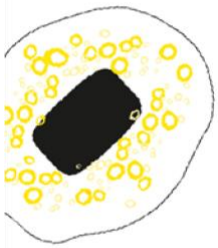
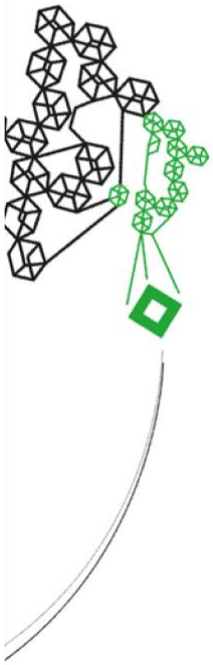


Data and Information 2023

'Security analysis'

Version	1.0
Date	26-05-2023
Status	Final
Team	EarnIT 4
Names	(3077489) Gracjan Chmielnicki – BIT (2993074) Tom Hansult - BIT (2997894) Dirck Mulder - TCS (2957566) Pepijn Meijer - TCS (2920352) Thomas Brants - TCS (2957868) Razvan Stefan - TCS



1 Security analysis

Ensuring the security of a website is of great importance to safeguard against malicious activities and unauthorized practices. The following will discuss the security measures and mechanisms that this application implements.

During user authentication, a JSON Web Token (JWT) is generated for each user upon sign-in. This JWT contains the user's unique identifier and additional information, such as the creation timestamp. To establish the authenticity of a JWT, the server signs it with a private key, enabling subsequent validation. Validating the JWT allows the server to confirm the authentication of the user. Subsequently, the token is returned to the user. If the user attempts to access any page or make requests without presenting this token, a 401 unauthorized error is returned.

All pages within the platform are categorized into three distinct groups: staff, student, and company. When a user requests access to a specific page, the server verifies the validity of the JWT and checks if the user is authorized to access the corresponding category. If authorization is denied, a 403 forbidden response is issued. This same authentication and authorization process applies to API requests as well.

In addition to authentication and authorization, the platform implements further measures to safeguards against potential malicious activities. Three identified problems have been addressed accordingly. Firstly, in the event of a database leak, the platform ensures the safety of user passwords by implementing a hashing algorithm. The chosen hashing algorithm is bcrypt with 12 rounds, which is based on the blowfish cipher. The blowfish cipher is a block cipher that employs dynamic key changes, thereby reducing the effectiveness of brute force attacks. Further information on this can be found at https://auth0.com/blog/hashing-in-action-understanding-bcrypt/.

Secondly, to prevent SQL injection vulnerabilities, the platform utilizes prepared statements. Prepared statements serve to separate code and data, preventing unintended execution of SQL code by the server that may result from the mixing of code and data.

Lastly, the platform addresses cross-site scripting (XSS) risks by implementing serverside text escaping for user-provided inputs. Additionally, on the client side, the application employs the use of "innerText" instead of "innerHTML," thus mitigating the potential execution of HTML code even in the event of bypassing the server-side text escaping measures.

2 Software testing

To properly test the software developed throughout this project, we employed both unit and user-story testing. The following will shortly discuss the purpose of the testing method, how it was realized and how it helped identifying problems in the system.

2.1 Unit testing approach

The unit testing focused on verifying the functionality and behavior of the DAO classes. To approach unit testing, the JUnit testing framework was used. The unit tests were designed to cover various scenarios to ensure that the DAOs work as intended.

The unit tests for the DAO classes aimed to verify the following aspects:

1. Correct execution and results of database queries: The tests validated that the SQL queries constructed by the DAO classes were correct and returned the expected results when interacting with the database.
2. Proper handling of exceptions: The tests verified that exceptions, such as SQLExceptions, were appropriately handled by the DAO classes, ensuring proper error handling and preventing any unexpected behavior.
3. Integration with other classes: The tests ensured that the DAO classes interacted correctly with the database and other classes, by setting up necessary dependencies and validating the interactions.
4. Business logic and data manipulation: The tests covered methods and functionalities of the DAO classes, verifying that the business logic and data manipulation operations were executed correctly, without any inconsistencies or unexpected behavior.

Throughout the unit testing process, test cases were designed to cover a variety of scenarios. We used an embedded PostgreSQL server for the unit tests, this enabled running tests that also write to the database without affecting the actual database of the platform.

Unit tests helped uncover hidden bugs, validate the correctness of the code, and ensure robustness.

2.2 User story testing approach

Mainly, user story testing was performed to validate the system behavior from an end-user perspective. This testing approach focused on verifying that the application fulfilled the intended functionalities and requirements, as defined by the user stories.

The user story testing process involved the following steps:

1. Interaction with the UI: Testers interacted with the user interface of the web application, performing actions and operations as a typical user would. This included navigating through different screens, entering input data, and triggering various functionalities.
2. Validating expected outputs: After performing actions, the testers compared the outputs and results against the expected behavior defined in the user stories. This verification ensured that the application responded correctly and produced the intended outcomes.
3. Error code printing: The application was configured to generate error codes or messages when encountering errors or exceptions. Testers monitored these error codes during user story testing to identify any issues or unexpected behaviors.
4. Monitoring the database: Testers closely monitored the database to ensure that the application correctly stored and retrieved data, verifiying that data modifications were processed correctly in the database.

Combining these steps provided a comprehensive assessment of the application's functionality and usability. User story testing was the primary testing tool used.

Overall, the combination of unit testing for code-level verification and user story testing for end-user testing ensures a robust and reliable application, various edge cases were considered in this process, enabling early bug detection, and improving the overall quality of the software.