Bee Design Award Competition System - Security Report

Global Design

V1.0

27th June 2021

WONG WEI JIAN & ADRIEL GOH WEI EN

Terms of Use

Document Log

Title	Version	Release Type	Author	Date
Competition System - Security Report	0.1	Draft	Wong Wei Jian	4th June 2021
Competition System - Security Report	0.2	Draft	Adriel Goh Wei En & Wong Wei Jian	11th June 2021
Competition System - Security Report	0.3	Draft	Adriel Goh Wei En & Wong Wei Jian	15th June 2021
Competition System - Security Report	0.4	Draft	Adriel Goh Wei En & Wong Wei Jian	17th June 2021
Competition System - Security Report	0.5	Draft	Adriel Goh Wei En & Wong Wei Jian	19th June 2021
Competition System - Security Report	0.6	Draft	Adriel Goh Wei En & Wong Wei Jian	22nd June 2021
Competition System - Security Report	0.7	Draft	Adriel Goh Wei En & Wong Wei Jian	24th June 2021
Competition System - Security Report	0.8	Draft	Adriel Goh Wei En & Wong Wei Jian	26th June 2021
Competition System - Security Report	1.0	Final Release	Adriel Goh Wei En & Wong Wei Jian	27th June 2021

Table of Contents

1 Executive Summary

1.1. Overview

Global Design approached Singapore Polytechnic's Enterprise System Development (ESDE) Students to conduct a code check of its Bee Design Award Competition System. The purpose of this engagement was to get the ESDE Team to evaluate the security of the system to validate its security mechanisms and identify possible vulnerabilities. The possible vulnerabilities found will be documented and with remedial advice provided.

This report also details the scope of the testing conducted, with all significant findings with detailed remedial advice documented. This summary has been written such that it provides a non-technical audience with the key findings and associate it to the impact on the business.

1.2. Our Team

Role	Name
Project Leader	Wong Wei Jian (P1908365)
Project Member	Adriel Goh Wei En (P1907872)

2 Key Findings

2.1. Introduction

During our initial scan of the codes, we found that:

(a) The program uses multiple callbacks (refer to Terminologies) when performing an action. This could lead to the system hanging like in the login page where if someone spams the login, the system will end up hanging after a few minutes. To avoid the usage of nested callbacks, there could be implementation of using Asynchronous functions or JavaScript promises.

(b) There were inconsistent JSON response data in several JavaScript files. To improve readability as well as ensure that the entire system is more streamlined and consistent, the JSON returns should be standardized to one style rather than multiple styles which indicate that there are more than 1 developer working the codes. Some JSON returns are put directly inline, while some are crafted into a variable before passing the variable.

<table>
<tr><th>Original Codes</th><th>Fixed Codes</th></tr>
<tr><td colspan="2">backEnd > src > controllers > authController.js
(exports.processLogin)</td></tr>
<tr><td>

```
exports.processLogin = (req, res, next) => {

    let email = req.body.email;
    let password = req.body.password;
    try {
        auth.authenticate(email, function (error, results) {
            if (error) {
                let message = 'Credentials are not valid.';
                //return res.status(500).json({ message: message });
                //If the following statement replaces the above statement
                //to return a JSON response to the client, the SQLMap or
                //any attacker (who relies on the error) will be very happy
                //because they relies a lot on SQL error for designing how to do
                //attack and anticipate how much "rewards" after the effort.
                //Rewards such as sabotage (seriously damage the data in database),
                return res.status(500).json({ message: error });
            } else {
                if (results.length == 1) {
                    if ((password == null) || (results[0] == null)) {
                        return res.status(500).json({ message: 'login failed' });
                    }
                    if (bcrypt.compareSync(password, results[0].user_password) == true) {

                        let data = {
                            user_id: results[0].user_id,
                            role_name: results[0].role_name,
                            token: jwt.sign({ id: results[0].user_id }, config.JWTKey, {
                                expiresIn: 86400 //Expires in 24 hrs
                            })
                        }; //End of data variable setup

                        return res.status(200).json(data);
                    } else {
                        return res.status(500).json({ message: error });
                    } //End of password comparison with the retrieved decoded password.
                } //End of checking if there are returned SQL results

            }

        })

    } catch (error) {
        return res.status(500).json({ message: error });
    } //end of try


};
```

As seen in the above code (highlighted yellow), you can see that throughout the page, there is inconsistent JSON object returns… Some returning the error directly to the frontEnd, some returning customized message to the frontEnd, and some returning data to the frontEnd

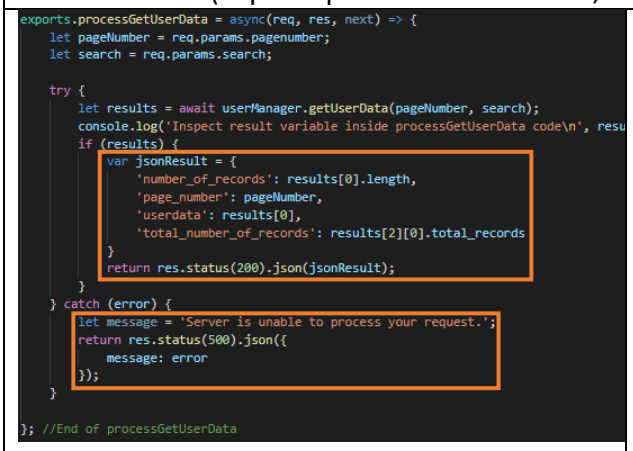
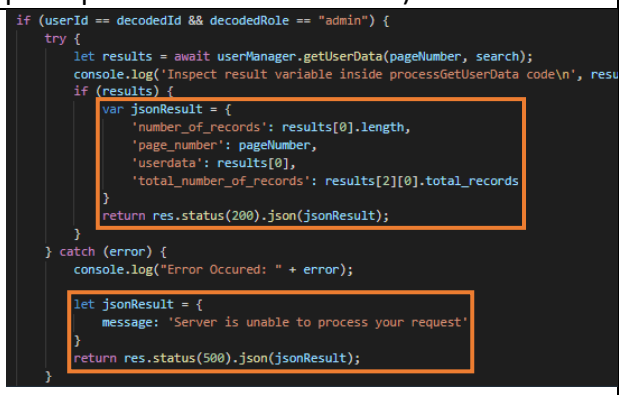
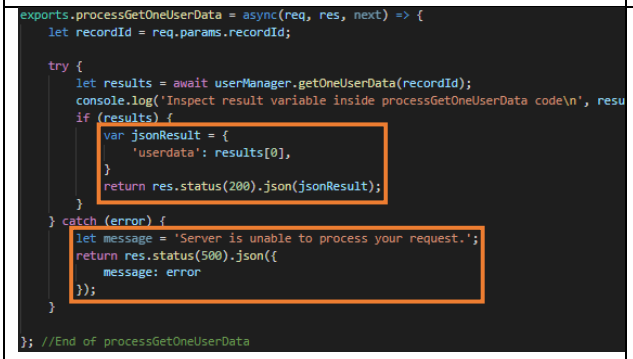
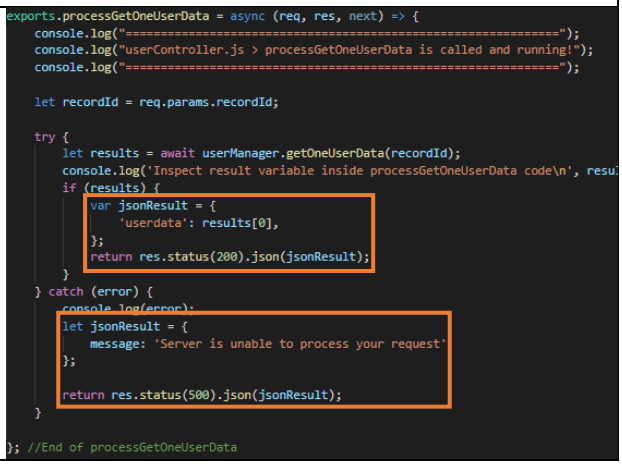
</td><td>

```
try {
    auth.authenticate(email, function (error, results) {
        if (error) {
            let jsonResult = {
                message: 'Credentials are not valid'
            };
            return res.status(500).json(jsonResult);

        } else {
            if (results.length == 1) {
                if ((password == null) || (results[0] == null)) {
                    let jsonResult = {
                        message: 'Login Failed'
                    };
                    return res.status(500).json(jsonResult);
                }

                if (bcrypt.compareSync(password, results[0].user_password) == true) {
                    let jsonResult = {
                        user_id: results[0].user_id,
                        role_name: results[0].role_name,
                        token: jwt.sign({ id: results[0].user_id, uRole: results[0].role_name }, config.JWTKey, {
                            expiresIn: 7200 //Expires in 2 hrs
                        })
                    }; //End of jsonResult variable setup

                    return res.status(200).json(jsonResult);
                } else {
                    let jsonResult = {
                        message: 'Login has failed.'
                    }
                    return res.status(500).json(jsonResult);
                } //End of password comparison with the retrieved decoded password.
            } //End of checking if there are returned SQL results
        }

    })

} catch (error) {
    let jsonResult = {
        message: 'An error has occured!'
    }
    return res.status(500).json(jsonResult);
} //end of try
```

To have a more consistent, standardized and streamlined view, we have decided to separate the returns into a variable (JSON Object) declaration first, before sending the variable (JSON Object) back to the frontEnd

This allows more readability of the codes, as well as more clarity on what you are sending back to the frontEnd.

</td></tr>
</table>

Original Codes	Fixed Codes
backEnd > src > controllers > userController.js (exports.processGetUserData, exports.processGetOneUserData)	
 ``` exports.processGetUserData = async(req, res, next) => { let pageNumber = req.params.pagenumber; let search = req.params.search; try { let results = await userManager.getUserData(pageNumber, search); console.log('Inspect result variable inside processGetUserData code\n', resu if (results) { var jsonResult = { 'number_of_records': results[0].length, 'page_number': pageNumber, 'userdata': results[0], 'total_number_of_records': results[2][0].total_records } return res.status(200).json(jsonResult); } } catch (error) { let message = 'Server is unable to process your request.'; return res.status(500).json({ message: error }); } }; //End of processGetUserData ```	 ``` if (userId == decodedId && decodedRole == "admin") { try { let results = await userManager.getUserData(pageNumber, search); console.log('Inspect result variable inside processGetUserData code\n', resu if (results) { var jsonResult = { 'number_of_records': results[0].length, 'page_number': pageNumber, 'userdata': results[0], 'total_number_of_records': results[2][0].total_records } return res.status(200).json(jsonResult); } } catch (error) { console.log("Error Occured: " + error); let jsonResult = { message: 'Server is unable to process your request' } return res.status(500).json(jsonResult); } ```
 ``` exports.processGetOneUserData = async(req, res, next) => { let recordId = req.params.recordId; try { let results = await userManager.getOneUserData(recordId); console.log('Inspect result variable inside processGetOneUserData code\n', resu if (results) { var jsonResult = { 'userdata': results[0], } return res.status(200).json(jsonResult); } } catch (error) { let message = 'Server is unable to process your request.'; return res.status(500).json({ message: error }); } }; //End of processGetOneUserData ```	 ``` exports.processGetOneUserData = async (req, res, next) => { console.log("=="); console.log("userController.js > processGetOneUserData is called and running!"); console.log("=="); let recordId = req.params.recordId; try { let results = await userManager.getOneUserData(recordId); console.log('Inspect result variable inside processGetOneUserData code\n', resu if (results) { var jsonResult = { 'userdata': results[0], }; return res.status(200).json(jsonResult); } } catch (error) { console.log(error); let jsonResult = { message: 'Server is unable to process your request' }; return res.status(500).json(jsonResult); } }; //End of processGetOneUserData ```

2.2. Terminologies

Below are the terminologies that our groups will be using for the following report. These explanations will be able to guide you into understanding the different terms that professionals use for doing security checks for the entire system:

Terminologies	Meanings
Callbacks / Callback Functions	Callbacks / Callback Functions is known as a "Call-After" function whereby the other function is passed as a parameter in the original function, and the other function is expected to call back the original function once it is complete.
CRUD	CRUD or Create, Retrieve, Update and Delete, is a terminology or shortcut used by developers to modify the database.
Query Statement	Query statements are statements that are sent to the database to retrieve any data based on the statements sent, from the database.
SQL Injection	SQL is the type of database that the companies use to store their application's data in. SQL Injection is where an attacker can send an input into the database (directly or indirectly) to tamper with the data in the database.
Authentication	Authentication is the action of validating the user's identity in the system before it can perform any action.
Authorization	Authorization is the action of checking the user's authorization level within a system, determining the type of access the user is granted.
Input Validation	Input validation is the code that exists in the client's side (Front-End) that will do the checking of the inputs that the user enters to ensure that it met the requirements set out by the developer or owner.
Output Sanitization	Output sanitizations is the code that exists in the developer's side (Back-End) that will, based on the outputs of the system (whether success, failure, and dependent on the HTTP Status Code), sends the appropriate response back to the user without revealing too much details about the outputs
Broken Access Control	Broken Access Control is when a user can act outside their intended permissions and perform actions or functions that are not intended for them.
Stored Cross-site Scripting (XSS)	Cross-site Scripting (XSS) is when a piece of JavaScript code, typically a script, is inserted and stored in the database and runs when that particular data is queried

Regular Expressions	Regular expressions (Regex) are a “String-of-Conditions” that has special meanings when processed as a regular expression. This “String-of-Conditions” processes text fields to determine if the text matches the “String-of-Conditions”. If it does not match the conditions set out, the text is invalid, otherwise it will be valid.

2.3. Tools List

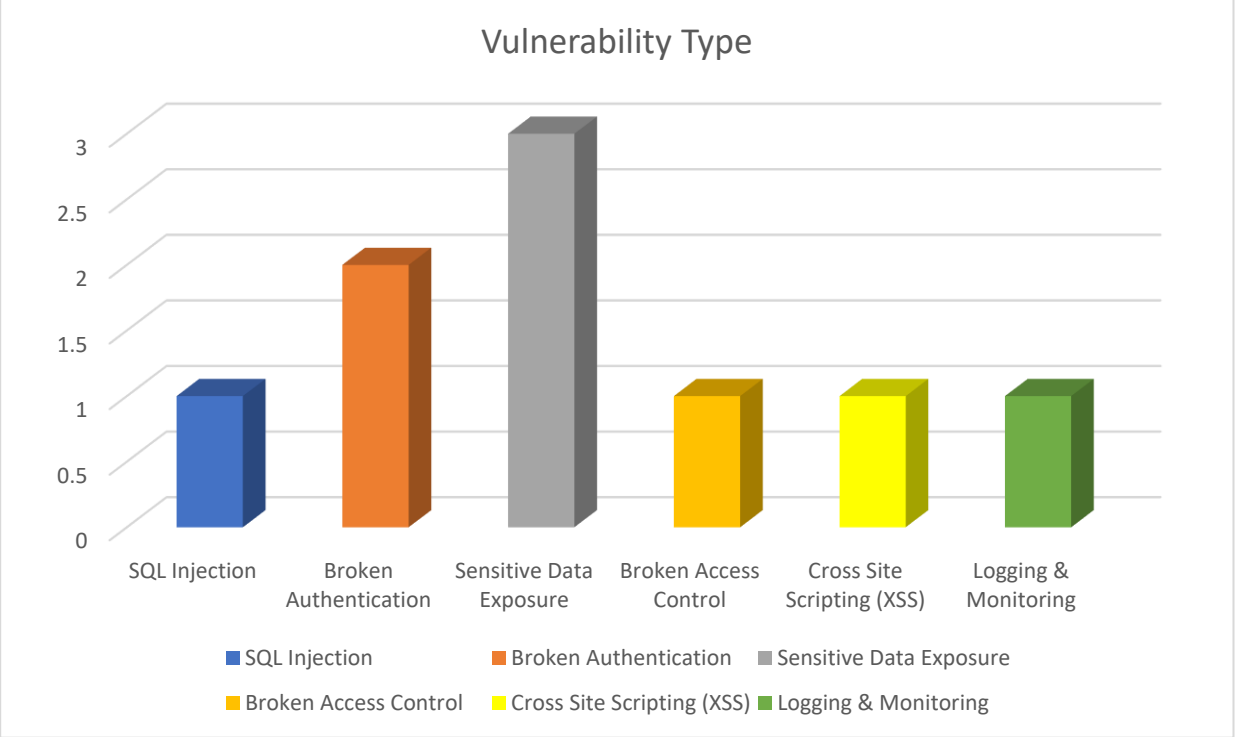
Below are the tools and Integrated Development Environment (IDE) that we have used to do the testing of the Bee Design Award Competition System. Most of these tools are used by professionals in the industry to test out different aspects of their systems.

Tools Used	Targeted Test Area
Burp Suite	Used to intercept the HTTP requests and/or any traffic made by the users
Postman	Used to target the Back-End server to see the responses that are received back from the server
Visual Studio Code (VSC)	This is one of the many professional Integrated Development Environment (IDE) that professionals use to write their codes in. It provides a nice, formatted view of the codes meant for developers to develop their codes in

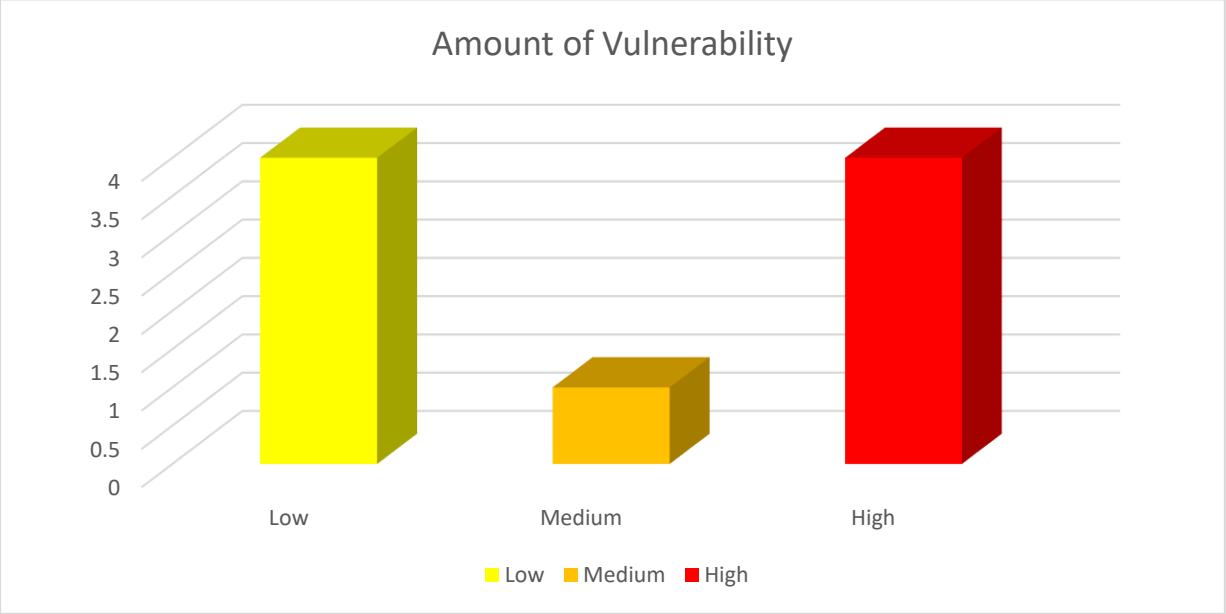
2.4. Key Issues

Through our thorough examination of Bee Design Award Competition System’s security, our team has found that there are inadequate amounts of security tools and features implemented and as such, has led to numerous vulnerabilities found in the code.

Our team has categorized the vulnerabilities found to 6 different types and 3 levels of impact it will have on the system:



Graph above shows the different vulnerability types our team has found.



Graph above shows the different vulnerability impacts it will have on the system.

In short, we have found that the system is prone to easy and simple XSS and SQL injection that makes the system very vulnerable. It is possible for a novice hacker to hack into and attack the system along with the database without much difficulty. Due to the lack to input validation, users can input malicious code and queries to tamper with the backend database as well as inject scripts into the database, harming the whole integrity of the system as well as potentially exposing danger to the administrators of the system.

Additionally, there has been little to no user authentication and authorization when functions are called from the website, leaving all pages accessible by anyone and with any authority level.

The following highlights the areas that are of upmost importance when dealing with web security as well as what the system is prone to be attacked by if the system is targeted.

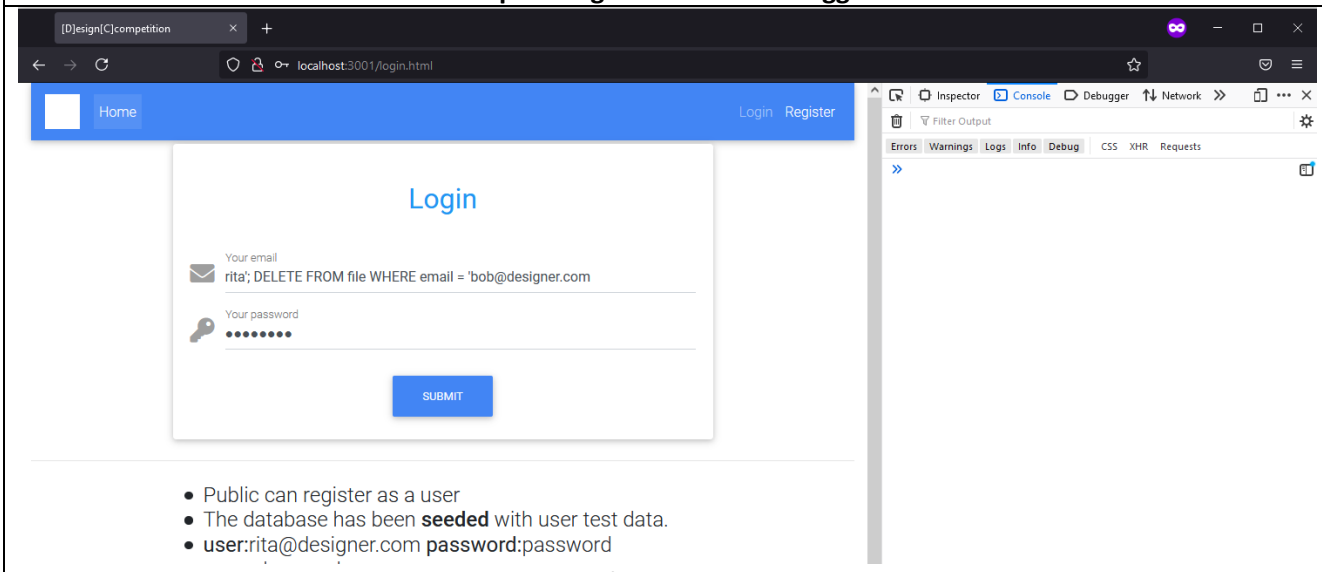
Key Issues	Issue Area	Impact
A significant number of query statements were found to be prone to SQL Injection attacks. Any attacker who attacks this system will be able to gain access to the database, modify the database as well as retrieve and delete data (CRUD) from the database.	SQL Injection	High
A significant number of pages were found to have no user authorization and authentication. This means that low-privileged users can gain access to administrator pages of the website. Besides that, a significant number of functions were also found to have no authorization checks leading to low-privileged users being able to run administrative functions like editing a user's permission level.	Broken Authentication & Access Control	High
A significant number of functions were assessed to be vulnerable to Cross-site Scripting (XSS) attacks. This allows attackers to steal information by targeting at users rather than the system itself, and hence able to portray themselves as the targeted user to run commands within the system	XSS	High

3 Detailed Findings

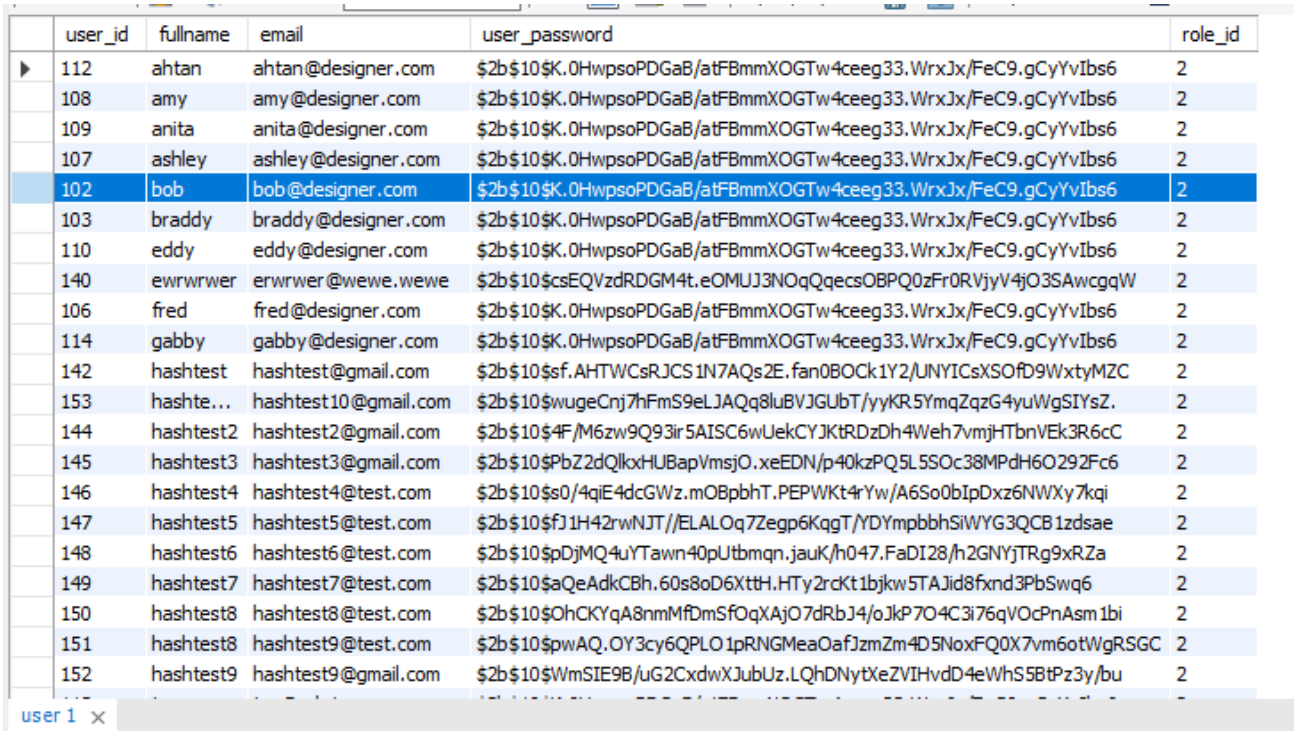
Vulnerability 1 – SQL Injection

Vulnerability ID	Impact	Ease of Exploit	Recommendations
V-01 SQL Injection	When any user uploads a file into the database, they can craft and insert any SQL statements into the text field and when submitted, it will tamper the data stored. This will result in having a possible database exposure. Impact – High	In the login.html page, the email input is vulnerable to SQL injection. When a SQL statement is inserted, it can tamper with the database and delete users as seen in the demo screenshots on page12. Ease of exploit - Easy	By simply changing the SQL statement in fileService.js to a prepared statement, we can prevent SQL injection as the user input will be handled as a string for the parameter instead of a SQL statement. In the code changes screenshots provided below**,** we have changed the "${}" to a "?" placeholder. Hence, when the values are called it prevents a SQL injection attack in the fileService.js Furthermore, the user input should be filtered as an attacker could still attempt a SQL injection attack.

