

6/26/2023

ESDE CA1 Report

Bee Design Website Vulnerability Report

Name: Tan Yong Kit, Eden

StudentID: P2243605

Class: DIT/FT/2A/06

Table of Contents

Executive Summary.....	9
1. Cross-Site Scripting	9
1.1 Definition.....	9
1.2 How to exploit vulnerability	10
Step 1:.....	10
Step 2:.....	10
Step 3:.....	11
Step 4:.....	11
Step 5:.....	12
Step 6:.....	12
Step 7:.....	13
Step 8:.....	13
1.3 How to rectify vulnerability.....	14
Step 1:.....	14
Step 2:.....	14
Step 3:.....	15
Step 4:.....	16
Step 5:.....	16
Step 6:.....	17
Step 7:.....	17
Step 8:.....	18
1.4 Evidence that the vulnerability has been rectified	19
Step 1:.....	19
Step 2:.....	19
Step 3:.....	20
Step 4:.....	20

2. Injection (SQL Injection)	21
2.1 Definition	21
2.2 How to exploit vulnerability	21
Step 1:.....	21
Step 2:.....	22
Step 3:.....	22
Step 4:.....	23
Step 5:.....	23
Step 6:.....	24
Step 7:.....	24
2.3 How to rectify vulnerability.....	25
Step 1:.....	25
Step 2:.....	25
Step 3:.....	26
Step 4:.....	26
Step 5:.....	27
Step 6:.....	27
2.4 Evidence that the vulnerability has been rectified	28
Step 1:.....	28
Step 2:.....	28
Step 3:.....	29
Step 4:.....	29
Step 5:.....	30
Step 6:.....	30
3. Broken Authentication	31
3.1 Definition	31
3.2 How to exploit vulnerability	31

Step 1:.....	31
Step 2:.....	32
Step 3:.....	33
3.3 How to rectify vulnerability.....	33
Step 1:.....	33
Step 2:.....	34
Step 3:.....	35
Step 4:.....	35
Step 5:.....	36
3.4 Evidence that the vulnerability has been rectified	37
Step 1:.....	37
Step 2:.....	37
Step 3:.....	38
4. Broken Access Control	39
4.1 Definition	39
4.3.1 How to exploit vulnerability - About Role access control	39
Step 1:.....	39
Step 2:.....	40
Step 3:.....	40
Step 4:.....	41
Step 5:.....	41
Step 6:.....	42
4.3.2 How to rectify vulnerability - About Role access control	42
Step 1:.....	42
Step 2:.....	43
Step 3:.....	43
Step 4:.....	44

Step 5:.....	45
Step 6:.....	46
Step 7:.....	47
Step 8:.....	48
4.4.2 Evidence that the vulnerability has been rectified - About Role access control	49
Step 1:.....	49
Step 2:.....	49
Step 3:.....	50
5. Sensitive Data Exposure	51
5.1 Definition	51
5.2 How to exploit vulnerability	51
Step 1:.....	51
Step 2:.....	52
Step 3:.....	52
Step 4:.....	53
Step 5:.....	53
Step 6:.....	54
Step 7:.....	55
5.3 How to rectify vulnerability.....	55
Step 1:.....	55
Step 2:.....	56
Step 3:.....	56
Step 4:.....	57
Step 5:.....	58
Step 6:.....	59
Step 7:.....	60
Step 8:.....	60

Step 9:	60
Step 10:	61
Step 11:	61
Step 12:	62
Step 13:	63
Step 14:	63
Step 15:	64
Step 16:	64
Step 17:	65
Step 18:	65
Step 19:	66
Step 20:	68
5.4 Evidence that the vulnerability has been rectified	69
Step 1:	69
Step 2:	69
Step 3:	70
Step 4:	70
Step 5:	71
Step 6:	72
6. Insufficient Logging & Monitoring	73
6.1 Definition	73
6.2 How to exploit vulnerability	73
Step 1:	73
Step 2:	74
6.3 How to rectify vulnerability	74
Step 1:	74
Step 2:	75

