implemented before deployment.

- A6:2017-Security Misconfiguration

For this risk, SonarQube is used to verify the effectiveness of the security configuration and report any problems.

There aren't any unnecessary features or components in this project.

This risk is not applicable for the application.

- A7:2017-Cross-Site Scripting (XSS)

React JS is automatically escaping XSS by design.
Reflected, Stored and DOM XSS attacks are not possible in the application, due both to React and the functions of the frontend.

- ## A8:2017-Insecure Deserialization

Serialization/Deserialization is only used for the logged in user object. Even if the attacker changes the serialized object, this will not result in any security breach, as the valuable data with all the permissions and session expiration date is stored within the Jason Web Token. Everything except the token is simply used for reading purposes only, and it doesn't contain any valuable information for the authentication or authorization of the users.

- ## A9:2017-Using Components with Known Vulnerabilities

Choosing the libraries at the beginning of the project, I assured that all of them are widely used and actively improved. I've researched almost all of the libraries/packages I am using both at the frontend and the backend.

Unused dependencies are removed, and only official sources are obtained from secure links.

Monitoring can be improved by setting up an automated patch management, that can help simplify the process and make sure everything is up to date.

- ## A10:2017-Insufficient Logging & Monitoring

Messages/actions that are logged are strictly modified so that they don't leak any vulnerable information, but they are not stored in any form, and are simply for the developer/or user's information. A potential improvement here could be storing and monitoring of the logged values, so that the developers can be alerted for a potential attempt or breach.