Demo Screenshots (Before)
- Uploading of New Files & Loggin In



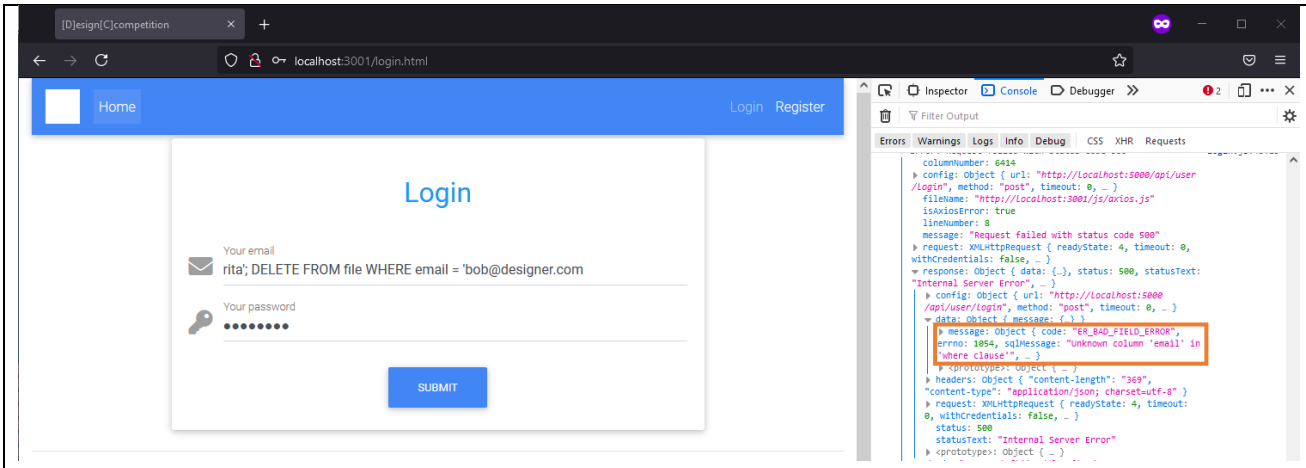
The above screenshot is used to test to see if there are any SQL response received based on the input



user_id	fullname	email	user_password	role_id
112	ahtan	ahtan@designer.com	$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg33.WrxJx/FeC9.gCyYvIbs6	2
108	amy	amy@designer.com	$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg33.WrxJx/FeC9.gCyYvIbs6	2
109	anita	anita@designer.com	$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg33.WrxJx/FeC9.gCyYvIbs6	2
107	ashley	ashley@designer.com	$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg33.WrxJx/FeC9.gCyYvIbs6	2
102	bob	bob@designer.com	$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg33.WrxJx/FeC9.gCyYvIbs6	2
103	braddy	braddy@designer.com	$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg33.WrxJx/FeC9.gCyYvIbs6	2
110	eddy	eddy@designer.com	$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg33.WrxJx/FeC9.gCyYvIbs6	2
140	ewrwrwer	erwrwer@wewe.wewe	$2b$10$csEQVzdRDGM4t.eOMUJ3NOqQqecsOBPQ0zFr0RVjyV4jO3SAwcgqW	2
106	fred	fred@designer.com	$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg33.WrxJx/FeC9.gCyYvIbs6	2
114	gabby	gabby@designer.com	$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg33.WrxJx/FeC9.gCyYvIbs6	2
142	hashtest	hashtest@gmail.com	$2b$10$sf.AHTWCsRJCS1N7AQs2E.fan0BOCk1Y2/UNYICsXSOfD9WxtyMZC	2
153	hashte...	hashtest10@gmail.com	$2b$10$wugeCnj7hFmS9eLJAQq8luBVJGUbT/yyKR5YmqZqzG4yuWgSIYsZ.	2
144	hashtest2	hashtest2@gmail.com	$2b$10$4F/M6zw9Q93ir5AISC6wUekCYJKtRDzDh4Weh7vmjHTbnVEk3R6cC	2
145	hashtest3	hashtest3@gmail.com	$2b$10$PbZ2dQlkxHUBapVmsjO.xeEDN/p40kzPQ5L5SOc38MPdH6O292Fc6	2
146	hashtest4	hashtest4@test.com	$2b$10$s0/4qiE4dcGWz.mOBpbhT.PEPWKt4rYw/A6So0bIpDxz6NWXy7kqi	2
147	hashtest5	hashtest5@test.com	$2b$10$fJ1H42rwNJT//ELALOq7Zegp6KqgT/YDYmpbbhSiWYG3QCB1zdsae	2
148	hashtest6	hashtest6@test.com	$2b$10$pDjMQ4uYTawn40pUtbmqn.jauK/h047.FaDI28/h2GNYjTRg9xRZa	2
149	hashtest7	hashtest7@test.com	$2b$10$aQeAdkCBh.60s8oD6XttH.HTy2rcKt1bjkw5TAJid8fxnd3PbSwq6	2
150	hashtest8	hashtest8@test.com	$2b$10$OhCKYqA8nmMfDmSfOqXAjO7dRbJ4/oJkP7O4C3i76qVOcPnAsm1bi	2
151	hashtest8	hashtest9@test.com	$2b$10$pwAQ.OY3cy6QPLO1pRNGMeaOafJzmZm4D5NoxFQ0X7vm6otWgRSGC	2
152	hashtest9	hashtest9@gmail.com	$2b$10$WmSIE9B/uG2CxdwXJubUz.LQhDNytXeZVIHvdD4eWhS5BtPz3y/bu	2

user 1

The above screenshot shows the original data inside the user table, and to show that the user “bob” (user_id: 102) is still in the table

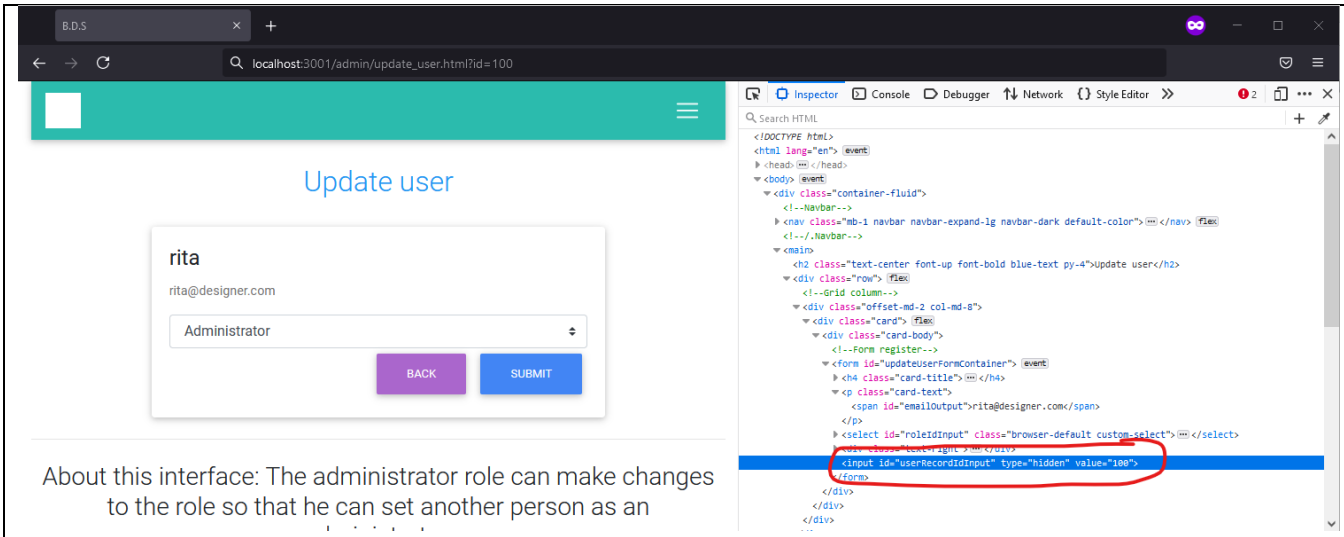


The screenshot above shows that the SQL statement was successfully ran in the database as it gives a SQL Error message in response. This shows that SQL injection was successful

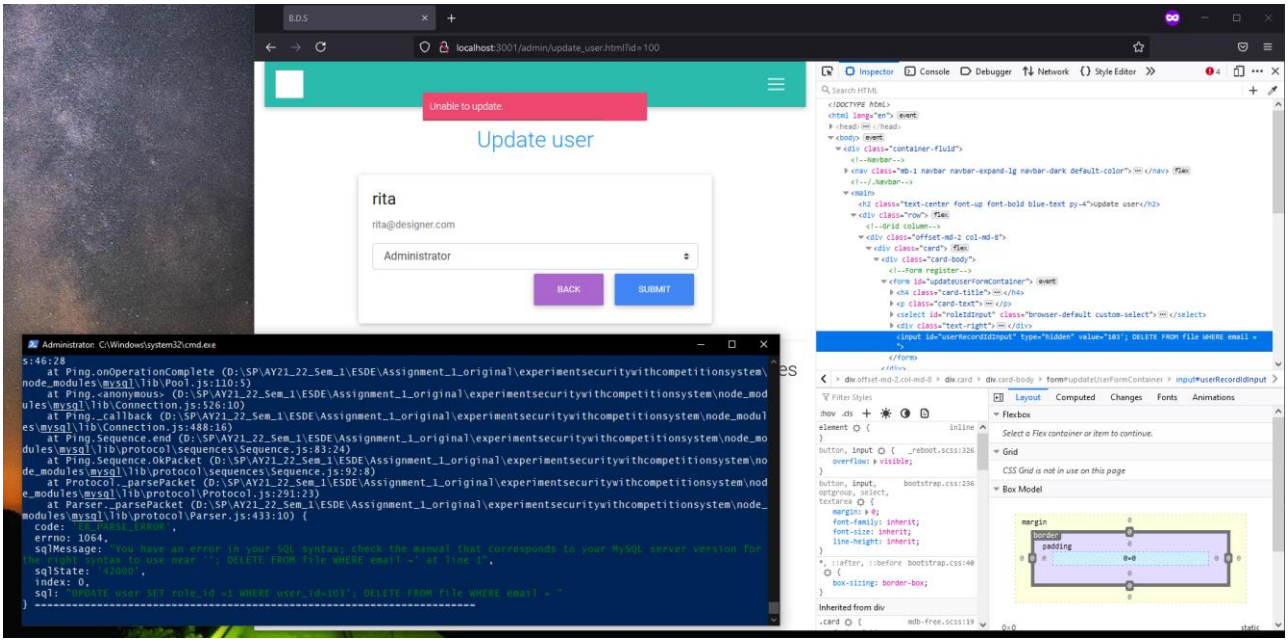
Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

user_id	fullname	email	user_password	role_id
100	rita	rita@designer.com	$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg33.WrxJx/FeC9.gCyYvIbs6	2
101	robert	robert@admin.com	$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg33.WrxJx/FeC9.gCyYvIbs6	1
103	braddy	braddy@designer.com	$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg33.WrxJx/FeC9.gCyYvIbs6	2
104	josh	josh@designer.com	$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg33.WrxJx/FeC9.gCyYvIbs6	2
105	john	john@designer.com	$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg33.WrxJx/FeC9.gCyYvIbs6	2
106	fred	fred@designer.com	$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg33.WrxJx/FeC9.gCyYvIbs6	2
107	ashley	ashley@designer.com	$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg33.WrxJx/FeC9.gCyYvIbs6	2
108	amy	amy@designer.com	$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg33.WrxJx/FeC9.gCyYvIbs6	2
109	anita	anita@designer.com	$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg33.WrxJx/FeC9.gCyYvIbs6	2
110	eddy	eddy@designer.com	$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg33.WrxJx/FeC9.gCyYvIbs6	2
111	larry	larry@designer.com	$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg33.WrxJx/FeC9.gCyYvIbs6	2
112	ahtan	ahtan@designer.com	$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg33.WrxJx/FeC9.gCyYvIbs6	2
113	joe	joe@admin.com	$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg33.WrxJx/FeC9.gCyYvIbs6	2
114	gabby	gabby@designer.com	$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg33.WrxJx/FeC9.gCyYvIbs6	2
140	ewrwrwer	erwrwer@wewe.wewe	$2b$10$csEQVzdRDGM4t.eOMUJ3NOqQqecsOBPQ0zFr0RVjyV4jO3SAwcgqW	2
141	Test	test@test.com	$2b$10$oEodoR2DY4q4O6Jdb2lsyu.FakyL33bTTZbkfkVkr44dVZvuvCelW	2
142	hashtest	hashtest@gmail.com	$2b$10$sf.AHTWCsRJCS1N7AQs2E.fan0BOCk1Y2/UNYICsXSOfD9WxtyMZC	2
144	hashtest2	hashtest2@gmail.com	$2b$10$4F/M6zw9Q93ir5AISC6wUekCYJKtRDzDh4Weh7vmjHTbnVEk3R6cC	2
145	hashtest3	hashtest3@gmail.com	$2b$10$PbZ2dQlkxHUBapVmsjO.xeEDN/p40kzPQ5L5SOc38MPdH6O292Fc6	2
146	hashtest4	hashtest4@test.com	$2b$10$s0/4qiE4dcGWz.mOBpbhT.PEPWKt4rYw/A6So0bIpDxz6NWXy7kqi	2
147	hashtest5	hashtest5@test.com	$2b$10$fJ1H42rwNJT//ELALOq7Zegp6KqgT/YDYmpbbhSiWYG3QCB1zdsae	2

After editing the SQL statement, based on the response in the previous picture, the screenshot above shows that the SQL statement was successfully ran in the database and that SQL Injection was successful as the user “bob” (user_id: 102) was deleted from the table



For updating of a user inside the /admin/update_user.html page, Changing the ID number inside the HTML document's hidden input, as shown above, will allow an SQL Injection to take place



In the screenshot above, you could see that there is a SQL response from the input, this shows that the above HTML document's hidden input is susceptible to an SQL Injection

Preventing Future Vulnerabilities

To prevent future vulnerability, please make use of the parameterized query so that you can prevent any future vulnerabilities due to SQL Injection.
The codes below are still susceptible to SQL injection if left untouched and as such, our team has taken measures to prevent SQL injection from occurring.

Original Codes	Fixed Codes
backEnd > src > services > authService.js (module.exports.authenticate)	

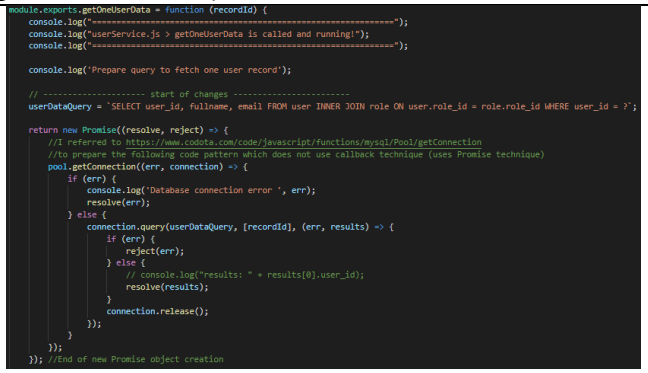
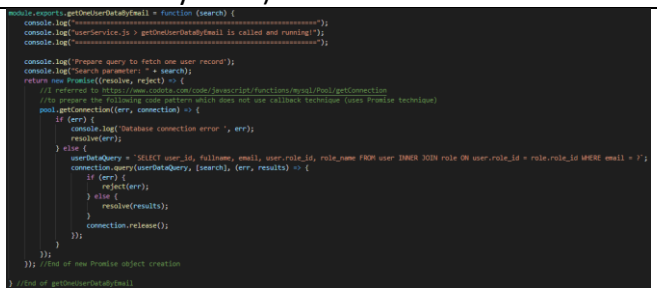
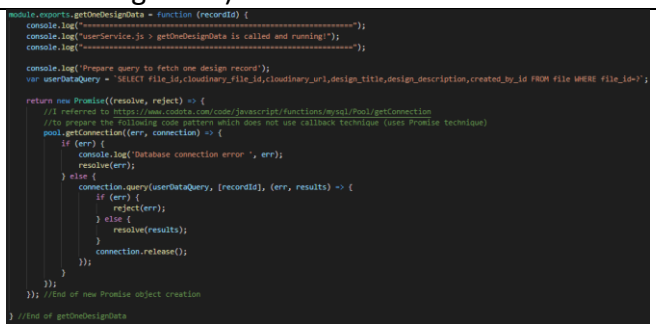
```
module.exports.authenticate = (email, callback) => {

    pool.getConnection((err, connection) => {
        if (err) {
            if (err) throw err;

        } else {
            try {
                connection.query(`SELECT user.user_id, fullname, email, user_password, role_name, user.role_id
                   FROM user INNER JOIN role ON user.role_id=role.role_id AND email='${email}'`, {}, (err, rows) => {
                    if (err) {
                        if (err) return callback(err, null);

                    } else {
                        if (rows.length == 1) {
                            console.log(rows);
                            return callback(null, rows);

                        } else {

                            return callback('Login has failed', null);
                        }
                    }
                    connection.release();

                });
            } catch (error) {
                return callback(error, null);;
            }
        }
    }); //End of getConnection

} //End of authenticate
```

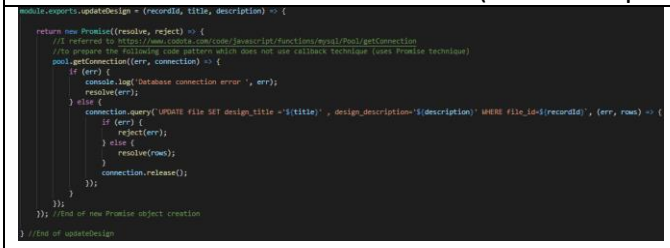
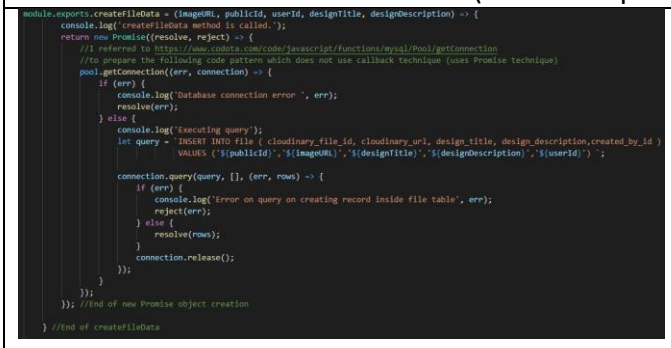
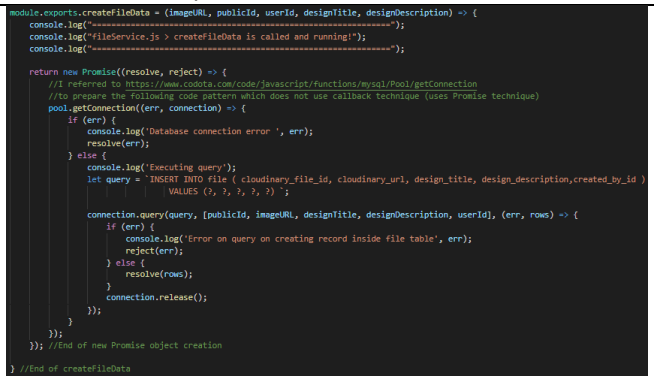
```
module.exports.authenticate = (email, callback) => {
    console.log("============================================================");
    console.log("authService.js > authenticate is called and running!");
    console.log("============================================================");

    pool.getConnection((err, connection) => {
        if (err) {
            if (err) throw err;

        } else {
            try {
                connection.query(`SELECT user.user_id, fullname, email, user_password, role_name, user.role_id
                    FROM user INNER JOIN role ON user.role_id=role.role_id AND email=?`, [email], (err, rows) => {
                    if (err) {
                        if (err) return callback(err, null);

                    } else {
                        if (rows.length == 1) {
                            console.log(rows);
                            return callback(null, rows);

                        } else {
                            return callback('Login has failed', null);
                        }
                    }
                    connection.release();

                });
            } catch (error) {
                return callback(error, null);;
            }
        }
    }); //End of getConnection

} //End of authenticate
```

backEnd > src > services > userService.js (module.exports.updateUser)	

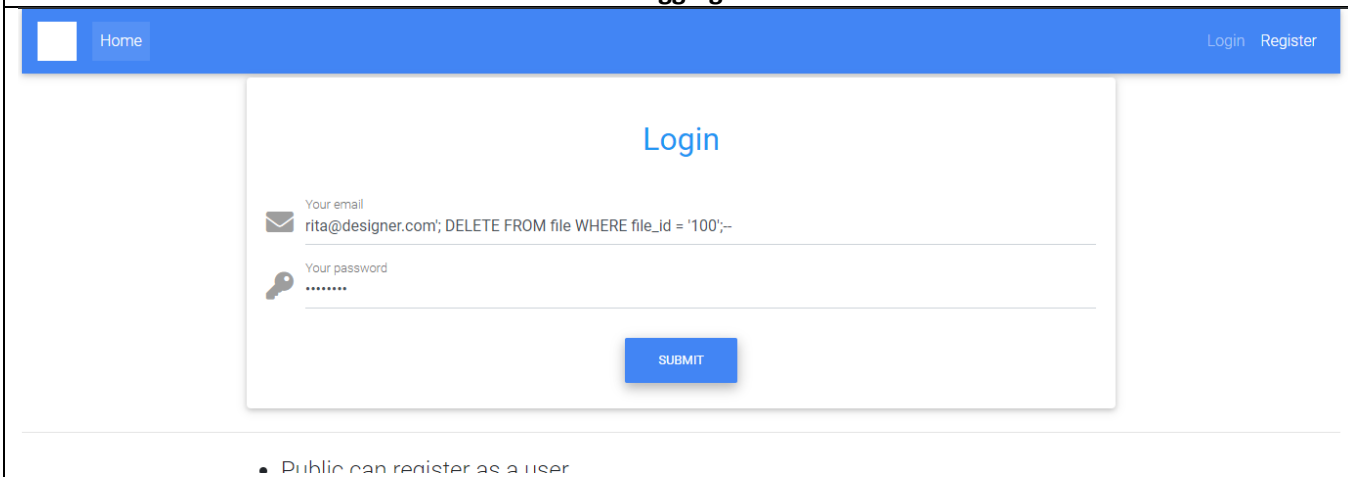
```
module.exports.updateUser = (recordId, newRoleId) => {

    return new Promise((resolve, reject) => {
        //I referred to https://www.codota.com/code/javascript/functions/mysql/Pool/getConnection
        //to prepare the following code pattern which does not use callback technique (uses Promise technique)
        pool.getConnection((err, connection) => {
            if (err) {
                console.log('Database connection error ', err);
                resolve(err);
            } else {
                connection.query(`UPDATE user SET role_id =${newRoleId} WHERE user_id=${recordId}`, (err, rows) => {
                    if (err) {
                        reject(err);
                    } else {
                        resolve(rows);
                    }
                    connection.release();
                });
            }
        });
    }); //End of new Promise object creation

} //End of updateUser
```

```
module.exports.updateUser = (recordId, newRoleId) => {
    console.log("============================================================");
    console.log("userService.js > updateUser is called and running!");
    console.log("============================================================");

    return new Promise((resolve, reject) => {
        //I referred to https://www.codota.com/code/javascript/functions/mysql/Pool/getConnection
        //to prepare the following code pattern which does not use callback technique (uses Promise technique)
        pool.getConnection((err, connection) => {
            if (err) {
                console.log('Database connection error ', err);
                resolve(err);
            } else {
                connection.query(`UPDATE user SET role_id = ? WHERE user_id = ?`, [newRoleId, recordId], (err, rows) => {
                    if (err) {
                        reject(err);
                    } else {
                        resolve(rows);
                    }
                    connection.release();
                });
            }
        });
    }); //End of new Promise object creation

} //End of updateUser
```