Step 3:.....	76
Step 4:.....	77
Step 5:.....	78
Step 6:.....	78
Step 7:.....	79
Step 8:.....	79
Step 9:.....	80
Step 10:.....	81
Step 11:.....	82
Step 12:.....	83
6.4 Evidence that the vulnerability has been rectified	85
Step 1:.....	85
Step 2:.....	88
7. Bonus.....	89
7.0.1 Validating Invalid File Types	89
Step 1:.....	89
Step 2:.....	90
Step 3:.....	92
Step 4:.....	92
Step 5:.....	93
Step 6:.....	94
Step 7:.....	95
7.1.1 How to exploit vulnerability – Limiting brute forcing attacks	96
Step 1:.....	96
Step 2:.....	97
Step 3:.....	98
Step 4:.....	98

Step 5:.....	99
Step 6:.....	99
7.1.2 How to rectify vulnerability - Limiting brute forcing attacks.....	100
Step 1:.....	100
Step 2:.....	100
Step 3:.....	101
Step 4:.....	102
Step 5:.....	102
Step 6:.....	103
7.1.3 Evidence that the vulnerability has been rectified - Limiting brute forcing attacks	104
Step 1:.....	104
Step 2:.....	105
7.2.1 How to exploit vulnerability - About URL Link access control (Broken Access Control)	106
Step 1:.....	106
Step 2:.....	106
Step 3:.....	107
7.2.2 How to rectify vulnerability - About URL Link access.....	107
Step 1:.....	107
Step 2:.....	108
Step 3:.....	109
Step 4:.....	110
Step 5:.....	111
Step 6:.....	111
Step 7:.....	112
Step 8:.....	113
Step 9:.....	114

Step 10:.....	115
Step 11:.....	115
Step 12:.....	116
Step 13:.....	117
7.2.3 Evidence that the vulnerability has been rectified - About URL Link access	118
Step 1:.....	118
Step 2:.....	118
Step 3:.....	119
Step 4:.....	119
Step 5:.....	120
Step 6:.....	120
Step 7:.....	121
Step 8:.....	121
7.3.0 Code Review – Preventing Future Vulnerabilities.....	122
Error handling and Nested Callbacks.....	122
Standardizing the JSON Response Data.....	126
Summary of bonus parts including those not shown above and rectifications that are not shown in the above 6 vulnerabilities.....	129
References.....	130

Executive Summary

This executive summary provides an overview of the vulnerabilities identified within the Bee Design website and outlines the recommended strategies for rectifying these vulnerabilities. Through analysis of the website there are critical security weaknesses that could potentially compromise the confidentiality, integrity, and availability of the website.

For this report, we will be addressing 6 vulnerabilities.

1. Cross-Site Scripting (XSS)
2. Injection (SQL Injection)
3. Broken Authentication
4. Broken Access Control
5. Sensitive Data Exposure
6. Insufficient Logging and Monitoring

1. Cross-Site Scripting

1.1 Definition

Cross-Site Scripting (XSS) is a type of web application vulnerability that occurs when an attacker injects malicious scripts into a trusted website, which are then executed by a victim's browser. It takes advantage of the trust a user has in a particular website to perform unauthorized actions or steal sensitive information.