Original Codes	Fixed Codes
backEnd > src > services > userService.js (module.exports. getOneUserData)	
	
backEnd > src > services > userService.js (module.exports. getOneUserDataByEmail)	
	
backEnd > src > services > userService.js (module.exports. getOneDesignData)	
	

Original Codes	Fixed Codes
backEnd > src > services > userService.js (module.exports.updateDesign)	
	
backEnd > src > services > fileService.js (module.exports.createFileData)	
	

Demo Screenshot (After)
- Logging In



- Public can register as a user

In our testing, we tried to delete the file with file_id of 100 from the database with the patched SQL injection codes

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

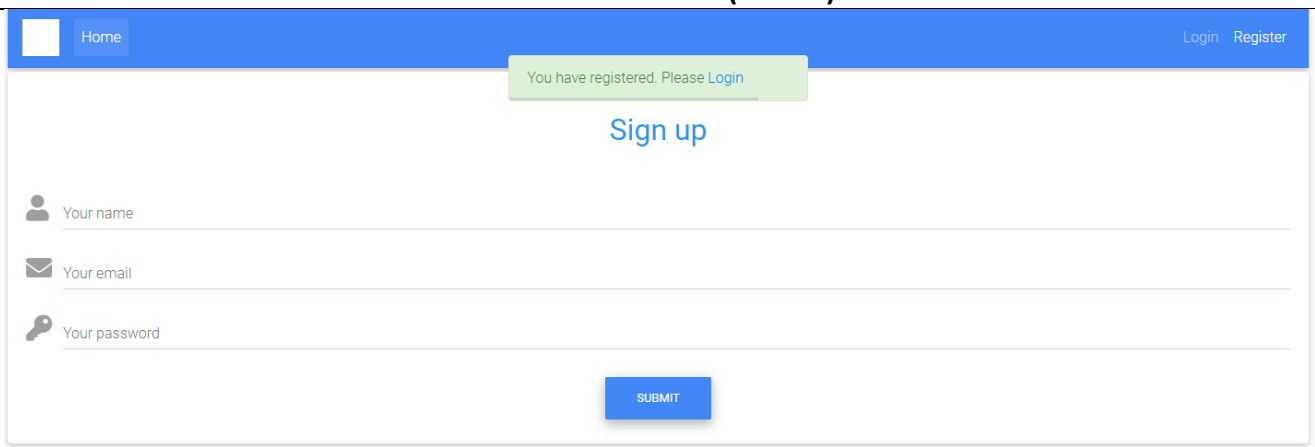
file_id	cloudinary_file_id	cloudinary_url	design_title	design_description	created_by_id
100	Design/hbukmdugiatdbmgobrdq	http://res.cloudinary.com/wongwj/image/uploa...	rita design ...	rita design 1 description text 1 text 2 text 3 text 4	100
101	Design/jecbaawpweki0ru0dq2c	http://res.cloudinary.com/wongwj/image/uploa...	rita design 2	rita design 2 description text 1 text 2 text 3 tex...	100
102	Design/bjwejl9alijhcvzgdrre	http://res.cloudinary.com/wongwj/image/uploa...	rita design 3	rita design 3 description text 1 text 2 text 3 tex...	100
103	Design/hinkgxzmcluzieqabdmf	http://res.cloudinary.com/wongwj/image/uploa...	rita design 4	rita design 4 description text 1 text 2 text 3 tex...	100
104	Design/kwjkdgf4rfbw1j4a1bj2	http://res.cloudinary.com/wongwj/image/uploa...	rita design 5	rita design 5 description text 1 text 2 text 3 tex...	100
105	Design/gttdb6yd5qeqqaiansvp	http://res.cloudinary.com/wongwj/image/uploa...	rita design 6	rita design 6 description text 1 text 2 text 3 tex...	100
106	Design/wul3xphulzmz6ekk8j68	http://res.cloudinary.com/wongwj/image/uploa...	rita design 7	rita design 7 description text 1 text 2 text 3 tex...	100
107	Design/dlihtck8cimouxg7hxwl	http://res.cloudinary.com/wongwj/image/uploa...	rita design 8	rita design 8 description text 1 text 2 text 3 tex...	100
108	Design/pb0r2sbqegtkxbv6gbvj	http://res.cloudinary.com/wongwj/image/uploa...	rita design 9	rita design 9 description text 1 text 2 text 3 tex...	100
109	Design/co96nxfvggp43waomrms	http://res.cloudinary.com/wongwj/image/uploa...	rita design 10	rita design 10 description text 1 text 2 text 3 te...	100
110	Design/jwxgo5fikjwdwdr3ma9t	http://res.cloudinary.com/wongwj/image/uploa...	rita design 11	rita design 11 description text 1 text 2 text 3 te...	100
111	Design/oxkkumijs08rvq3wv8on	http://res.cloudinary.com/wongwj/image/uploa...	rita design 12	rita design 12 description text 1 text 2 text 3 te...	100
112	Design/alyphhtik9sossk6cngo	http://res.cloudinary.com/wongwj/image/uploa...	Bob Design 1	Bob Design Description 1	102
115	Design/smuryoptiwbgraam2zs1	http://res.cloudinary.com/wongwj/image/uploa...	Bob Design 2	Bob Design 2 Description	102
117	Design/r5emil6krvlsztjo5osj	http://res.cloudinary.com/wongwj/image/uploa...	Bob Design 3	Bob Design Description 3	102
118	Design/ijojg07t2caosy6bfpbg	http://res.cloudinary.com/wongwj/image/uploa...	!	!	102
119	Design/tnvgbeghdurlx4m1sftn	http://res.cloudinary.com/wongwj/image/uploa...	Bob Design 6	Details	102
120	Design/mt69kbtvou9aztqzh2qa	http://res.cloudinary.com/wongwj/image/uploa...	Braddy 1	BRaddy 1 D	103
121	Design/zo7mpazjawscguzq0cxe	http://res.cloudinary.com/wongwj/image/uploa...	Braddy 2	BRaddy 2D	103

As seen in our database, our patched codes worked and file_id 100 still exists, preventing SQL injection in our login.html

Vulnerability 2 – Broken Authentication (Input Validation)

V-02 Broken Authenticati on (Input Validation)	Input Validation was not done for registration, therefore any user can register without an email or password. Impact – Medium	Any user can go the registration page and sign up without any name, email or password entered in the text field. Ease of exploit - Easy	Recommend using regular expression checks as front-end user input validation to ensure that user does not input any malicious JavaScript codes or empty fields. This also helps fix the SQL injection issue as mentioned in Vulnerability 1

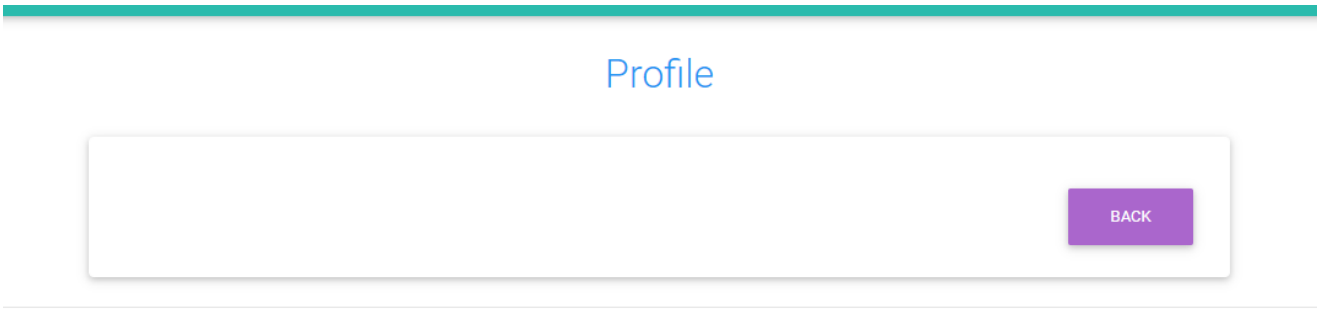
Demo Screenshot (Before)



The screenshot above shows that users can register without any input.

113	joe	joe@admin.com	$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg...	2
114	gabby	gabby@designer.com	$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg...	2
121			$2b$10$QLkBkffqSNaGQrNfRa.yTusM7HJVg6zZ...	2

As seen in the screenshot above, a new role is created in the user database where there is no name, email, or password in the database. The password will still be automatically created as the empty password is hashed.

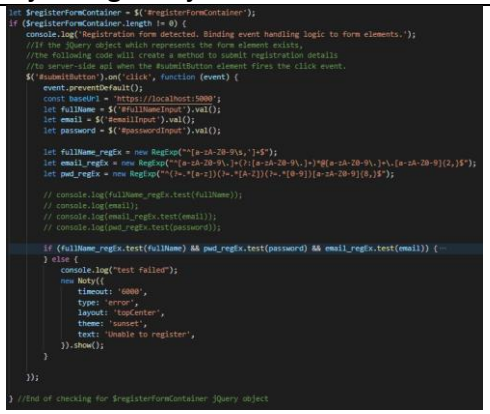


The nameless user has created an account inside the database with no information and has normal user access.

frontEnd > public > js > register.js

```
let $registerFormContainer = $('#registerFormContainer');
if ($registerFormContainer.length != 0) {
    console.log('Registration form detected. Binding event handling logic to form elements.');
    //If the jQuery object which represents the form element exists,
    //the following code will create a method to submit registration details
    //to server-side api when the #submitButton element fires the click event.
    $('#submitButton').on('click', function(event) {
        event.preventDefault();
        const baseUrl = 'http://localhost:5000';
        let fullName = $('#fullNameInput').val();
        let email = $('#emailInput').val();
        let password = $('#passwordInput').val();
        let webFormData = new FormData();
        webFormData.append('fullName', fullName);
        webFormData.append('email', email);
        webFormData.append('password', password);
        axios({···
            })
            .then(function(response) {···
            })
            .catch(function(response) {···
            });
    });

} //End of checking for $registerFormContainer jQuery object
```

The above codes extracted from the frontEnd: register.js shows that the input, once received, is not validated in client-side codes using JavaScript and is immediately sent to the Backend.



To ensure that the input is validated, we included regex check for all 3 components of the input form: username, email as well as password.

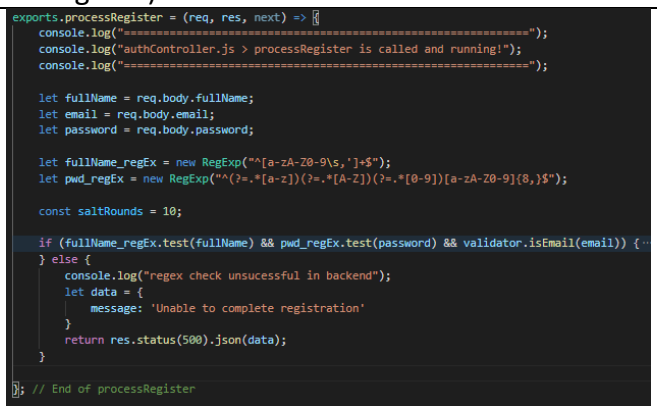
This ensures that all 3 components of the input form are not being submitted with malicious codes that could potentially cause data leakage or harm to the database.

backEnd > src > controllers > authController.js
(exports.processRegister)

```
exports.processRegister = (req, res, next) => {
    console.log('processRegister running.');
    let fullName = req.body.fullName;
    let email = req.body.email;
    let password = req.body.password;


    bcrypt.hash(password, 10, async (err, hash) => {
        if (err) {
            console.log('Error on hashing password');
            return res.status(500).json({ statusMessage: 'Unable to complete registration' });
        } else {

            results = user.createUser(fullName, email, hash, function (results, error) {
                if (results != null) {
                    console.log(results);
                    return res.status(200).json({ statusMessage: 'Completed registration.' });
                }
                if (error) {
                    console.log('processRegister method : callback error block section is running.');
                    console.log(error, '============================================================');
                    return res.status(500).json({ statusMessage: 'Unable to complete registration' });
                }
            });//End of anonymous callback function


        }
    });

}; // End of processRegister
```

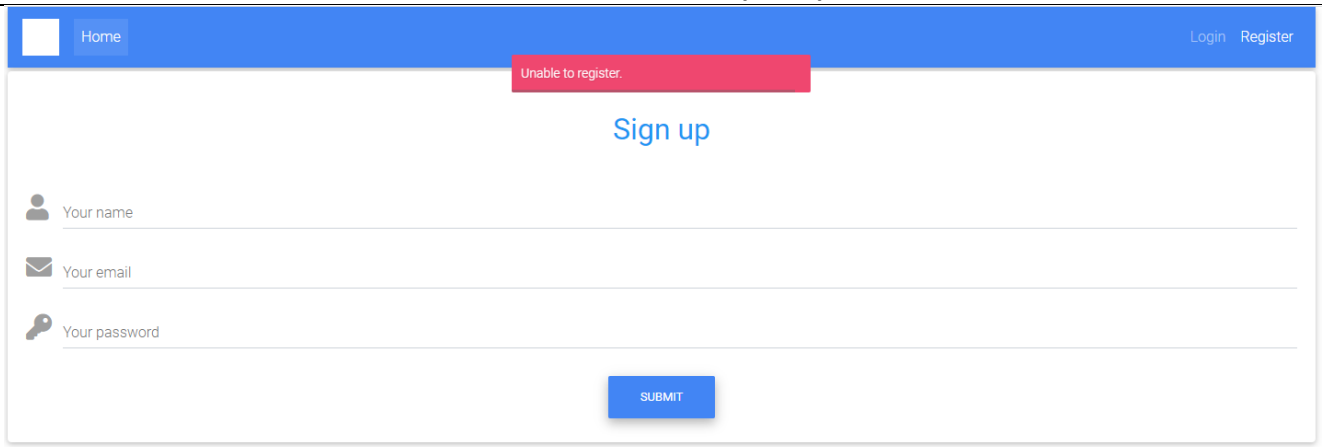
As you can see in the above screenshot, the values received are not checked in anyway, and immediately sent to database through the call of “user.createUser” to insert into the database.

This is a call for concern as the fullName may not be a nice input and email may not be email at all. Attackers could input scripts or malicious codes which will get stored inside the database and executed when it is being called. This leads to something called a Stored XSS attack.

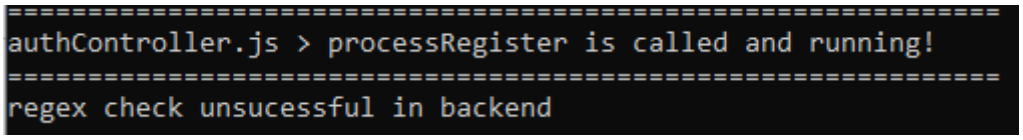


Hence, to solve that issue, we make use of Regular Expressions (regex) to test the input based on the regex that we created to ensure that the input is validated before sending it to the database, otherwise it will be kicked out of the system.

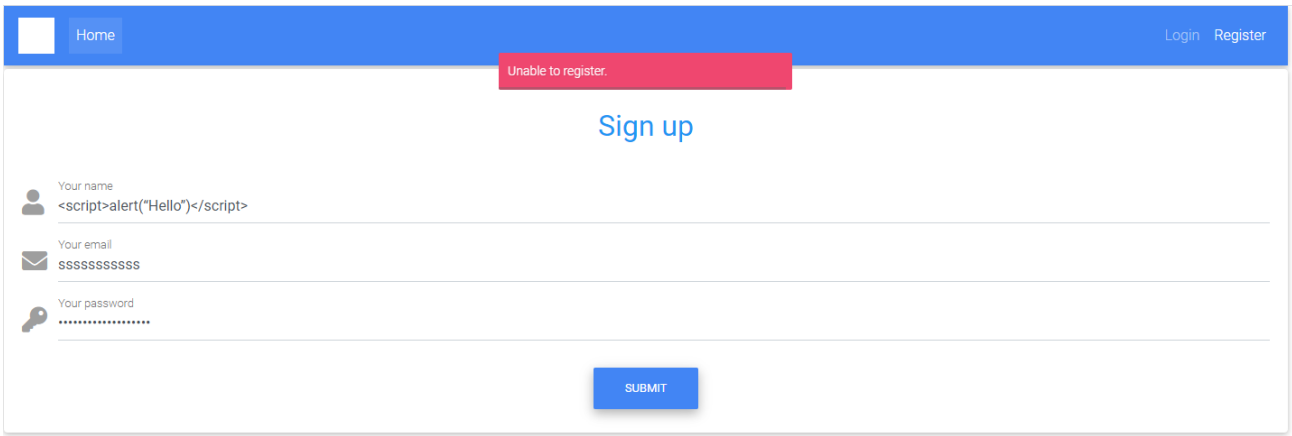
Demo screenshot (After)



After adding the regular expression check, the website no longer registers users if they have no input



The back endconsole displays that the regular expression check for user regristration is not successful



Additionally, if users use special characters in their name, email or password (e.g. < > ( ) { } “) they will not be able to make an account. (Excluding the usage of @ and . for email)

Vulnerability 3 – Logging and Monitoring

V-06 Logging and monitoring	Insufficient logging throughout the system can cause difficulty when troubleshooting issues in the future. Logging can also help provide real time warnings for admin and developers. Impact - Low	One example is that hackers can simply use a brute force attack when they obtain the account's email to try and login with commonly used passwords. Without logging, the hacking attempt will be unknown to the administrator, and they cannot warn the user that their account has been compromised Ease of exploit - Easy	Recommend to log failed login attempts as well as display on console so that suspicious activity can be monitored as well as logged and account can be locked to prevent brute force attacks. Automatically locking out accounts after certain number of failed attempts to block brute force attacks would help but also could be ineffective as hackers can simply bypass the lockout by attempting to login before the lockout occurs. Other backend errors should be logged as well to help future administrators and programmers troubleshoot easily when a future issue arises.

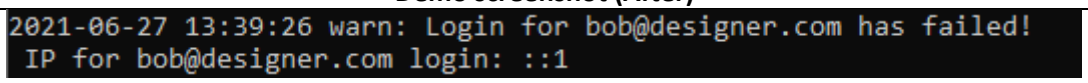
Fixed Codes

backEnd > src > controllers > authController.js
(exports.processRegister) ← e.g.

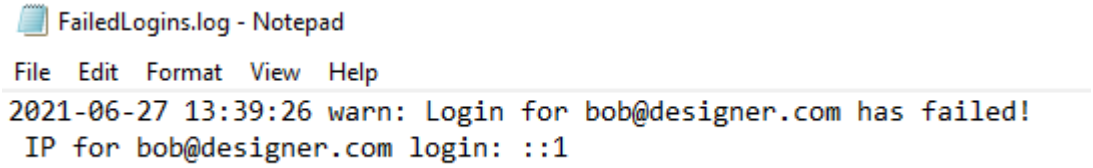
```
const loggerformat = printf(({ level, message, timestamp }) => {
    return `${timestamp} ${level}: ${message}`;
});
const logger = createLogger({
    format: format.combine(
        format.timestamp({ format: 'YYYY-MM-DD HH:mm:ss' }),
        loggerformat
    ),
    transports: [
        new transports.Console(),
        new transports.File({ filename: './Logs/FailedLogins.log', level: 'warn' }),
    ],
});
logger.warn('Login for ' + email + ' has failed!\n IP for ' + email + ' login: ' + req.ip)
let data = {
    message: 'Credentials are not valid'
};
return res.status(500).json(data);
```

In our codes, we have installed the module Winston to help us in our logging. It displays the warning in the console and furthermore logs it in a sperate folder called Logs

Demo screenshot (After)



In the console, there is a warning for failed logins for administrators to monitor from winston

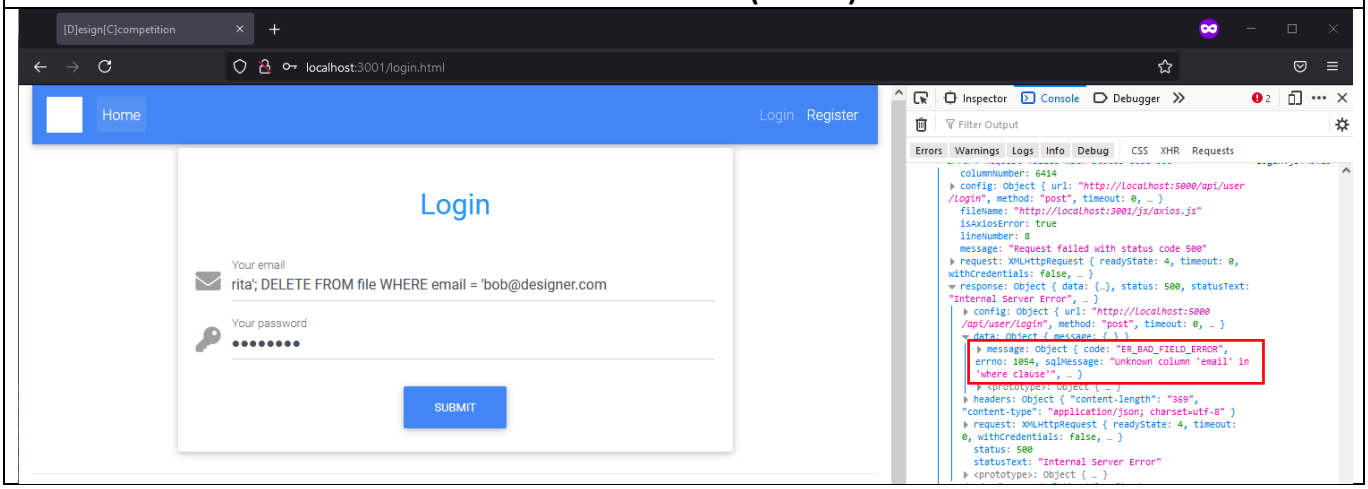


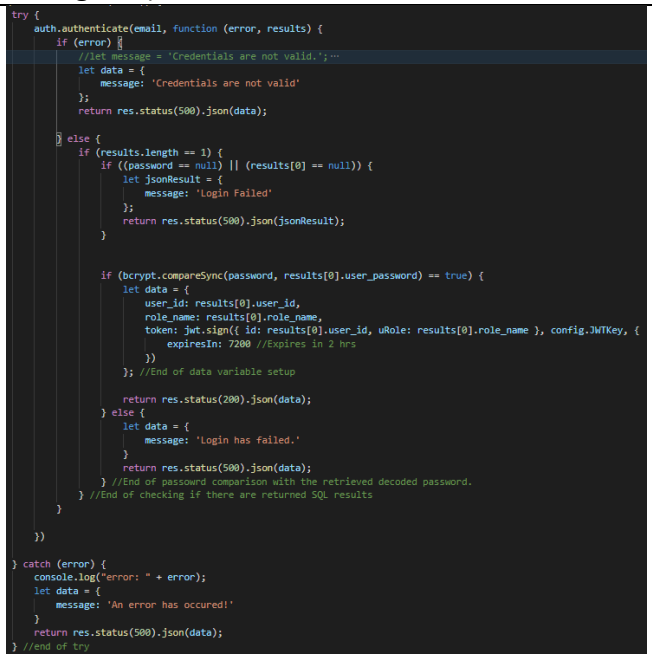
In this case, the time of failed login and email has been logged in a file called FailedLogs.log. The IP for the failed login is shown as : : 1 because it is being ran in localhost.

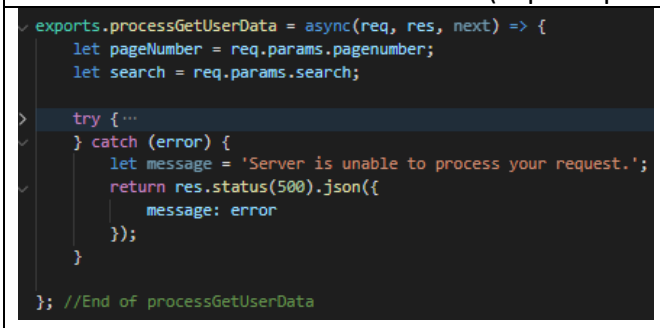
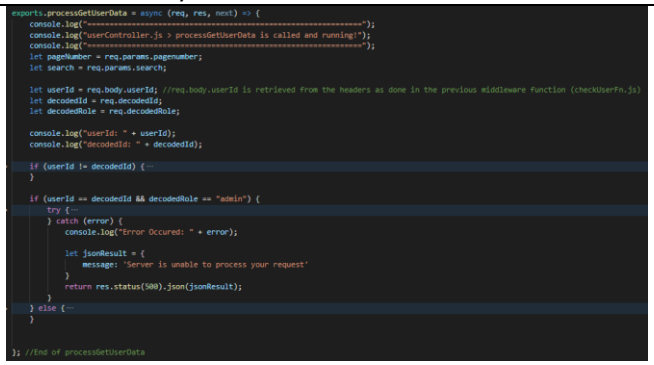
Vulnerability 4 – Sanitize JSON Results

V-03 Sensitive data exposure (Sanitize JSON response)	SQL Error messages are directly printed for the Frontend users to see, which can be a vulnerability as this allows the attacker to make use of the SQL errors to craft their responses Impact - Low	When an error is made logging into the website, the google chrome console displays the error which attackers can fine tune their SQL injection statements based on the error messages that are getting sent to the user.	Recommend to change all the JSON response to return customised error messages rather than printing out the SQL error directly to the user. This is to prevent the attacker who has strong knowledge on SQL statements from relying heavily on SQL return statements to craft their next attack and send it back through the endpoints.

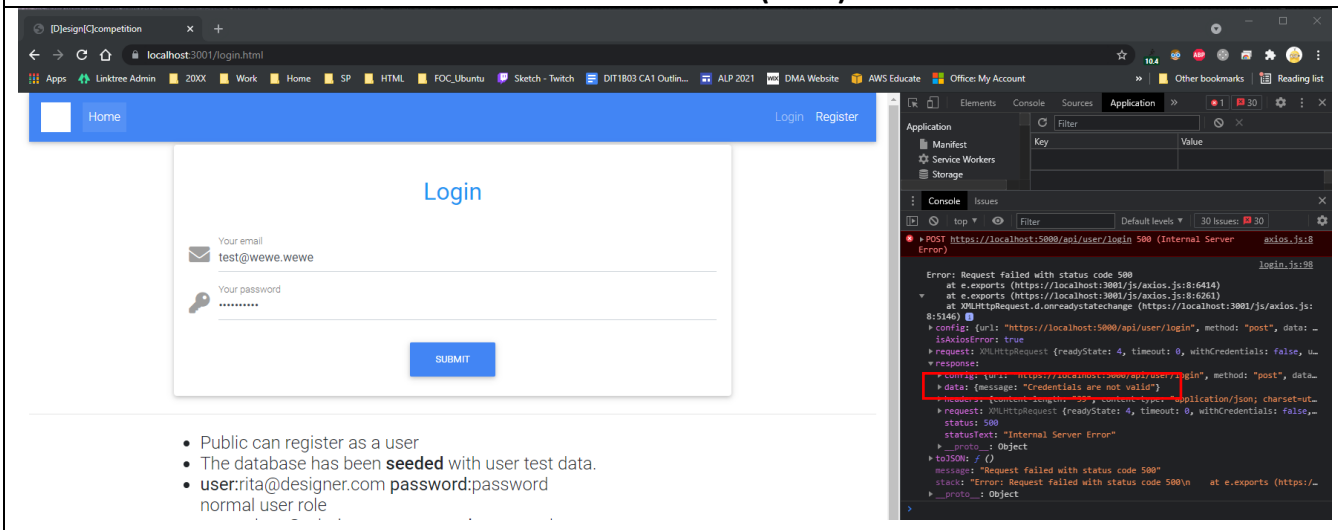
Demo screenshot (Before)



Original Codes	Fixed Codes
backEnd > src > controllers > authController.js (exports.processRegister)	
	
In the above codes, you can see that when there is an error of when the login has failed, the SQL error is immediately packaged into a JSON object and sent back to the client (frontEnd). This leads to a potential sensitive data exposure as if the endpoint called is to retrieve user data, it will display the data of that user.	To prevent or minimize the risk of attackers getting the SQL statement responses, we decided to package a custom error message to send to the client so that they will only see what we want them to see while the SQL statement responses is printed in the console in the backEnd server.

Original Codes	Fixed Codes
backEnd > src > controllers > userController.js (exports.processGetUserData)	
	
The above code sends the error that is received from the server directly to the client, which is not good as it leads to potential sensitive data leakage. Moreover, there is a message statement declared, but it was never read/sent to the client, hence the message variable declared was useless	The above codes address the issues mentioned and minimizes the risk of the user getting the data. The jsonResult object will send back the customized message to the client so that the client is well aware of the current state of their request.

Demo screenshot (After)

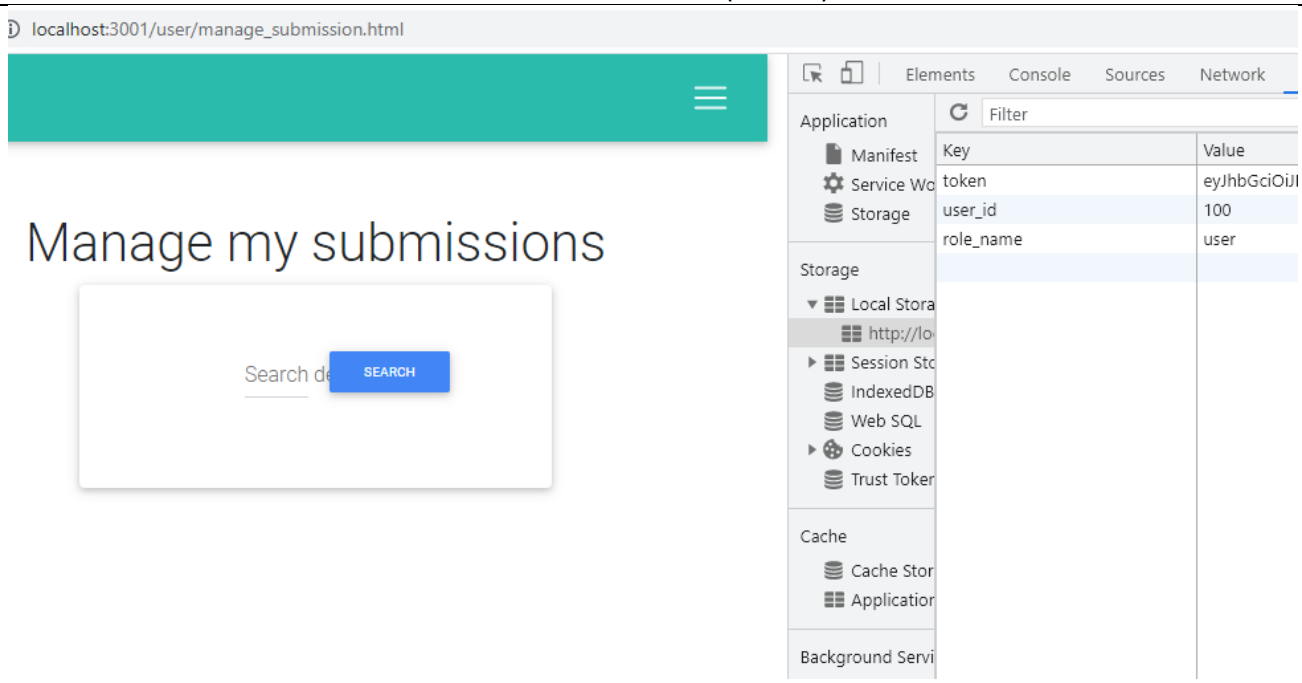


As seen in the google chrome console, the JSON result has been sanatized

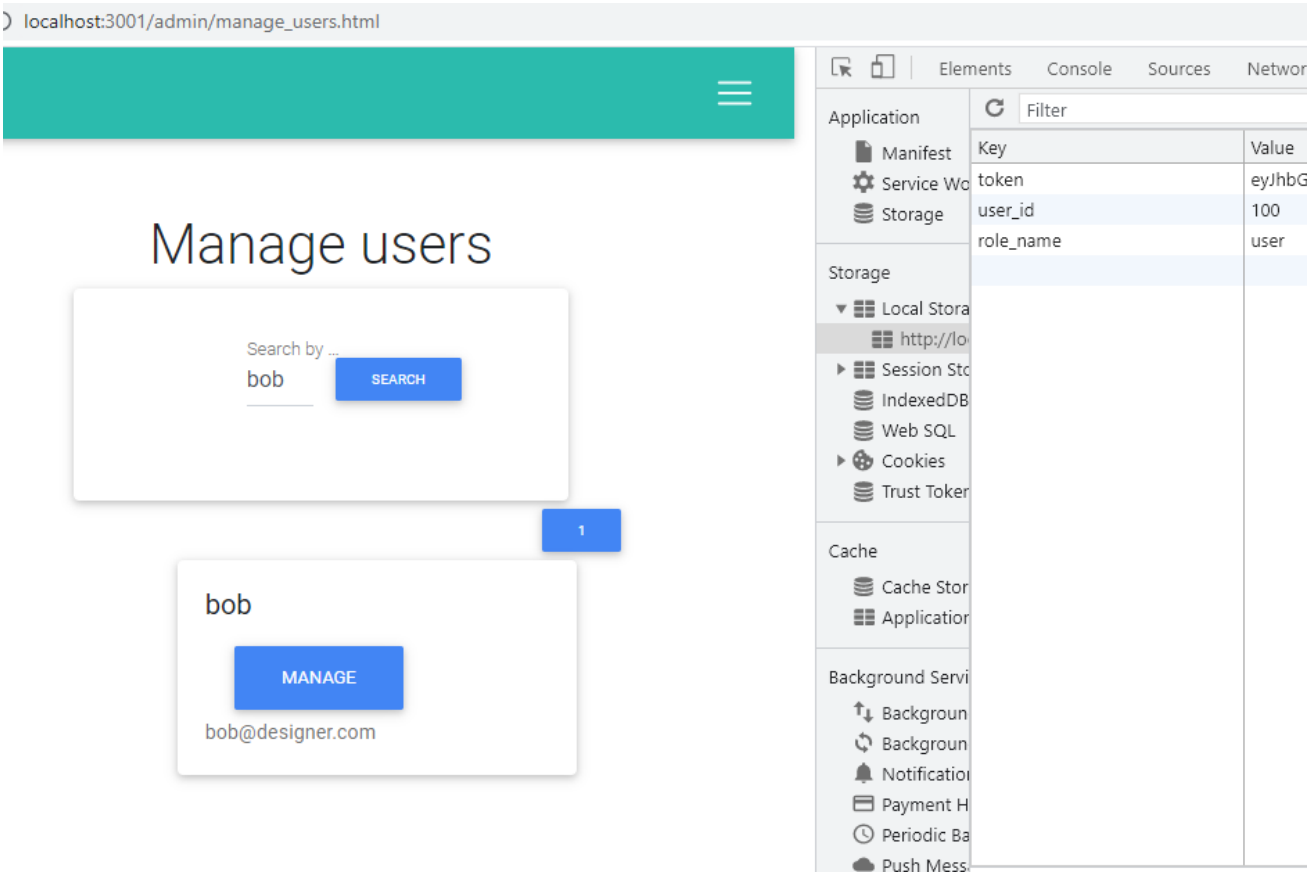
Vulnerability 5 – Broken Authentication (Restricted Pages Access)

V-02 Broken Authentication (Restricted Pages Access)	Users can access administrator only pages by changing the URL. To ensure that the request can be made when the logged in user is admin, we must send authorization headers to the Backend to be successfully authorized. Impact – High	The URL can simply be changed to an administrator only page and can be accessed by all users. In this case as the URL of the pages are easy to guess, the hacker can change to /admin/manage_user.html. This allows hackers to access administrator only pages with a normal user account and perform administrator only. Ease of exploit - Easy	Recommend validating the user's jwt token and user id to lock unauthorized users out of administrator pages using a middleware function to ensure that privileges can only be performed by an administrator and not by any user.

Demo screenshots (Before)



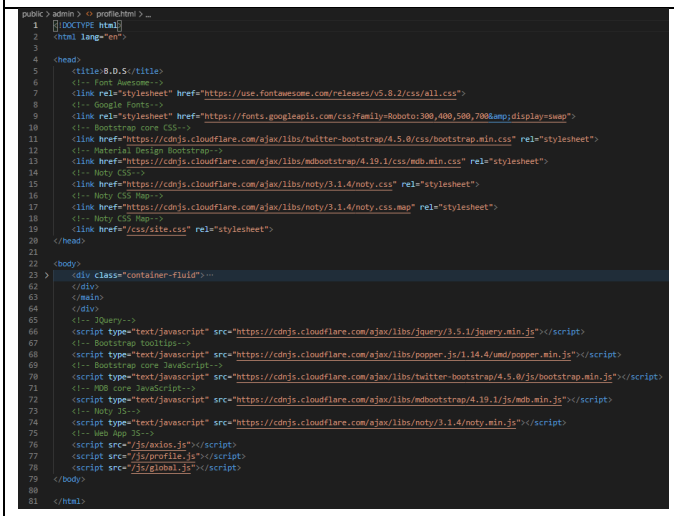
Currently, the user is logged in as rita@designer.com as seen in the localstorage on the right



By simply changing the URL to /admin/manage_users.html as seen above, rita even though a normal user is able to access administrator pages as there is no authentication being made.

Added a script for the follwing codes that will run everytime the webpage is called. This script will send the token to the endpoint in the backEnd to check if the user has the proper authentication and authorization for that page. If not, the user is directed out of the page to the login page, indicating that the user does not have access to that particular page

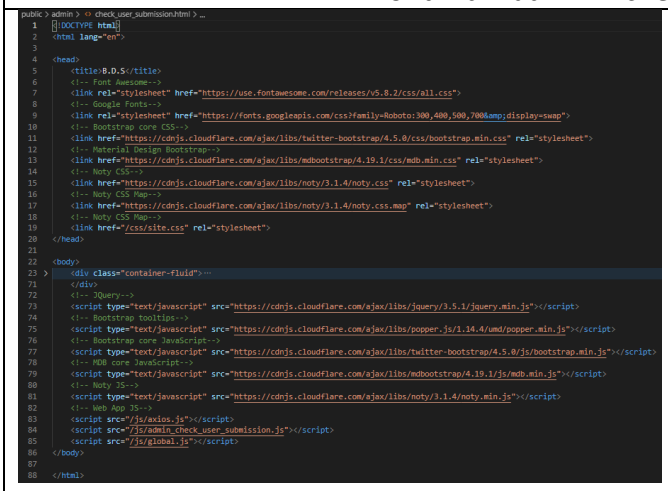
Original Codes	Fixed Codes
frontEnd > admin > profile.html	



```
public > admin > profile.html > ...
<!DOCTYPE html>
<html lang="en">

<head>
    <title>B.D.S</title>
    <!-- Font Awesome-->
    <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.8.2/css/all.css">
    <!-- Google Fonts-->
    <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Roboto:300,400,500,700&amp;display=swap">
    <!-- Bootstrap core CSS-->
    <link href="https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/4.5.0/css/bootstrap.min.css" rel="stylesheet">
    <!-- Material Design Bootstrap-->
    <link href="https://cdnjs.cloudflare.com/ajax/libs/mdbootstrap/4.19.1/css/mdb.min.css" rel="stylesheet">
    <!-- Noty CSS-->
    <link href="https://cdnjs.cloudflare.com/ajax/libs/noty/3.1.4/noty.css" rel="stylesheet">
    <!-- Noty CSS Map-->
    <link href="https://cdnjs.cloudflare.com/ajax/libs/noty/3.1.4/noty.css.map" rel="stylesheet">
    <!-- Noty CSS Map-->
    <link href="/css/site.css" rel="stylesheet">
</head>

<body>
    <div class="container-fluid">···
    </div>
    </main>
    </div>
    <!-- JQuery-->
    <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
    <!-- Bootstrap tooltips-->
    <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.4/umd/popper.min.js"></script>
    <!-- Bootstrap core JavaScript-->
    <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/4.5.0/js/bootstrap.min.js"></script>
    <!-- MDB core JavaScript-->
    <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/mdbootstrap/4.19.1/js/mdb.min.js"></script>
    <!-- Noty JS-->
    <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/noty/3.1.4/noty.min.js"></script>
    <!-- Web App JS-->
    <script src="/js/axios.js"></script>
    <script src="/js/profile.js"></script>
    <script src="/js/global.js"></script>
</body>

</html>
```

```
public > user > profile.html > html
    <script type="text/javascript"
        src="https://cdnjs.cloudflare.com/ajax/libs/mdbootstrap/4.19.1/js/mdb.min.js"></script>
    <!-- Noty JS-->
    <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/noty/3.1.4/noty.min.js"></script>
    <!-- Web App JS-->
    <script src="/js/axios.js"></script>
    <script>

        const token = sessionStorage.getItem("token");
        const role = sessionStorage.getItem("role_name");
        const loggedInUserID = parseInt(sessionStorage.getItem("user_id"));
        const baseUrl = "https://localhost:5000";

        // ----------------------------------------------
        // Checking if the user has logged in or not
        // ----------------------------------------------
        if (token === null || isNaN(loggedInUserID)) {
            console.log("no token so outta here");
            window.location.href = "/login.html";
        } else {
            // ----------------------------------------------------------
            // Backend call to check token validity & authorisation
            // ----------------------------------------------------------
            axios({
                method: 'post',
                url: baseUrl + '/api/user/authentication-check',
                //data: webFormData,
                headers: {
                    authorization: "Bearer " + token,
                    'user': loggedInUserID
                }
            })
                .then(function (response) {
                    if (response.data.userRole == "user") {
                        console.log("Access Granted!");
                    } else {
                        console.log("Access Denied!");
                        sessionStorage.clear();
                        window.location.href = "/home.html";
                    }
                })
                .catch(function (response) {
                    console.log("Error!!");
                    console.log(response);

                    if (response.response.status == 403) {
                        sessionStorage.clear();
                        window.location.href = "/login.html";
                    } else {
                        window.location.href = "/home.html";
                    }
                });
        }
    </script>
</head>

<body>
    <div class="container-fluid">···
    </div>
```

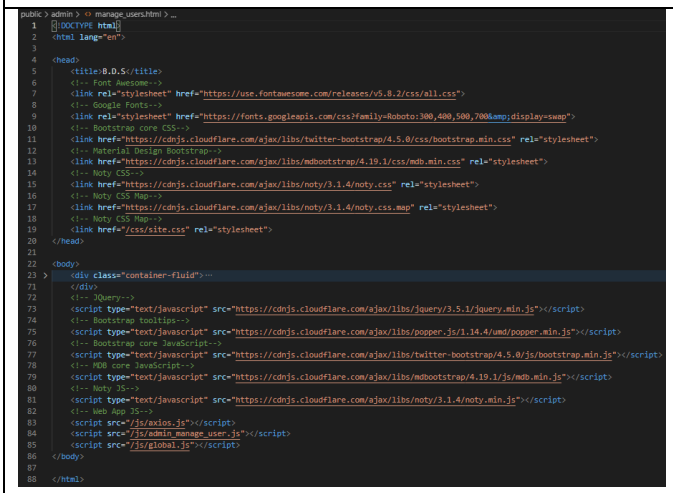
frontEnd > admin > check_user_submission.html



```
<!DOCTYPE html>
<html lang="en">

<head>
    <title>B.D.S</title>
    <!-- Font Awesome -->
    <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.8.2/css/all.css">
    <!-- Google Fonts -->
    <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Roboto:300,400,500,700&amp;display=swap">
    <!-- Bootstrap core CSS -->
    <link href="https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/4.5.0/css/bootstrap.min.css" rel="stylesheet">
    <!-- Material Design Bootstrap -->
    <link href="https://cdnjs.cloudflare.com/ajax/libs/mdbootstrap/4.19.1/css/mdb.min.css" rel="stylesheet">
    <!-- Noty CSS -->
    <link href="https://cdnjs.cloudflare.com/ajax/libs/noty/3.1.4/noty.css" rel="stylesheet">
    <!-- Noty CSS Map -->
    <link href="https://cdnjs.cloudflare.com/ajax/libs/noty/3.1.4/noty.css.map" rel="stylesheet">
    <!-- Noty CSS Map -->
    <link href="/css/site.css" rel="stylesheet">
</head>

<body>
    <div class="container-fluid">
    </div>
    <!-- JQuery -->
    <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
    <!-- Bootstrap tooltips -->
    <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.4/umd/popper.min.js"></script>
    <!-- Bootstrap core JavaScript -->
    <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/4.5.0/js/bootstrap.min.js"></script>
    <!-- MDB core JavaScript -->
    <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/mdbootstrap/4.19.1/js/mdb.min.js"></script>
    <!-- Noty JS -->
    <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/noty/3.1.4/noty.min.js"></script>
    <!-- Web App JS -->
    <script src="/js/axios.js"></script>
    <script src="/js/admin_check_user_submission.js"></script>
    <script src="/js/global.js"></script>
</body>

</html>
```

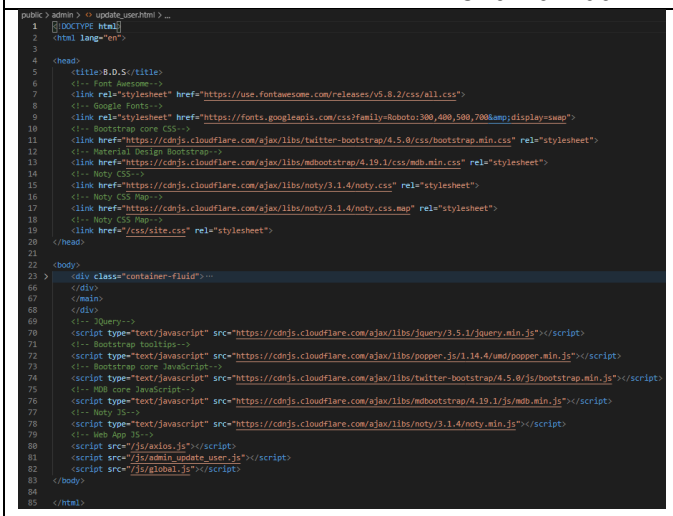
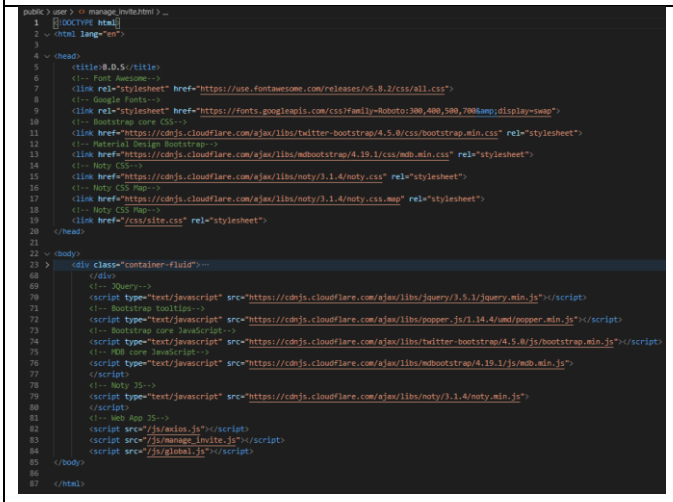
```
        <script>

            const token = sessionStorage.getItem("token");
            const role = sessionStorage.getItem("role_name");
            const loggedInUserID = parseInt(sessionStorage.getItem("user_id"));
            constbaseUrl = 'https://localhost:5000';

            // ------------------------------------------------
            // Checking if the user has logged in or not
            // ------------------------------------------------
            if (token === null || isNaN(loggedInUserID)) {
                console.log("no token so outta here");
                window.location.href = "/login.html";
            } else {
                // ------------------------------------------------------
                // Backend call to check token validity & authorisation
                // ------------------------------------------------------
                axios({
                    method: 'post',
                    url: baseUrl + '/api/user/authentication-check',
                    //data: webFormData,
                    headers: {
                        authorization: "Bearer " + token,
                        'user': loggedInUserID
                    }
                })
                    .then(function (response) {
                        if (response.data.userRole == "admin") {
                            console.log("Access Granted!");
                        } else {
                            console.log("Access Denied!");
                            sessionStorage.clear();
                            window.location.href = "/home.html";
                        }
                    })
                    .catch(function (response) {
                        console.log("Error!!");
                        console.log(response);

                        if (response.response.status == 403) {
                            sessionStorage.clear();
                            window.location.href = "/login.html";
                        } else {
                            window.location.href = "/home.html";
                        }
                    });
            }
        </script>
    </head>

> <body> ···
    </body>

</html>
```

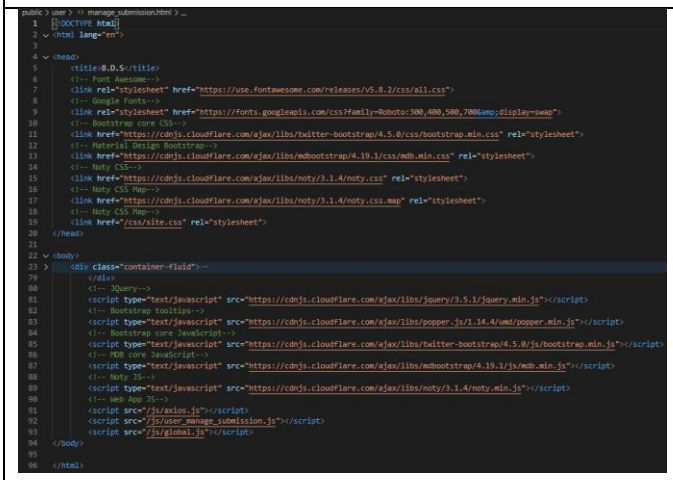
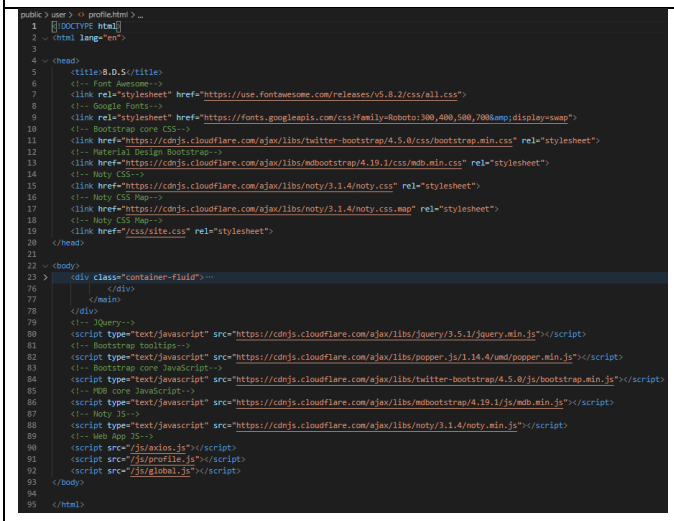
frontEnd > admin > manage_users.html

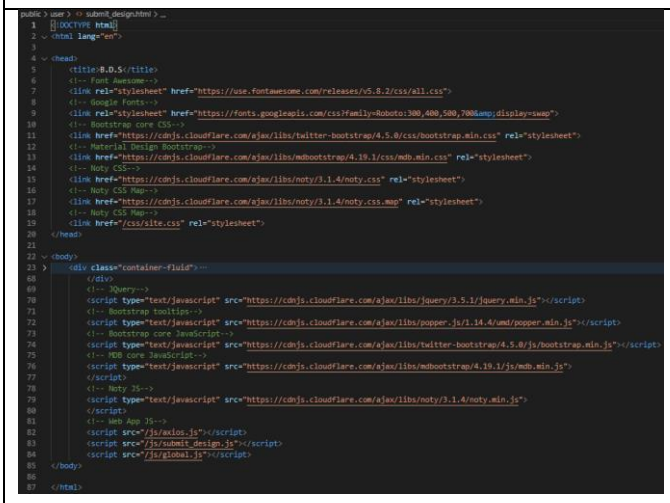
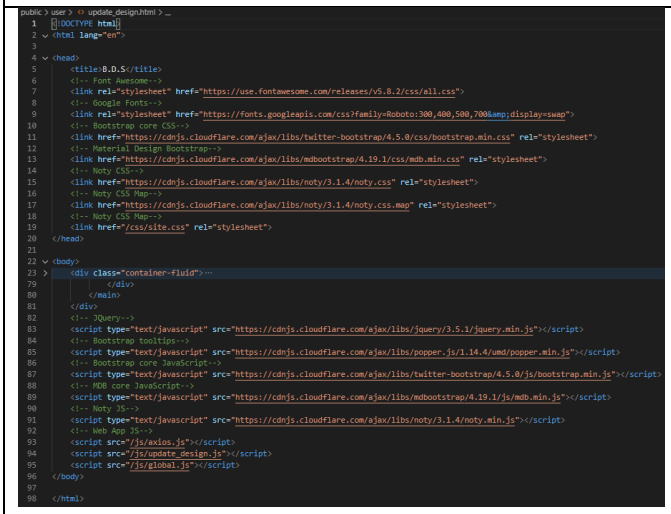


```
<!DOCTYPE html>
<html lang="en">

<head>
    <title>B.D.S</title>
    <!-- Font Awesome -->
    <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.8.2/css/all.css">
    <!-- Google Fonts -->
    <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Roboto:300,400,500,700&amp;display=swap">
    <!-- Bootstrap core CSS -->
    <link href="https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/4.5.0/css/bootstrap.min.css" rel="stylesheet">
    <!-- Material Design Bootstrap -->
    <link href="https://cdnjs.cloudflare.com/ajax/libs/mdbootstrap/4.19.1/css/mdb.min.css" rel="stylesheet">
    <!-- Noty CSS -->
    <link href="https://cdnjs.cloudflare.com/ajax/libs/noty/3.1.4/noty.css" rel="stylesheet">
    <!-- Noty CSS Map -->
    <link href="https://cdnjs.cloudflare.com/ajax/libs/noty/3.1.4/noty.css.map" rel="stylesheet">
    <!-- Noty CSS Map -->
    <link href="/css/site.css" rel="stylesheet">
</head>

<body>
    <div class="container-fluid">
    </div>
    <!-- JQuery -->
    <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
    <!-- Bootstrap tooltips -->
    <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.4/umd/popper.min.js"></script>
    <!-- Bootstrap core JavaScript -->
    <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/4.5.0/js/bootstrap.min.js"></script>
    <!-- MDB core JavaScript -->
    <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/mdbootstrap/4.19.1/js/mdb.min.js"></script>
    <!-- Noty JS -->
    <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/noty/3.1.4/noty.min.js"></script>
    <!-- Web App JS -->
    <script src="/js/axios.js"></script>
    <script src="/js/admin_manage_user.js"></script>
    <script src="/js/global.js"></script>
</body>

</html>
```

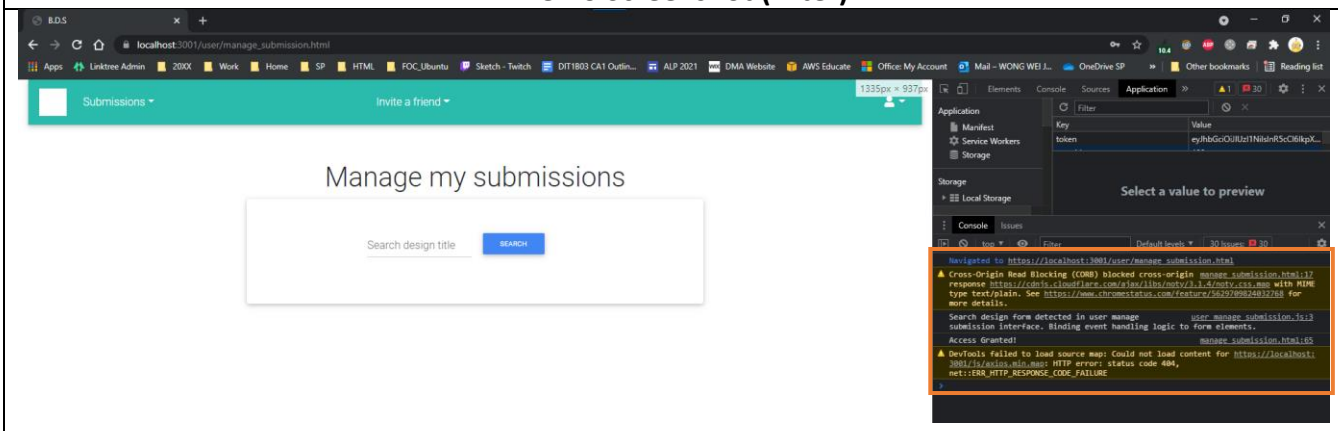
```
        <script>

            const token = sessionStorage.getItem("token");
            const role = sessionStorage.getItem("role_name");
            const loggedInUserID = parseInt(sessionStorage.getItem("user_id"));
            const baseUrl = "https://localhost:5000";

            // ------------------------------------------------
            // Checking if the user has logged in or not
            // ------------------------------------------------
            if (token === null || isNaN(loggedInUserID)) {
                console.log("no token so outta here");
                window.location.href = "/login.html";
            } else {
                // ------------------------------------------------------
                // Backend call to check token validity & authorisation
                // ------------------------------------------------------
                axios({
                    method: 'post',
                    url: baseUrl + '/api/user/authentication-check',
                    //data: webFormData,
                    headers: {
                        authorization: "Bearer " + token,
                        'user': loggedInUserID
                    }
                })
                    .then(function (response) {
                        if (response.data.userRole == "admin") {
                            console.log("Access Granted!");
                        } else {
                            console.log("Access Denied!");
                            sessionStorage.clear();
                            window.location.href = "/home.html";
                        }
                    })
                    .catch(function (response) {
                        console.log("Error!!");
                        console.log(response);
                        console.log(response);
                        if (response.response.status == 403) {
                            sessionStorage.clear();
                            window.location.href = "/login.html";
                        } else {
                            window.location.href = "/home.html";
                        }
                    });
            }
        </script>
    </head>

> <body> ···
    </body>

</html>
```

Original Codes	Fixed Codes
frontEnd > admin > update_user.html	
	
frontEnd > user > manage_invite.html	
	

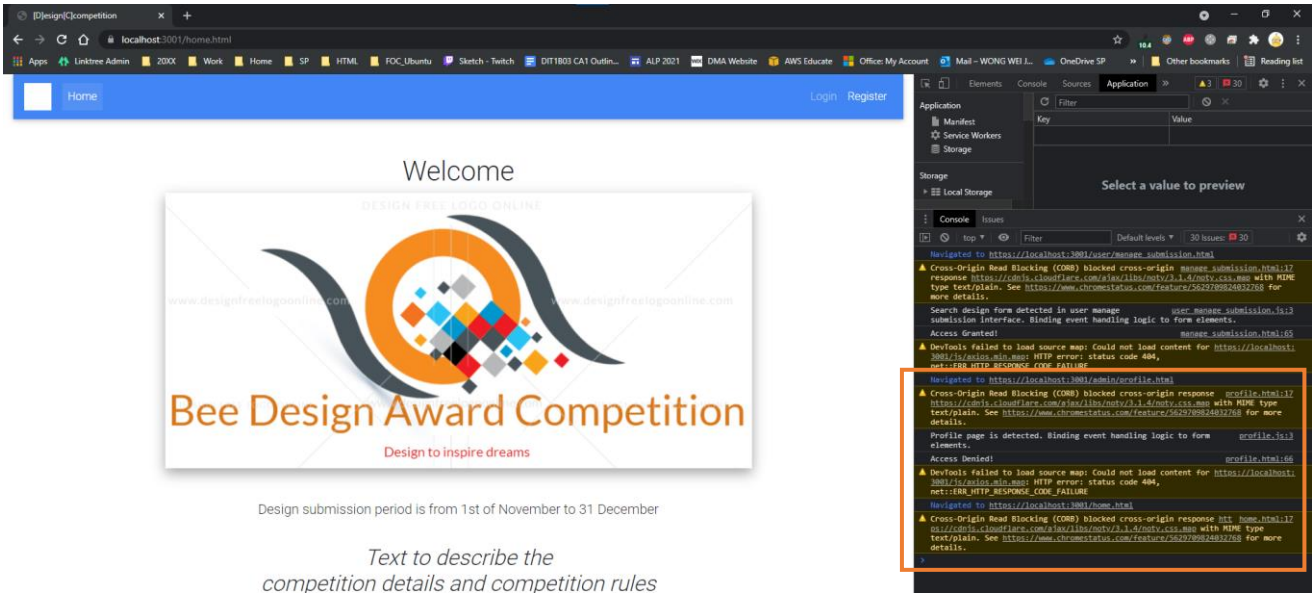
Original Codes	Fixed Codes
frontEnd > user > manage_submission.html	
	
frontEnd > user > profile.html	
	

Original Codes	Fixed Codes
frontEnd > user > submit_design.html	
	
frontEnd > user > update_design.html	
	

Demo Screenshot (After)



In the developer console, there is confirmation that access has been granted as the logged in user is an administrator



In this case, the user has been redirected to the home page and their localstorage is cleared as the script has detected that the user does not have the privilege to visit that page.

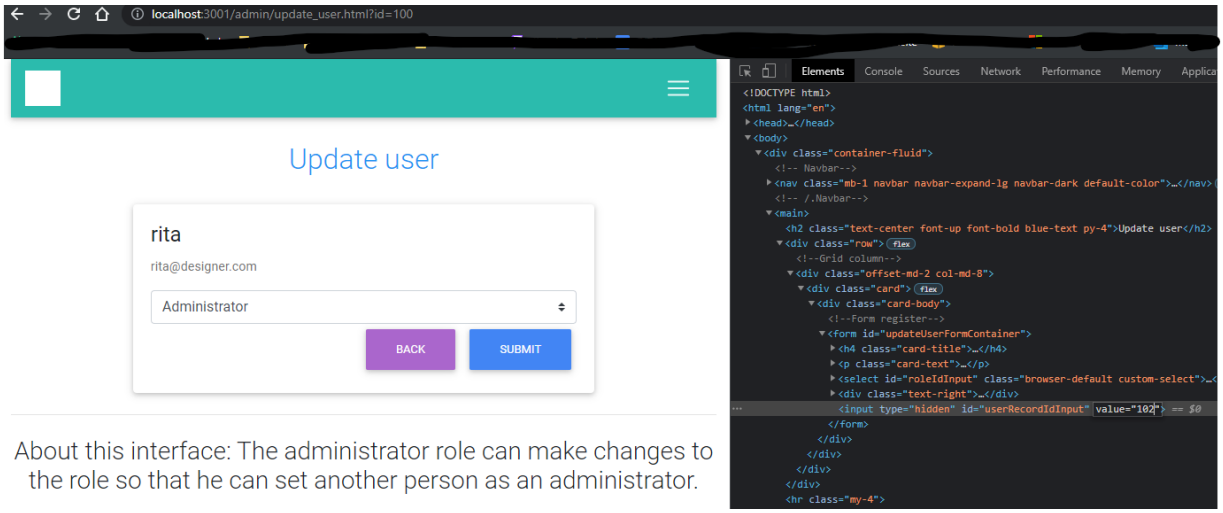
Vulnerability 6 – Broken Access Control

V-04 Broken Access Control	Users can run any of the admin functions regardless of user roles/permissions such updating the user role. Impact - High	After gaining access to the administrator only pages by changing the URL, the administrator only functions can also be called. One such example is changing the user's role in the system. By going to /admin/manage_user.html, any user can simply change their role in the system with the dropdown box and submit the change. Ease of exploit - Easy	Recommend including a validation middleware to validate any user's token before any function is called. This ensures that the user's identity is validated before any chances to the database can be made.

Demo Screenshot (Before)

user_id	fullname	email	user_password	role_id
100	rita	rita@designer.com	$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg...	2
101	robert	robert@admin.com	$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg...	1
102	bob	bob@designer.com	$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg...	2
103	braddy	braddy@designer.com	$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg...	2
104	josh	josh@designer.com	$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg...	2
105	john	john@designer.com	$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg...	2

This screenshot shows the database connected to the system where role_id 1 is an administrator and role_id 2 is a normal user. Currently bob is a normal user.

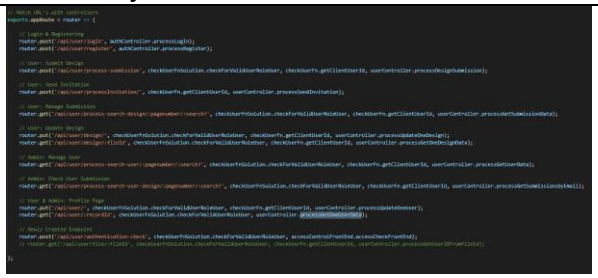
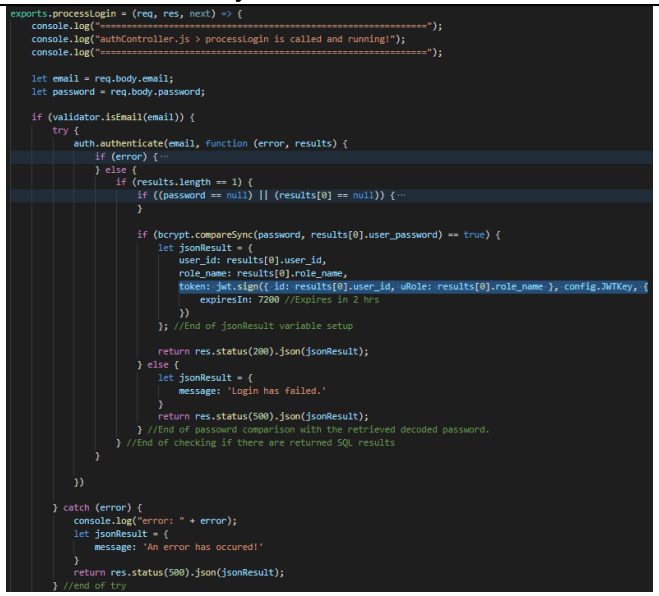


In the screenshot above, rita@designer.com a normal user can change the roles of any members as there is no authentication in the front or back end. Once submit is entered, the role of bob changes to administrator.

user_id	fullname	email	user_password	role_id
100	rita	rita@designer.com	$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg...	2
101	robert	robert@admin.com	$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg...	1
102	bob	bob@designer.com	$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg...	1
103	braddy	braddy@designer.com	$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg...	2
104	josh	josh@designer.com	$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg...	2

As seen in the database, the role id of bob@designer.com has been changed to 1 from 2, making him an administrator in the system.

Code Fixes for Vulnerability 6

Original Codes	Fixed Codes
backEnd > src > routes.js	
	
Before: No middleware function was added to verify the user's identity	Added middleware function to check user's session validity
Original Codes	**Fixed Codes**
backEnd > controllers > authController.js	
	
	Added user role into the token so that it is accessible for all functions that require user role authentication

backEnd > src > middlewares > checkUserFnSolution.js

```
src > middlewares > JS checkUserFnSolution.js > ...
const config = require('../config/config');
const jwt = require('jsonwebtoken');
module.exports.checkForValidUserRoleUser = (req, res, next) => {
    //If the token is valid, the logic extracts the user id and the role information.
    //If the role is not user, then response 403 UnAuthorized
    //The user id information is inserted into the request.body.userId
        console.log('http header - user ', req.headers['user']);
        if (typeof req.headers.authorization !== "undefined") {
            // Retrieve the authorization header and parse out the
            // JWT using the split function
            let token = req.headers.authorization.split(' ')[1];
            //console.log('Check for received token from frontend : \n');
            //console.log(token);
            jwt.verify(token, config.JWTKey, (err, data) => {
                console.log('data extracted from token \n',data);
                if (err) {
                    console.log(err);
                    return res.status(403).send({ message: 'Unauthorized access' });
                }
                else {
                    req.body.userId = data.id;
                    next();
                }
            })

      }else{
        res.status(403).send({ message: 'Unauthorized access' });

      }
    } //End of checkForValidUserRoleUser
```

```
src > middlewares > JS checkUserFnSolution.js > checkForValidUserRoleUser > checkForValidUserRoleUser > jwt.
const config = require('../config/config');
const jwt = require('jsonwebtoken');
const { send } = require('process');
const { json } = require('body-parser');
module.exports.checkForValidUserRoleUser = (req, res, next) => {
    console.log("============================================================");
    console.log("checkUserFnSolution.js is called and running!");
    console.log("============================================================");

    //If the token is valid, the logic extracts the user id and the role information.
    //If the role is not user, then response 403 UnAuthorized
    //The user id information is inserted into the request.body.userId
    console.log('http header - user: ', req.headers['user']);
    if (typeof req.headers.authorization !== "undefined") {
        // Retrieve the authorization header and parse out the
        // JWT using the split function
        let token = req.headers.authorization.split(' ')[1];
        //console.log('Check for received token from frontend : \n');
        //console.log(token);
        jwt.verify(token, config.JWTKey, (err, data) => {
            console.log('data extracted from token \n', data);
            if (err) {
                console.log(err);

                let jsonResult = {
                    message: 'Unauthorized access'
                };
                return res.status(403).send(jsonResult);
            }
            else {
                // console.log("---------------------------------");
                // console.log("data id: " + data.id);
                // console.log("data uRole: " + data.uRole);
                // console.log("---------------------------------");

                req.decodedId = data.id;
                req.decodedRole = data.uRole

                // console.log("\n---------------------------------");
                // console.log("req.decodedId: " + req.decodedId);
                // console.log("req.decodedRole: " + req.decodedRole);
                // console.log("---------------------------------");
                next();
            }
        })

    } else {
        console.log("here!!!!");
        var jsonResult = {
            message: 'Unauthorized access'
        };
        res.status(403).send(jsonResult);

    }
} //End of checkForValidUserRoleUser
```

Added a req.decodedRole to store the role that is retrieved from the token

Original Codes	Fixed Codes

backEnd > src > middlewares > accessCheckFrontEnd.js

N/A

```
src > middlewares > JS accessCheckFrontEnd.js > ...
module.exports.accessCheckFrontEnd = (req, res, next) => {
    // If this middleware function is called, it will pass the data that was stored previously
    // back to the FrontEnd.

    // This module is just a lightweight module that passes the information back to FrontEnd and
    // nothing else
    console.log("============================================");
    console.log("accessCheckFrontEnd.js is called and running!");
    console.log("============================================");
    console.log("");

    //console.log("req.decodedId: " + req.decodedId);
    //console.log("req.decodedRole: " + req.decodedRole);

    var sendBack = {
        'userId': req.decodedId,
        'userRole': req.decodedRole
    };

    res.status(200).json(sendBack);

} //End of accessCheckFrontEnd
```

Added a new middleware function that will pass the data back to the frontEnd if the frontEnd needs the data to do page authentication

frontEnd > public > js > admin_manage_user.js

```
public > js > admin_manage_user.js > on('click') callback > then() callback
let $searchDesignFormContainer = $('#searchUserFormContainer');
if ($searchDesignFormContainer.length != 0) {
    console.log('Search user form detected in manage user interface. Binding event handling logic to form elements.');
    //If the jQuery object which represents the form element exists,
    //the following code will create a method to submit search parameters
    //to server-side api when the #submitButton element fires the click event.
    $('#submitButton').on('click', function(event) {
        event.preventDefault();
        const baseUrl = 'http://localhost:5000';
        let searchInput = $('#searchInput').val();
        let userId = localStorage.getItem('user_id');
        axios({
                headers: {
                    //Modify this will affect the checkUserFn.js middleware file at the backend.
                    'user': userId,
                },
                method: 'get',
                url: baseUrl + '/api/user/process-search-user/1/' + searchInput,
            })
            .then(function(response) {
                //Using the following to inspect the response.data data structure ···
```

```
public > js > admin_manage_user.js > on('click') callback > then() callback
1  let $searchDesignFormContainer = $('#searchUserFormContainer');
2  if ($searchDesignFormContainer.length != 0) {
3      console.log('Search user form detected in manage user interface. Binding event handling logic to form elements.');
4      //If the jQuery object which represents the form element exists,
5      //the following code will create a method to submit search parameters
6      //to server-side api when the #submitButton element fires the click event.
7      $('#submitButton').on('click', function(event) {
8          event.preventDefault();
9          const baseUrl = 'http://localhost:5000';
10         let searchInput = $('#searchInput').val();
11         let userId = localStorage.getItem('user_id');
12         let token = localStorage.getItem('token');
13         axios({
14                 headers: {
15                     //Modify this will affect the checkUserFn.js middleware file at the backend.
16                     'user': userId,
17                     authorization: "Bearer " + token
18                 },
19                 method: 'get',
20                 url: baseUrl + '/api/user/process-search-user/1/' + searchInput,
21             })
22             .then(function(response) {
```

Added Line 12 and Line 17 to enable authorization check for the webpage by sending the token to the endpoint

frontEnd > public > js > admin_manage_user_submission.js

```
public > js > admin_check_user_submission.js > on('click') callback > then() callback
1
2  let $searchUserFormContainer = $('#searchUserFormContainer');
3  if ($searchUserFormContainer.length != 0) {
4      console.log('Search user form detected in user manage submission interface. Binding event handling logic to form element:
5      //If the jQuery object which represents the form element exists,
6      //the following code will create a method to send key-value pair information to do record searching
7      //to server-side api when the #submitButton element fires the click event.
8      $('#submitButton').on('click', function(event) {
9          event.preventDefault();
10         const baseUrl = 'http://localhost:5000';
11         let searchInput = $('#searchInput').val();
12         let userId = localStorage.getItem('user_id');
13         axios({
14                 headers: {
15                     'user': userId
16                 },
17                 method: 'get',
18                 url: baseUrl + '/api/user/process-search-user-design/1/' + searchInput,
19             })
20             .then(function(response) {
```

```
public > js > admin_check_user_submission.js > ...
1
2  let $searchUserFormContainer = $('#searchUserFormContainer');
3  if ($searchUserFormContainer.length != 0) {
4      console.log('Search user form detected in user manage submission interface. Binding event handling logic to form elements.');
5      //If the jQuery object which represents the form element exists,
6      //the following code will create a method to send key-value pair information to do record searching
7      //to server-side api when the #submitButton element fires the click event.
8      $('#submitButton').on('click', function(event) {
9          event.preventDefault();
10         const baseUrl = 'https://localhost:5000';
11         let searchInput = $('#searchInput').val();
12         let userId = sessionStorage.getItem('user_id');
13         let token = sessionStorage.getItem('token');
14         axios({
15                 headers: {
16                     'user': userId,
17                     authorization: "Bearer " + token
18                 },
19                 method: 'get',
20                 url: baseUrl + '/api/user/process-search-user-design/1/' + searchInput,
21             })
22             .then(function(response) {
23                 //Using the following to inspect the response.data data structure
```

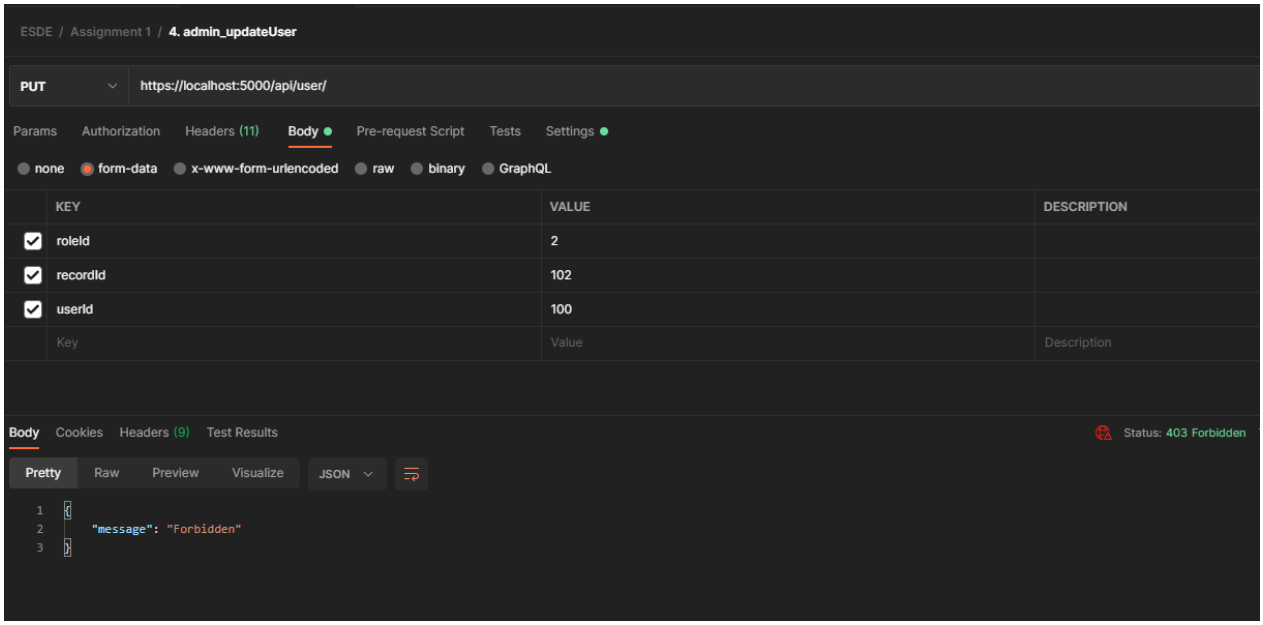
Added Line 13 and Line 17 to enable authorization check for the webpage by sending the token to the endpoint

backEnd > src > services > userService.js
(module.exports. getOneUserDataByEmail)

```
module.exports.getOneUserDataByEmail = function (search) {
    console.log('getOneUserDataByEmail method is called.');
    console.log('Prepare query to fetch one user record');
    userDataQuery = `SELECT user_id, fullname, email, user.role_id, role_name
        FROM user INNER JOIN role ON user.role_id = role.role_id WHERE email='` + search + `'`;

    return new Promise((resolve, reject) => {
        //I referred to https://www.codota.com/code/javascript/functions/mysql/Pool/getConnection
        //to prepare the following code pattern which does not use callback technique (uses Promise technique)
        pool.getConnection((err, connection) => {
            if (err) {
                console.log('Database connection error ', err);
                resolve(err);
            } else {
                connection.query(userDataQuery, (err, results) => {
                    if (err) {
                        reject(err);
                    } else {
                        resolve(results);
                    }
                    connection.release();
                });
            }
        });
    }); //End of new Promise object creation

} //End of getOneUserDataByEmail
```

```
module.exports.getOneUserDataByEmail = function (search) {
    console.log("=============================================================");
    console.log("userService.js > getOneUserDataByEmail is called and running!");
    console.log("=============================================================");

    console.log('Prepare query to fetch one user record');
    console.log("Search parameter: " + search);
    return new Promise((resolve, reject) => {
        //I referred to https://www.codota.com/code/javascript/functions/mysql/Pool/getConnection
        //to prepare the following code pattern which does not use callback technique (uses Promise technique)
        pool.getConnection((err, connection) => {
            if (err) {
                console.log('Database connection error ', err);
                resolve(err);
            } else {
                userDataQuery = `SELECT user_id, fullname, email, user.role_id, role_name FROM user INNER JOIN role ON user.role_id = role.role_id WHERE email = ?`;
                connection.query(userDataQuery, [search], (err, results) => {
                    if (err) {
                        reject(err);
                    } else {
                        resolve(results);
                    }
                    connection.release();
                });
            }
        });
    }); //End of new Promise object creation

} //End of getOneUserDataByEmail
```

Demo Screenshots (After)

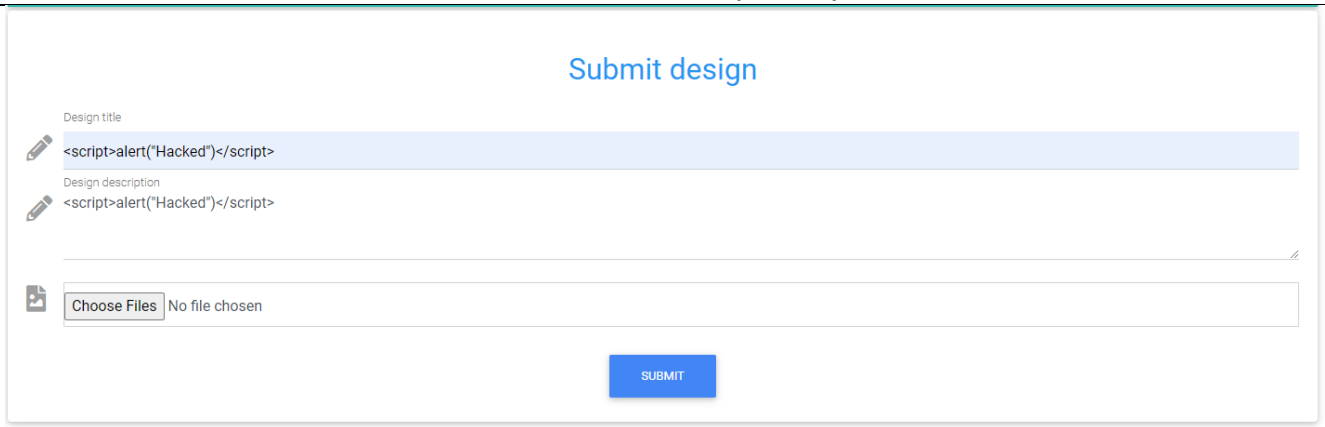


To prevent any user from querying someone else's data, we protected that endpoint such that the user is not able to change any user's role in the database if the user is not authorized

Vulnerability 7 – Cross-site Scripting Attacks (XSS)

V-05 Cross-site Scripting (XSS)	When a stored XSS attack occurs, the script will be called every time the html page is accessed. In this case when the administrator searches for the user's submission, the script will be called in the database and ran on the administrator page. Impact - High	Any user can simply submit a design or edit an existing submission with a script in the title or the description and submit it into the database. This script will then become a stored XSS attacked in the database and will target the administrators of the system. Ease of exploit - Easy	Recommend doing a regular expression(regex) check on the design title and design description input to remove any special characters to be inputted, making script injections impossible.

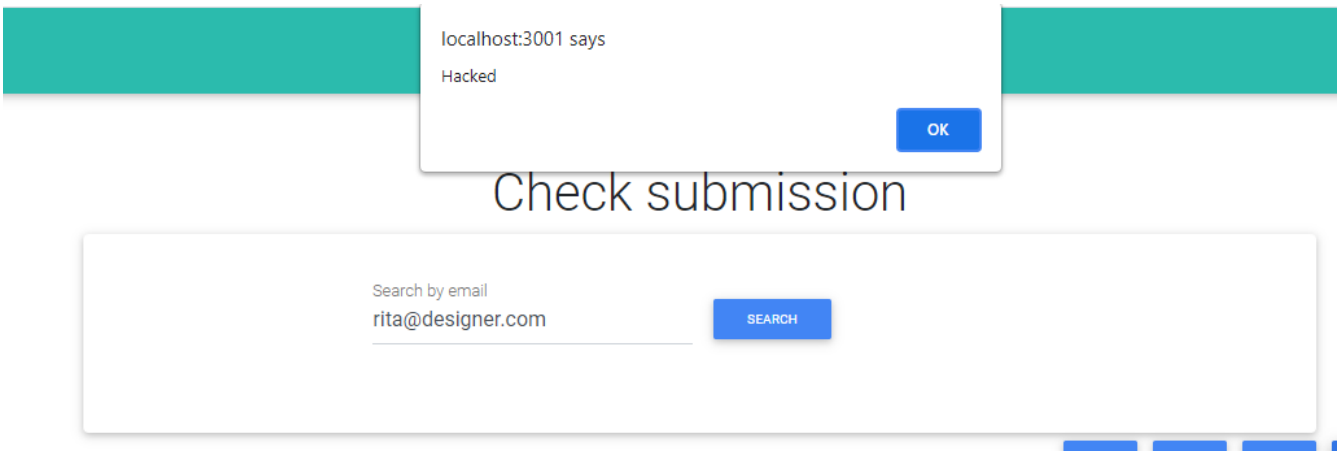
Demo Screenshot (Before)



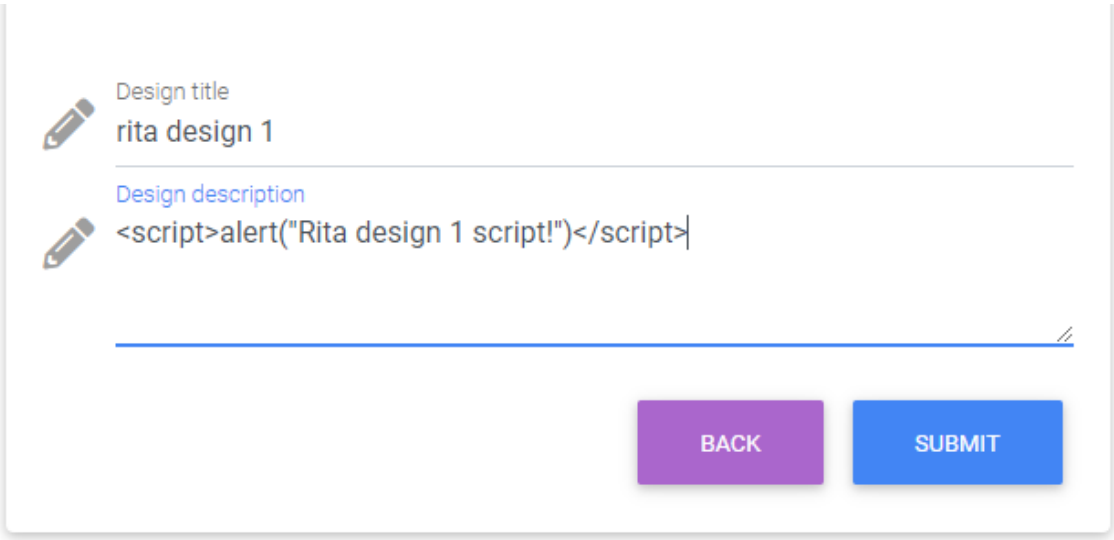
In submit_design.html, a script can be inserted into the fields of design title and design description and will be stored in the database when submitted.

file_id	cloudinary_file_id	cloudinary_url	design_title	design_description	created_by_id
105	Design/zrle55auuz9zkv25k4cw	http://res.cloudinary.com/deuevi29r/image/upl...	rita design 6	rita design 6 description text 1 text 2 text 3 tex...	100
106	Design/zruzta6htdsggeltumxp	http://res.cloudinary.com/deuevi29r/image/upl...	rita design 7	rita design 7 description text 1 text 2 text 3 tex...	100
107	Design/ujhymyes5yp1qdtowokq	http://res.cloudinary.com/deuevi29r/image/upl...	rita design 8	rita design 8 description text 1 text 2 text 3 tex...	100
108	Design/ku4htozthqb5ainmzajk	http://res.cloudinary.com/deuevi29r/image/upl...	rita design 9	rita design 9 description text 1 text 2 text 3 tex...	100
109	Design/eswqxolryiysoo7gdvui	http://res.cloudinary.com/deuevi29r/image/upl...	rita design 10	rita design 10 description text 1 text 2 text 3 te...	100
110	Design/t23po97jwj8nfksbeymi	http://res.cloudinary.com/deuevi29r/image/upl...	rita design 11	rita design 11 description text 1 text 2 text 3 te...	100
111	Design/ouxgn5ursdmbf1hridrj	http://res.cloudinary.com/deuevi29r/image/upl...	rita design 12	rita design 12 description text 1 text 2 text 3 te...	100
112	Design/wxe1k2hs0wfjg7fdg2mr	http://res.cloudinary.com/deuevi29r/image/upl...	asdads	adsasdsa	100
113	Design/qdoh94bzxcepypft8uta	http://res.cloudinary.com/deuevi29r/image/upl...	asdads	adsasdsa	100
114	Design/fki2e1jg9tvg4rkztaaq	http://res.cloudinary.com/deuevi29r/image/upl...	asdads	DELETE *	100
115	Design/skj7acegugtexlvsddei	http://res.cloudinary.com/deuevi29r/image/upl...	<script>alert("Hacked")</script>	<script>alert("Hacked")</script>	100
116	Design/zaotexn3jql5arelzjwz	http://res.cloudinary.com/deuevi29r/image/upl...	<script>alert("Hacked")</script>	asdasdasd" DELETE *	100

As seen in the database, the script is stored in the design title and description.



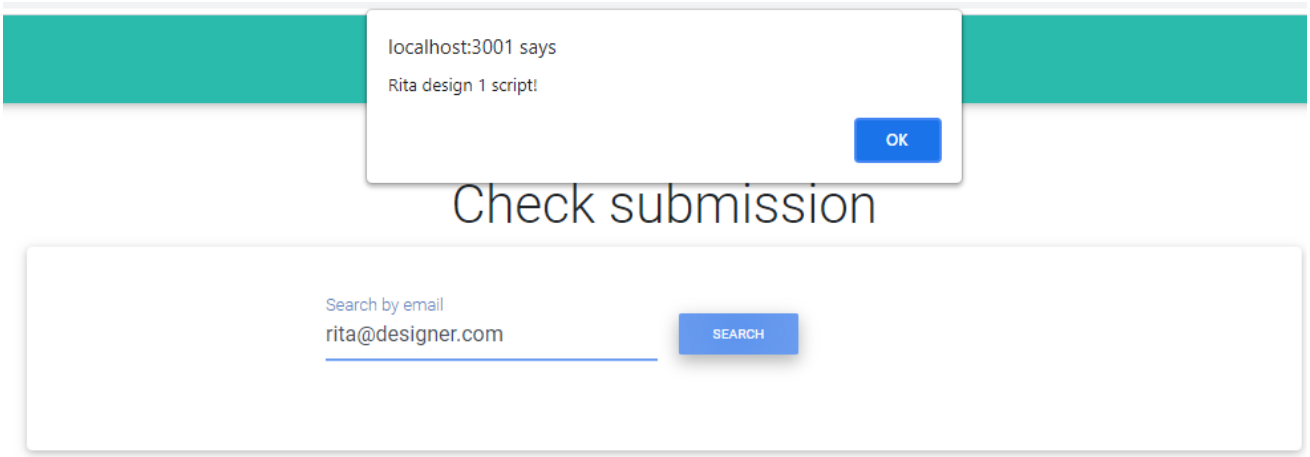
When the administrator checks the submission of rita, the script will then load from the database and display the script, harming the administrator.



As for update deisgn, inserting a script also works for the design title and design description.

file_id	cloudinary_file_id	cloudinary_url	design_title	design_description	created_by_id
100	Design/gz0g0sdy3hwgqhpgnqvf	http://res.cloudinary.com/deuevi29r/image/upl...	rita design 1	<script>alert("Rita design 1 script!")</script>	100
101	Design/yjevf0kf5ryv4iyf290i	http://res.cloudinary.com/deuevi29r/image/upl...	rita design 2	rita design 2 description text 1 text 2 text 3 tex...	100

In the database, the design description of rita design 1 is the script from the update design



And when the administrator checks rita submission, the script is ran from the database.

<table>
<tr><th>Original Codes</th><th>Fixed Codes</th></tr>
<tr><td colspan="2">frontEnd > public > js > submit_design.js</td></tr>
<tr><td>

```
let $submitDesignFormContainer = $('#submitDesignFormContainer');
if ($submitDesignFormContainer.length != 0) {
    console.log('Submit design form detected. Binding event handling logic to form elements.');
    //If the jQuery object which represents the form element exists,
    //the following code will create a method to submit design details
    //to server-side api when the #submitButton element fires the click event.
    $('#submitButton').on('click', function(event) {
        event.preventDefault();
        const baseUrl = 'http://localhost:5000';
        let userId = localStorage.getItem('user_id');
        let designTitle = $('#designTitleInput').val();
        let designDescription = $('#designDescriptionInput').val();
        let webFormData = new FormData();
        webFormData.append('designTitle', designTitle);
        webFormData.append('designDescription', designDescription);
        // HTML file input, chosen by user
        webFormData.append("file", document.getElementById('fileInput').files[0]);
        axios({
                method: 'post',
                url: baseUrl + '/api/user/process-submission',
                data: webFormData,
                headers: { 'Content-Type': 'multipart/form-data', 'user': userId }
            })
            .then(function(response) {
                Noty.overrideDefaults({
                    callbacks: {
                        onTemplate: function() {
                            if (this.options.type === 'systemresponse') {
                                this.barDom.innerHTML = '<div class="my-custom-template noty_body">';
                                this.barDom.innerHTML += '<div class="noty-header">Message</div>';
                                this.barDom.innerHTML += '<p class="noty-message-body">' + this.options.text + '</p>';
                                this.barDom.innerHTML += '<img src="' + this.options.imageURL + '">';
                                this.barDom.innerHTML += '<div>';
                            }
                        }
                    }
                })
```

</td><td>

```
let $submitDesignFormContainer = $('#submitDesignFormContainer');
if ($submitDesignFormContainer.length != 0) {
    console.log('Submit design form detected. Binding event handling logic to form elements.');
    //If the jQuery object which represents the form element exists,
    //the following code will create a method to submit design details
    //to server-side api when the #submitButton element fires the click event.
    $('#submitButton').on('click', function (event) {
        event.preventDefault();
        const baseUrl = 'https://localhost:5000';
        let userId = sessionStorage.getItem('user_id');
        let designTitle = $('#designTitleInput').val();
        let designDescription = $('#designDescriptionInput').val();
        let webFormData = new FormData();
        let token = sessionStorage.getItem('token');

        // regex check to ensure that the designTitle and designDescription does not contain malicious codes
        let designTitle_regex = new RegExp('^[a-zA-Z0-9 ,!]+$');
        let designDescription_regex = new RegExp('^[a-zA-Z0-9 ,!]+$');

        if (designTitle_regex.test(designTitle) && designDescription_regex.test(designDescription)) {
            webFormData.append('designTitle', designTitle);
            webFormData.append('designDescription', designDescription);
            // HTML file input, chosen by user
            webFormData.append("file", document.getElementById('fileInput').files[0]);
            axios({
                method: 'post',
                url: baseUrl + '/api/user/process-submission',
                data: webFormData,
                headers: {
                    'Content-Type': 'multipart/form-data',
                    'user': userId,
                    authorization: 'Bearer ' + token
                }
            })
```

Added regular expression checks on all input field by the user to ensure that the input is clean and does not contain malicious codes

</td></tr>
<tr><td colspan="2">backEnd > src > controllers > userController.js
(exports.processDesignSubmission)</td></tr>
<tr><td>

```
exports.processDesignSubmission = (req, res, next) => {
    let designTitle = req.body.designTitle;
    let designDescription = req.body.designDescription;
    let userId = req.body.userId;
    let file = req.body.file;
    fileDataManager.uploadFile(file, async function(error, result) {
        console.log('check result variable in fileDataManager.upload code block\n', result);
        console.log('check error variable in fileDataManager.upload code block\n', error);
        let uploadResult = result;
        if (error) {
            let message = 'Unable to complete file submission.';
            res.status(500).json({ message: message });
            res.end();
        } else {
```

</td><td>

```
exports.processDesignSubmission = (req, res, next) => {
    console.log("============================================================");
    console.log("userController.js > processDesignSubmission is called and running!");
    console.log("============================================================");

    let designTitle = req.body.designTitle;
    let designDescription = req.body.designDescription;
    var userId = req.body.userId;
    let file = req.body.file;

    // ============== Start of Changes ======================
    let decodedId = req.decodedId;
    userId = checkUserID(userId, decodedId);

    //regex check to ensure that the designTitle and designDescription does not contain malicious codes
    let designTitle_regex = new RegExp('^[a-zA-Z0-9 ,!]+$');
    let designDescription_regex = new RegExp('^[a-zA-Z0-9 ,!]+$');

    //if regex passes, then store the files in cloudinary
    if (designTitle_regex.test(designTitle) && designDescription_regex.test(designDescription)) {
        fileDataManager.uploadFile(file, async function (error, result) {
            console.log('check result variable in fileDataManager.upload code block\n', result);
            console.log('check error variable in fileDataManager.upload code block\n', error);
            let uploadResult = result;
            if (error) {
                let jsonResult = {
                    message: 'Unable to complete file submission'
                };
                res.status(500).json(jsonResult);
                res.end();
            } else {
```

Added regular expression checks on all input field in the backEnd, just in case the user has JavaScript disabled, or it is a Man-in-the-middle attacker which bypasses frontEnd validation checks, to ensure that the input is clean and does not contain malicious codes

</td></tr>
</table>

<table>
<tr><th>Original Codes</th><th>Fixed Codes</th></tr>
<tr><td colspan="2">frontEnd > public > js > update_design.js</td></tr>
<tr><td>

```
public > js > JS update_design.js > ...
let $updateDesignFormContainer = $('#updateDesignFormContainer');
if ($updateDesignFormContainer.length != 0) {
    console.log('Update Design form is detected. Binding event handling logic to form ele
    //If the jQuery object which represents the form element exists,
    //the following code will create a method to submit design details
    //to server-side api when the #submitButton element fires the click event.
    $('#submitButton').on('click', function(event) {
        event.preventDefault();
        const baseUrl = 'http://localhost:5000';
        //Collect fileId value from the input element, fileIdInput (hidden input element)
        let fileId = $('#fileIdInput').val();
        //Obtain user id from local storage
        let userId = localStorage.getItem('user_id');
        //Collect design title and description input
        let designTitle = $('#designTitleInput').val();
        let designDescription = $('#designDescriptionInput').val();
        //Create a FormData object to build key-value pairs of information before
        //making a PUT HTTP request.
        let webFormData = new FormData();
        webFormData.append('designTitle', designTitle);
        webFormData.append('designDescription', designDescription);
        webFormData.append('fileId', fileId);
        axios({
                method: 'put',
                url: baseUrl + '/api/user/design/',
                data: webFormData,
                headers: { 'Content-Type': 'multipart/form-data', 'user': userId }
            })
            .then(function(response) {
                new Noty({
                    type: 'success',
                    layout: 'topCenter',
                    theme: 'sunset',
                    timeout: '5000',
                    text: 'Updated design information.',
                }).show();
            })
```

</td><td>

```
public > js > JS update_design.js > ⚲ on('click') callback
let $updateDesignFormContainer = $('#updateDesignFormContainer');
if ($updateDesignFormContainer.length != 0) {
    console.log('Update Design form is detected. Binding event handling logic to form elements.');
    //If the jQuery object which represents the form element exists,
    //the following code will create a method to submit design details
    //to server-side api when the #submitButton element fires the click event.
    $('#submitButton').on('click', function (event) {
        event.preventDefault();
        const baseUrl = 'https://localhost:5000';
        //Collect fileId value from the input element, fileIdInput (hidden input element)
        let fileId = $('#fileIdInput').val();
        //Obtain user id from local storage
        let userId = sessionStorage.getItem('user_id');
        //Collect design title and description input
        let designTitle = $('#designTitleInput').val();
        let designDescription = $('#designDescriptionInput').val();

        // -------------------- Start of changes ----------------------
        let designTitle_regex = new RegExp('^[a-zA-Z0-9 ,!]+$');
        let designDescription_regex = new RegExp('^[a-zA-Z0-9 ,!]+$');

        if (designTitle_regex.test(designTitle) && designDescription_regex.test(designDescription)) {
            //Create a FormData object to build key-value pairs of information before
            //making a PUT HTTP request.
            let webFormData = new FormData();
            webFormData.append('designTitle', designTitle);
            webFormData.append('designDescription', designDescription);
            webFormData.append('fileId', fileId);

            // -------------------- Start of changes ----------------------
            let token = sessionStorage.getItem('token');
            axios({
```

Added regular expression checks on all input field by the user to ensure that the input is clean and does not contain malicious codes

</td></tr>
<tr><td colspan="2">backEnd > src > controllers > userController.js
(exports.processUpdateOneDesign)</td></tr>
<tr><td>

```
exports.processUpdateOneDesign = async(req, res, next) => {
    console.log('processUpdateOneFile running');
    //Collect data from the request body
    let fileId = req.body.fileId;
    let designTitle = req.body.designTitle;
    let designDescription = req.body.designDescription;
    try {
        results = await userManager.updateDesign(fileId, designTitle, designDescription);
        console.log(results);
        return res.status(200).json({ message: 'Completed update' });
    } catch (error) {
        console.log('processUpdateOneUser method : catch block section code is running');
        console.log(error, '=================================================================');
        return res.status(500).json({ message: 'Unable to complete update operation' });
    }


}; //End of processUpdateOneDesign
```

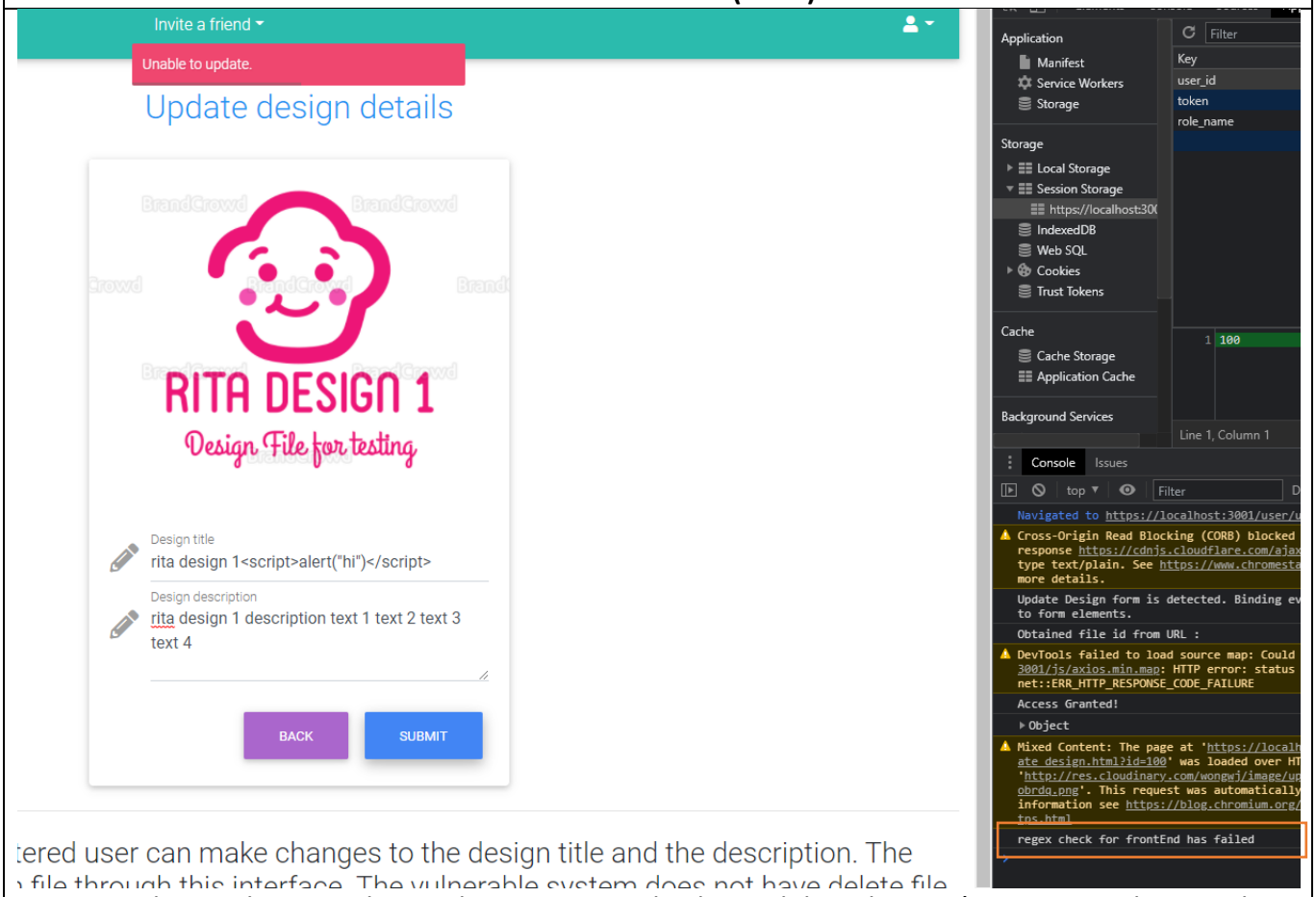
</td><td>

```
exports.processUpdateOneDesign = async (req, res, next) => {
    console.log("============================================================");
    console.log("userController.js > processUpdateOneDesign is called and running!");
    console.log("============================================================");

    //Collect data from the request body
    let fileId = req.body.fileId;
    let designTitle = req.body.designTitle;
    let designDescription = req.body.designDescription;

    // ============== Start of Changes ======================
    let userId = req.body.userId;
    let decodedId = req.decodedId;

    if (userId != decodedId) {···
    }

    // regex check to ensure that it will not accept any unclean inputs from the user
    let dTitle_regex = new RegExp('^[a-zA-Z0-9 .]*$');
    let dDesc_regex = new RegExp('^[a-zA-Z0-9 .]*$');

    console.log("fileId: " + fileId);
    //console.log(dTitle_regex.test(designTitle));
    //console.log(dDesc_regex.test(designDescription));

    // below if check ensures that both regex is successful before updating
    if (dTitle_regex.test(designTitle) && dDesc_regex.test(designDescription)) {
        try {
            userCheck = await userManager.getUserIdFromFileId(fileId);
            // console.log("userCheck: " + userCheck);
            if (userCheck != decodedId) {
                console.log("The TokenID does not match the UserID of the file that needs updating");
                console.log("Terminating update operation");
                let jsonResult = {
                    message: 'Unable to complete update operation'
                }
                return res.status(500).json(jsonResult);
            } else {
```

Added regular expression checks on all input field in the backEnd, just in case the user has JavaScript disabled, or it is a Man-in-the-middle attacker which bypasses frontEnd validation checks,to ensure that the input is clean and does not contain malicious codes

</td></tr>
</table>

Demo Screenshot (After)

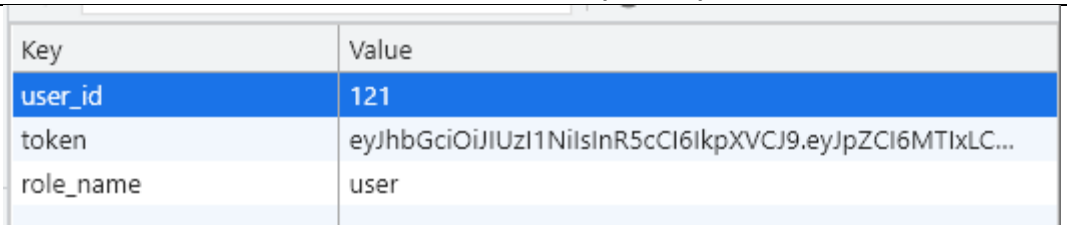


In our codes, we have used a regular expression check to validate the user's input to make sure that there is no chance of storing a script in the database. In this screenshot, when the user tries to update any design title or design description that has special characters, the update will fail as it has failed the regular expression check.

Vulnerability 8 – Sensitive Data Exposure (localStorage)

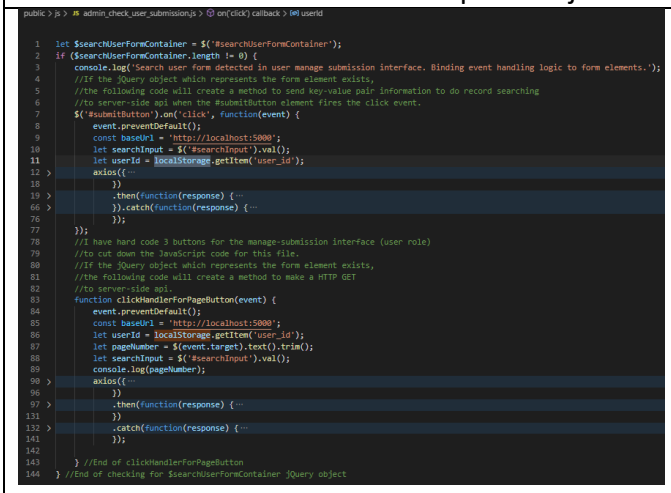
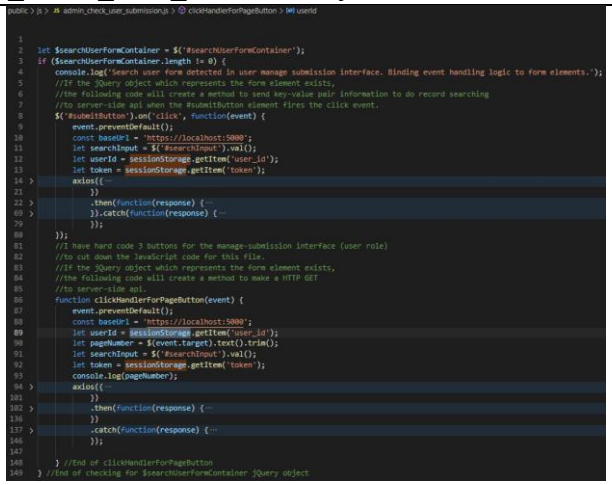
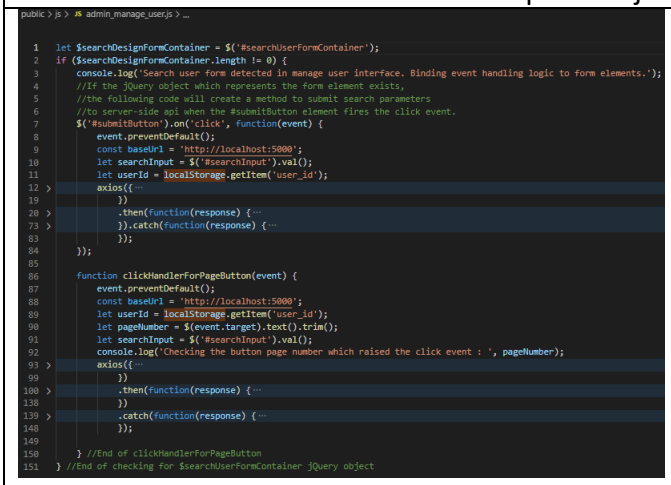
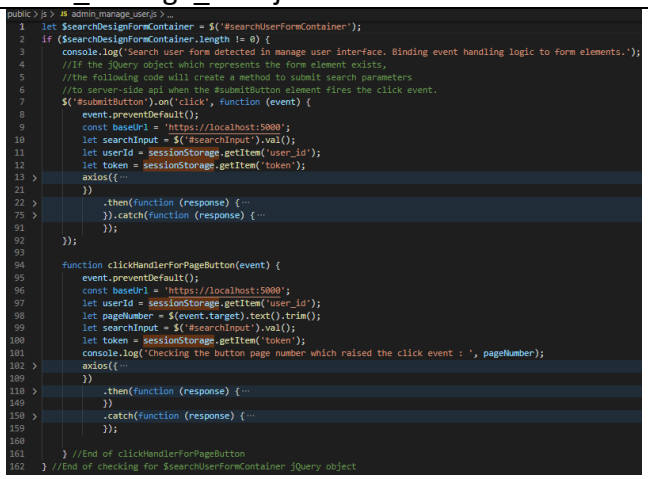
V-03 Sensitive Data Exposure (localstorage)	The downside of localStorage is that the data stored in localStorage does not expire, leaving their token to be targeted by hackers. The change to session storage alleviates this problem as the data in session storage is cleared when the session ends (e.g. Tab closed) Impact - Low	If hackers can gain access to the user's computer, their information might be unknowingly stolen from them as local storage will not be wiped when the browser is closed. They can then use their token to login with their account. Ease of exploit - Moderate	Recommend making use of sessionStorage instead of localStorage so that once the browser is closed, the data gets wiped from the browser rather than being persistent in the storage. Furthermore, the JWT token in the system is set to expire in 2 hours to strengthen security if the token is leaked.

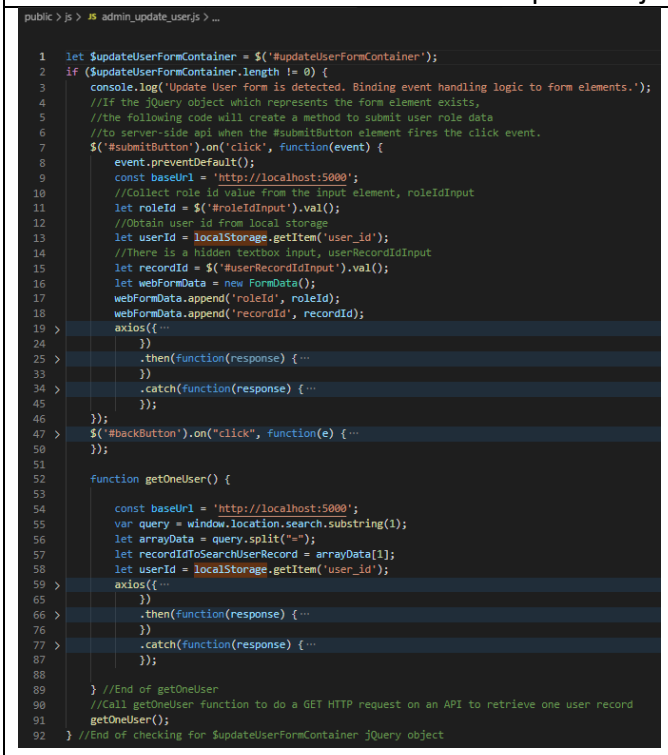
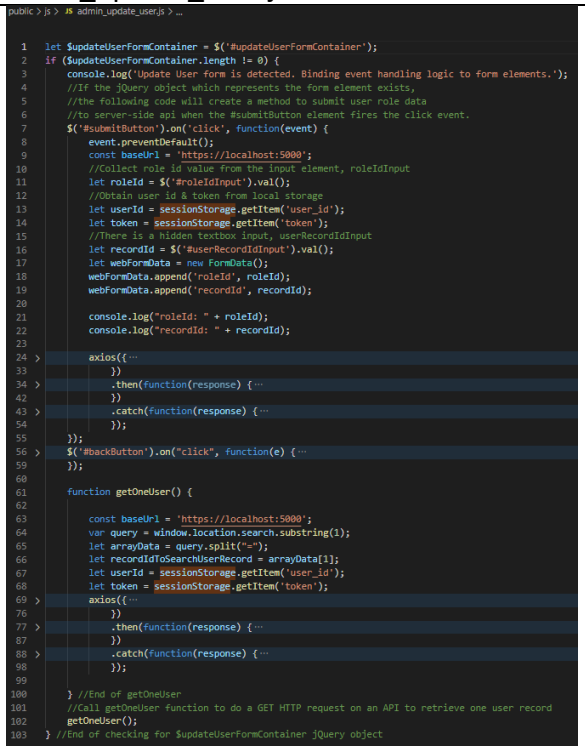
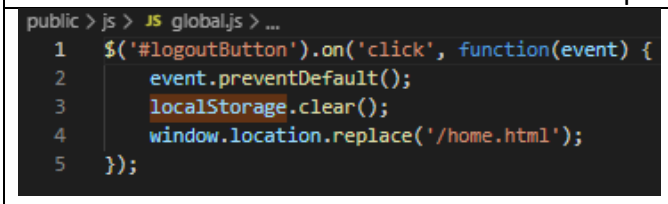
Demo Screenshot (Before)

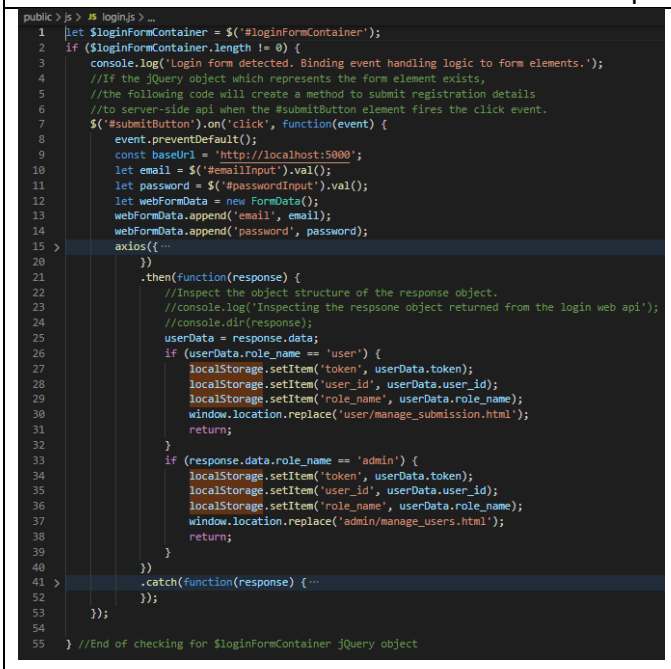
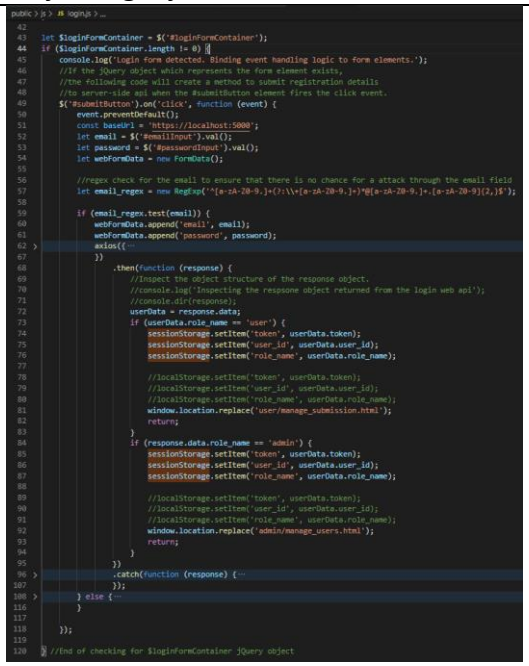
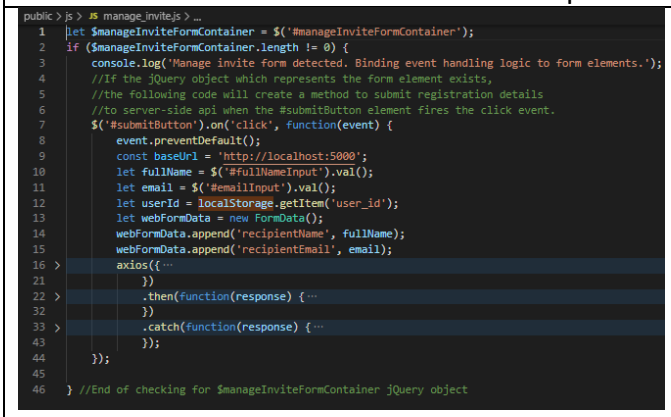
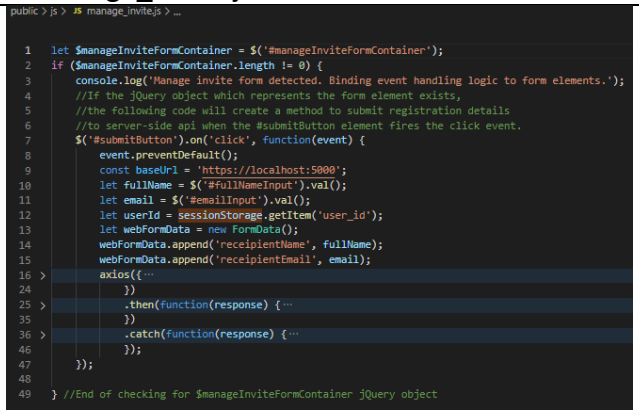
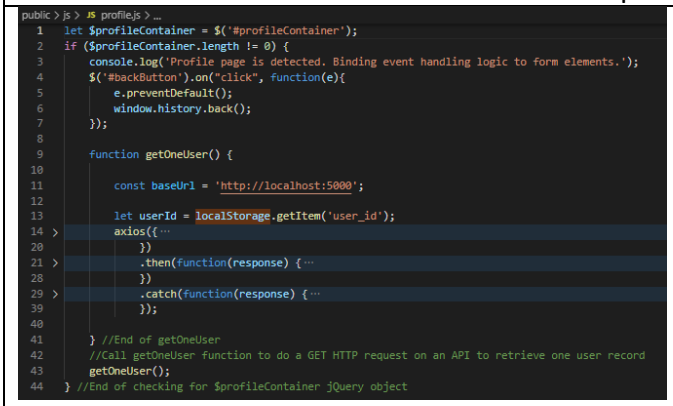
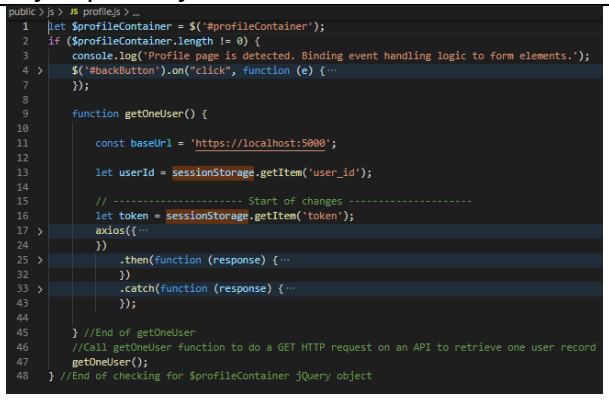


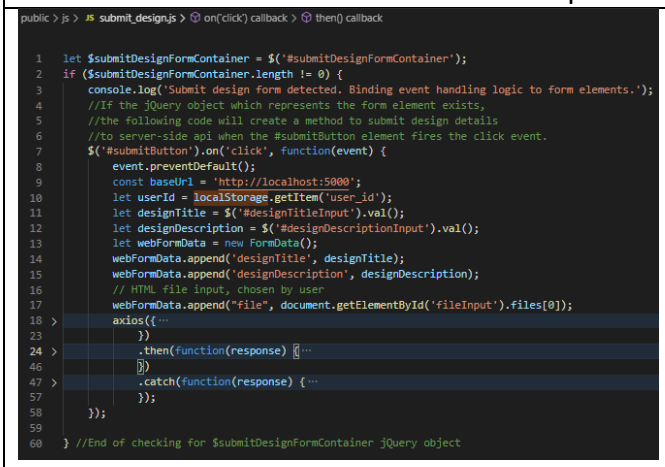
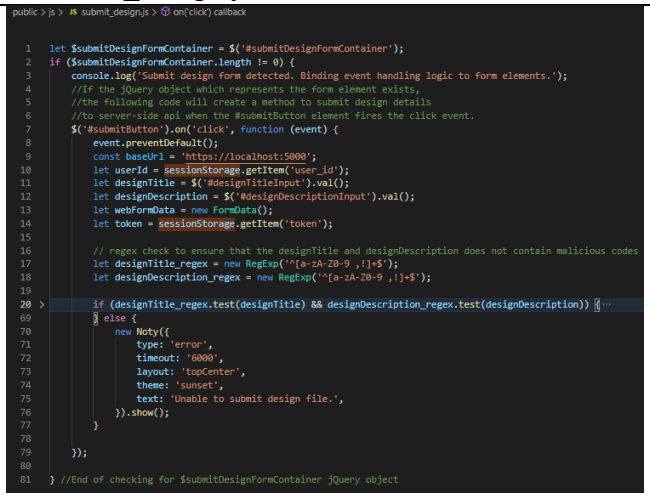
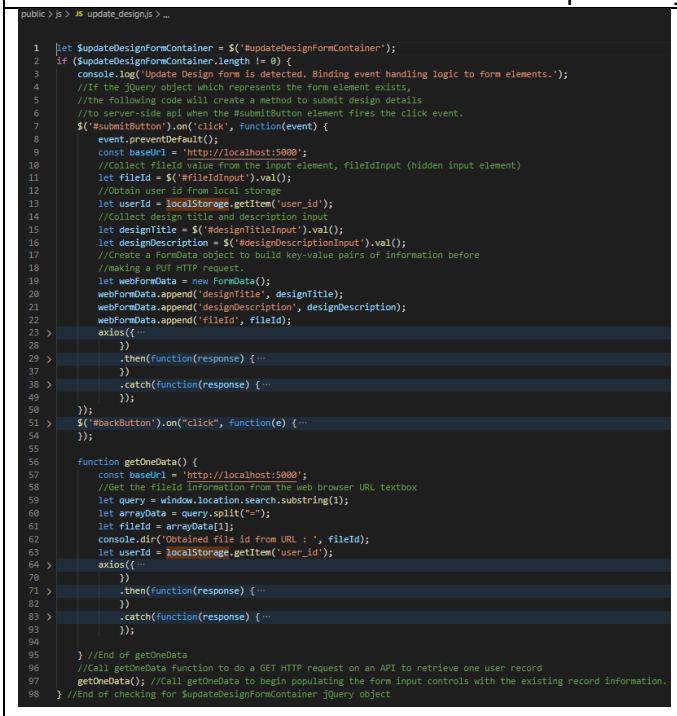
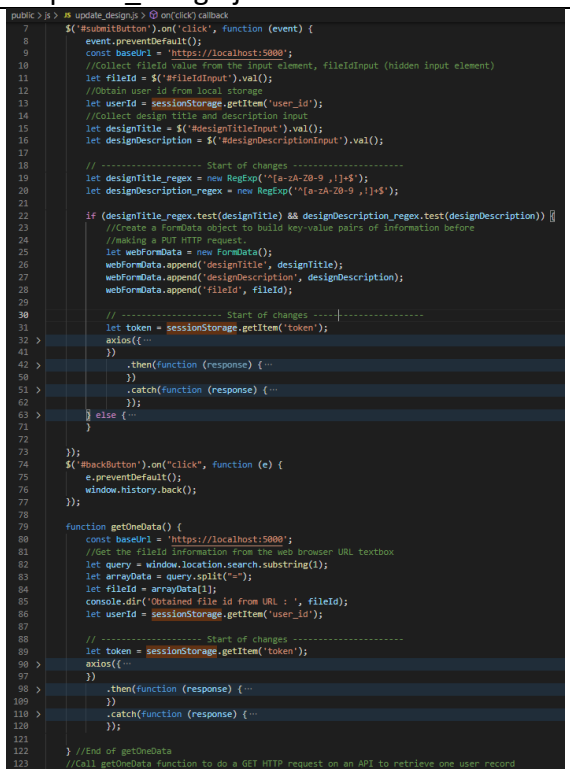
Website uses local storage to store the user_id, token and role_name, thus we have decided to switch over from localStorage to sessionStorage so that the storage will be cleared the moment the tab is closed while the data in localstorage will still be stored untill it is called to clear.

The code below shows the switch from localStorage to sessionStorage in the front end.

Original Codes	Fixed Codes
frontEnd > public > js > admin_check_user_submission.js	
	
frontEnd > public > js > admin_manage_user.js	
	

Original Codes	Fixed Codes
frontEnd > public > js > admin_update_user.js	
	
frontEnd > public > js > global.js	
	

Original Codes	Fixed Codes
frontEnd > public > js > login.js	
	
frontEnd > public > js > manage_invite.js	
	
frontEnd > public > js > profile.js	
	

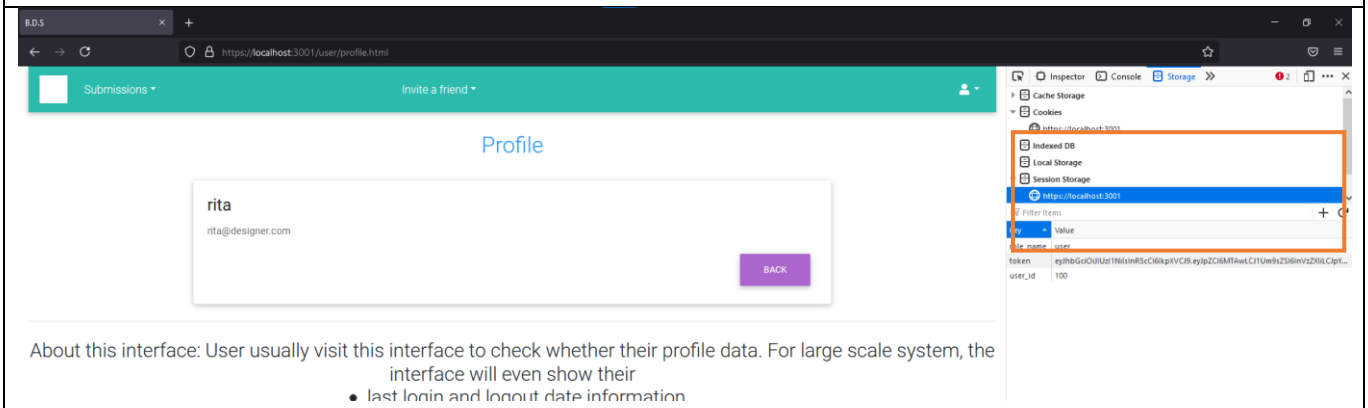
Original Codes	Fixed Codes
frontEnd > public > js > submit_design.js	
	
frontEnd > public > js > update_design.js	
	

Original Codes	Fixed Codes
frontEnd > public > js > user_manage_submission.js	

```
let $searchDesignFormContainer = $('#searchDesignFormContainer');
if ($searchDesignFormContainer.length != 0) {
    console.log('Search design form detected in user manage submission interface. Binding event handling logic to form elements.');
    //If the jQuery object which represents the form element exists,
    //the following code will create a method to send key-value pair information to do record searching
    //to server-side api when the #submitButton element fires the click event.
    $('#submitButton').on('click', function(event) {
        event.preventDefault();
        const baseUrl = 'http://localhost:5000';
        let searchInput = $('#searchInput').val();
        let userId = localStorage.getItem('user_id');
        axios({···
            })
            .then(function(response) {···
            }).catch(function(response) {···
            });
    });
    //I have hard code 3 buttons for the manage-submission interface (user role)
    //to cut down the JavaScript code for this file.
    //If the jQuery object which represents the form element exists,
    //the following code will create a method to make a HTTP GET
    //to server-side api.
    function clickHandlerForPageButton(event) {
        event.preventDefault();
        const baseUrl = 'http://localhost:5000';
        let userId = localStorage.getItem('user_id');
        let pageNumber = $(event.target).text().trim();
        let searchInput = $('#searchInput').val();
        console.log(pageNumber);
        axios({···
            })
            .then(function(response) {···
            })
            .catch(function(response) {···
            });

    } //End of clickHandlerForPageButton
} //End of checking for $searchDesignFormContainer jQuery object
```

```
let $searchDesignFormContainer = $('#searchDesignFormContainer');
if ($searchDesignFormContainer.length != 0) {
    console.log('Search design form detected in user manage submission interface. Binding event handling
    //If the jQuery object which represents the form element exists,
    //the following code will create a method to send key-value pair information to do record searching
    //to server-side api when the #submitButton element fires the click event.
    $('#submitButton').on('click', function(event) {
        event.preventDefault();
        const baseUrl = 'https://localhost:5000';
        let searchInput = $('#searchInput').val();
        let userId = sessionStorage.getItem('user_id');
        let token = sessionStorage.getItem('token');
        axios({···
            })
            .then(function(response) {···
            }).catch(function(response) {···
            });
    });
    //I have hard code 3 buttons for the manage-submission interface (user role)
    //to cut down the JavaScript code for this file.
    //If the jQuery object which represents the form element exists,
    //the following code will create a method to make a HTTP GET
    //to server-side api.
    function clickHandlerForPageButton(event) {
        event.preventDefault();
        const baseUrl = 'https://localhost:5000';
        let userId = sessionStorage.getItem('user_id');
        let pageNumber = $(event.target).text().trim();
        let searchInput = $('#searchInput').val();
        let token = sessionStorage.getItem('token');
        console.log(pageNumber);
        axios({···
            })
            .then(function(response) {···
            })
            .catch(function(response) {···
            });

    } //End of clickHandlerForPageButton
} //End of checking for $searchDesignFormContainer jQuery object
```

Demo Screenshot (After)

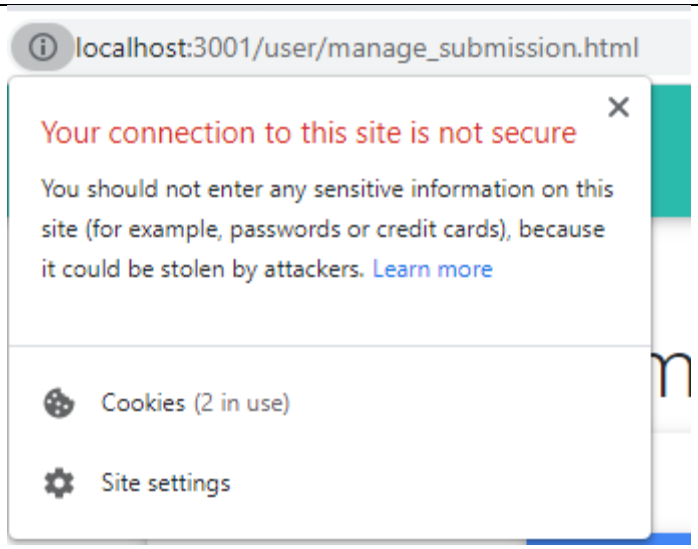


After the code changes, the role_name, token and user_id is now in the sessionStorage instead of localStorage.

Vulnerability 9 – Sensitive Data Exposure (HTTP)

V-03 Sensitive Data Exposure (HTTP)	The current website should use the newer HTTPS encryption as data passed from HTTPS should be encrypted, especially for login which contains sensitive data. This data can potentially be stolen. Impact - Low	Packet sniffers can intercept user's traffic and steal their information. This website uses the normal HTTP protocol which does not encrypt its traffic. However, this requires the hacker to be in the same network as the user to intercept the data and requires some experience. Ease of exploit - Difficult	Recommend changing to HTTPS with an SSL certificate from a Certificate authority (such as Cloudflare) so that user's information will be encrypted during traffic to prevent sensitive information from being stolen.

Demo Screenshots (Before)



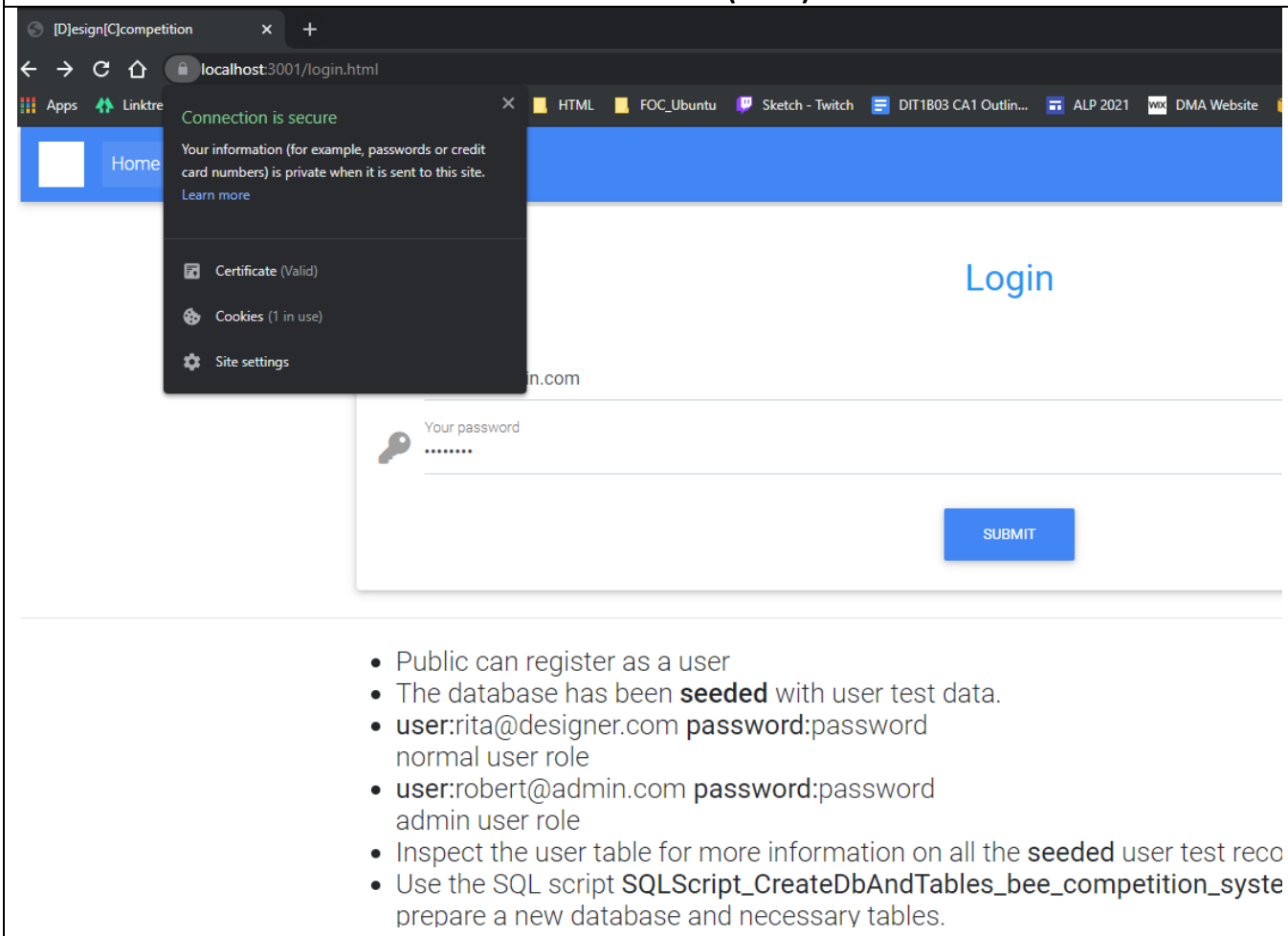
As seen in the screenshot above, Google warns users that the connection is not secure as the site does not have a certified SSL certificate as it still uses HTTP instead of HTTPS

```
Content-Disposition: form-data; name="email"

robert@admin.com
------WebKitFormBoundaryjpGgapIrfkJs1ddH
Content-Disposition: form-data; name="password"

password
------WebKitFormBoundaryjpGgapIrfkJs1ddH--
```

At the Login Page, When the user presses on the “Login” button, if intercepted with Burp, the username and password is shown in the HTTP request as plain text

Original Codes	Fixed Codes
`const baseUrl = 'http://localhost:5000';`	`const baseUrl = 'https://localhost:5000';`
Every single endpoint in the frontEnd codes that has a baseUrl pointing to HTTP websites will be switched to HTTPS instead	

Demo screenshot (After)



To fix that issue, we needed to switch all the webpages to use HTTPS instead of HTTP where S represents “Secure”. Hence, all the traffic that runs through HTTPS and HSTS (HTTP Strict-Transport Security) enforced such that the requests will not be prone to Man-in-the-middle attacks

Other Code Improvements – Adding Regular Expressions to Check Inputs

Original Codes	Fixed Codes
frontEnd > public > js > user_manage_submission.js (exposts.processLogin)	

```
exports.processLogin = (req, res, next) => {

    let email = req.body.email;
    let password = req.body.password;

    try {
        auth.authenticate(email, function(error, results) {
            if (error) {
                //let message = 'Credentials are not valid.';
                //return res.status(500).json({ message: message });
                //If the following statement replaces the above statement
                //to return a JSON response to the client, the SQLMap or
                //any attacker (who relies on the error) will be very happy
                //because they relies a lot on SQL error for designing how to do
                //attack and anticipate how much "rewards" after the effort.
                //Rewards such as sabotage (seriously damage the data in database),
                //data theft (grab and sell).
                //return res.status(500).json({ message: error });
                let data = {
                    message: 'Credentials are not valid'
                };
                return res.status(500).json(data);

            } else {
                if (results.length == 1) {
                    if ((password == null) || (results[0] == null)) {
                        return res.status(500).json({ message: 'login failed' });
                    }
                    if (bcrypt.compareSync(password, results[0].user_password) == true) {

                        let data = {
                            user_id: results[0].user_id,
                            role_name: results[0].role_name,
                            token: jwt.sign({ id: results[0].user_id, uRole: results[0].role_name }, config.JWTKey, {
                                expiresIn: 86400 //Expires in 24 hrs
                            })
                        }; //End of data variable setup

                        return res.status(200).json(data);
                    } else {
                        let data = {
                            message: 'Login has failed.'
                        }
                        return res.status(500).json(data);
                    } //End of password comparison with the retrieved decoded password.
                } //End of checking if there are returned SQL results

            }

        })
```

src > controllers > JS authController.js > processLogin > exports.processLogin

```
5   const jwt = require('jsonwebtoken');
6   const validator = require('validator');
7
8
9
10  exports.processLogin = (req, res, next) => {
11
12      let email = req.body.email;
13      let password = req.body.password;
14
15  >   if (validator.isEmail(email)) {···
69      } else {
70          console.log("\nregex check unsucessful in backend\n");
71          let data = {
72              message: 'Unable to complete login'
73          }
74          return res.status(500).json(data);
75      }
76
77
```

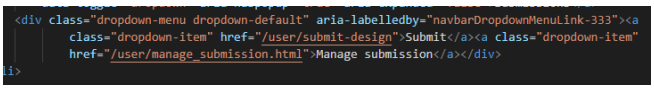
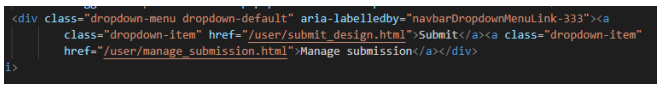
For all input fields that requires the data to be sent to the database for either querying of results or saving to the database, it must go through a few layers of check before proceeding to enter the database. If no checks are carried out, it makes the system very prone to all sorts of attacks

To ensure that the above layers of checks are enforced, regular expressions will be introduced for all input fields to test the input before proceeding to enter the database

If the regular expression for that particular input fails, it will immediately throw and error back to the client for them to re-enter their inputs and not proceed to tamper with the database

To ensure that this check is done thoroughly, the regular expressions will be checked in both frontEnd codes as well as backEnd codes so that in any case if the frontEnd codes fail to check, there is always a fallback checker which is the backEnd

Other Code Improvements – Wrong Redirection

Original Codes	Fixed Codes
frontEnd > public > user > update_design.html	
	
Before, it redirects to a empty page called submit-design.	After, it successfully redirects users to submit_design.html now

Other Code Improvements – Code Reusability

backEnd > src > middleware > userController.js

```
function checkUserID(userId, decodedId) {
    console.log("---------");
    console.log("Running checkUserID function inside userController.js");
    console.log("---------");
    console.log("userId: " + userId);
    console.log("decodedId: " + decodedId);

    if (userId != decodedId) {
        console.log("----------------");
        console.log("UserID does not match Token ID");
        console.log("Query/Upload can only be done with the same Token ID!");
        console.log("----------------");

        userId = decodedId;
        console.log("userID changed to Token ID");

        return userId;
    }

    return userId;
};
```

We have noticed that there was a repeat function that is being placed inside the functions which can take up a little time to re-write. Hence to improve code reusability, we have decided to create a function for this, such that when you pass in the values, it will correctly pass back the values required.

In this example, when you pass in the userId retrieved from the sessionStorage and the decodedId from the token, the function will check to see if the ID matches, if it does not match it will correct it and send the right value back to be used.

As this code does the same for all authorization checks, putting it in a function and just calling it will increase code reusability

Other Code Improvements – Salted Passwords

backEnd > src > controller > authController.js

```
bcrypt.genSalt(saltRounds, (err, salt) => {
    bcrypt.hash(password, salt, async (err, hash) => {
```

To Improve confidentiality as well as security, we have decided to salt all the passwords for an extra layer of security.

user_password
$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg33.WrxJx/FeC9.gCyYvIbs6
$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg33.WrxJx/FeC9.gCyYvIbs6
$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg33.WrxJx/FeC9.gCyYvIbs6
$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg33.WrxJx/FeC9.gCyYvIbs6
$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg33.WrxJx/FeC9.gCyYvIbs6
$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg33.WrxJx/FeC9.gCyYvIbs6
$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg33.WrxJx/FeC9.gCyYvIbs6
$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg33.WrxJx/FeC9.gCyYvIbs6
$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg33.WrxJx/FeC9.gCyYvIbs6
$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg33.WrxJx/FeC9.gCyYvIbs6
$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg33.WrxJx/FeC9.gCyYvIbs6
$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg33.WrxJx/FeC9.gCyYvIbs6
$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg33.WrxJx/FeC9.gCyYvIbs6
$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg33.WrxJx/FeC9.gCyYvIbs6
$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg33.WrxJx/FeC9.gCyYvIbs6
$2b$10$QLkBkffqSNaGQrNfRa.yTusM7HJVg6zZc.4c5syGOERxxXXQm3UzS

As shown in the database screenshot above, these passwords are only hashed and are not salted, hence they are all the same.

142	hashtest	hashtest@gmail.com	$2b$10$sf.AHTWCsRJCS1N7AQs2E.fan0BOCk1Y2/UNYICsXSOfD9WxtyMZC
144	hashtest2	hashtest2@gmail.com	$2b$10$4F/M6zw9Q93ir5AISC6wUekCYJKtRDzDh4Weh7vmjHTbnVEk3R6cC
145	hashtest3	hashtest3@gmail.com	$2b$10$PbZ2dQlkxHUBapVmsjO.xeEDN/p40kzPQ5L5SOc38MPdH6O292Fc6
146	hashtest4	hashtest4@test.com	$2b$10$s0/4qiE4dcGWz.mOBpbhT.PEPWKt4rYw/A6So0bIpDxz6NWXy7kqi
147	hashtest5	hashtest5@test.com	$2b$10$fJ1H42rwNJT//ELALOq7Zegp6KqgT/YDYmpbbhSiWYG3QCB1zdsae
148	hashtest6	hashtest6@test.com	$2b$10$pDjMQ4uYTawn40pUtbmqn.jauK/h047.FaDI28/h2GNYjTRg9xRZa
149	hashtest7	hashtest7@test.com	$2b$10$aQeAdkCBh.60s8oD6XttH.HTy2rcKt1bjkw5TAJid8fxnd3PbSwq6
150	hashtest8	hashtest8@test.com	$2b$10$OhCKYqA8nmMfDmSfOqXAjO7dRbJ4/oJkP7O4C3i76qVOcPnAsm1bi
151	hashtest8	hashtest9@test.com	$2b$10$pwAQ.OY3cy6QPLO1pRNGMeaOafJzmZm4D5NoxFQ0X7vm6otWgRSGC
152	hashtest9	hashtest9@gmail.com	$2b$10$WmSIE9B/uG2CxdwXJubUz.LQhDNytXeZVIHvdD4eWhS5BtPz3y/bu
153	hashte...	hashtest10@gmail.com	$2b$10$wugeCnj7hFmS9eLJAQq8luBVJGUbT/yyKR5YmqZqzG4yuWgSIYsZ.

In the new password creation process, the passwords are salted and kept in the database as seen above. In this example, all the hashtest users created have the same password: 'Passw0rd' entered, but their hashed and salted values make them appear differently inside the database table.

-- End of Report --