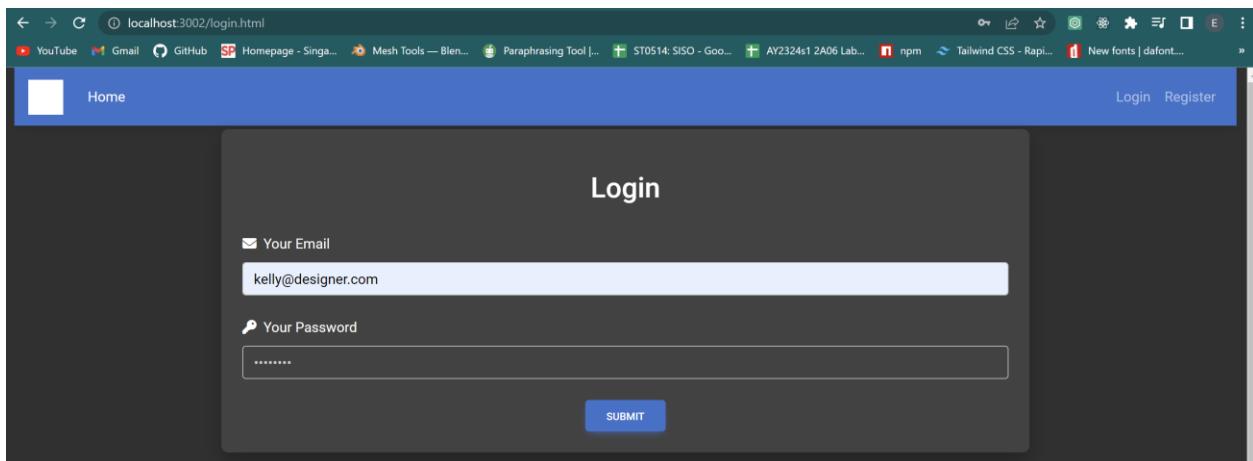
The vulnerability arises when a web application does not properly validate or sanitize user input before displaying it on a web page. This allows an attacker to insert malicious scripts, typically in the form of HTML or JavaScript, into the website's output. When a user visits the compromised page, the malicious script is executed in their browser, often without their knowledge.

There are 3 types of XXS attacks which are Stored, Reflected and DOM-Based.

1.2 How to exploit vulnerability

Step 1:

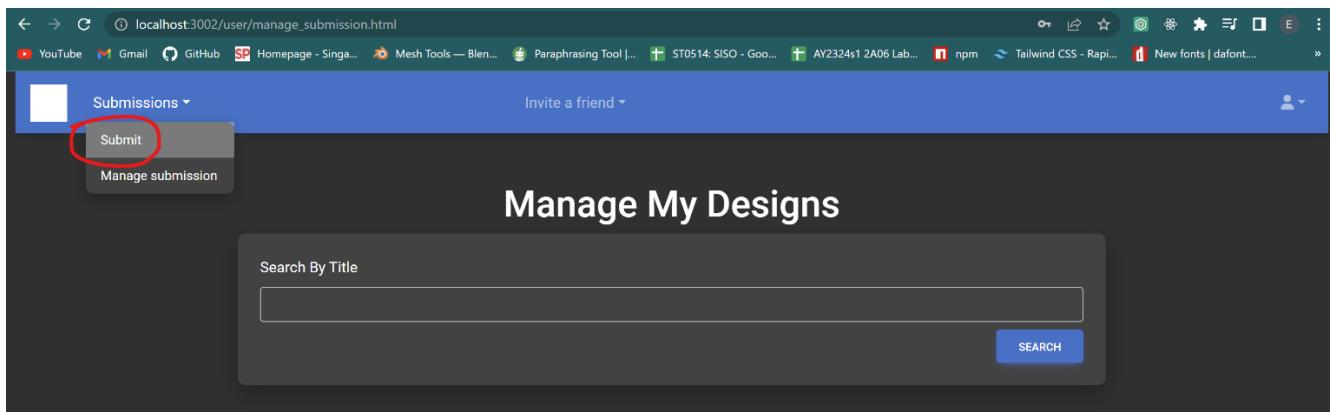
Login using Kelly's Email and Password into the Vulnerable Bee Design Website Login Page.



The screenshot shows a web browser window with the URL `localhost:3002/login.html` in the address bar. The page has a dark-themed header with a blue bar containing the word "Home". Below the header, there is a "Login" form. The form has two input fields: one for "Your Email" containing "kelly@designer.com" and another for "Your Password" containing "*****". At the bottom of the form is a blue "SUBMIT" button. In the top right corner of the browser window, there are several icons for various websites and tools.

Step 2:

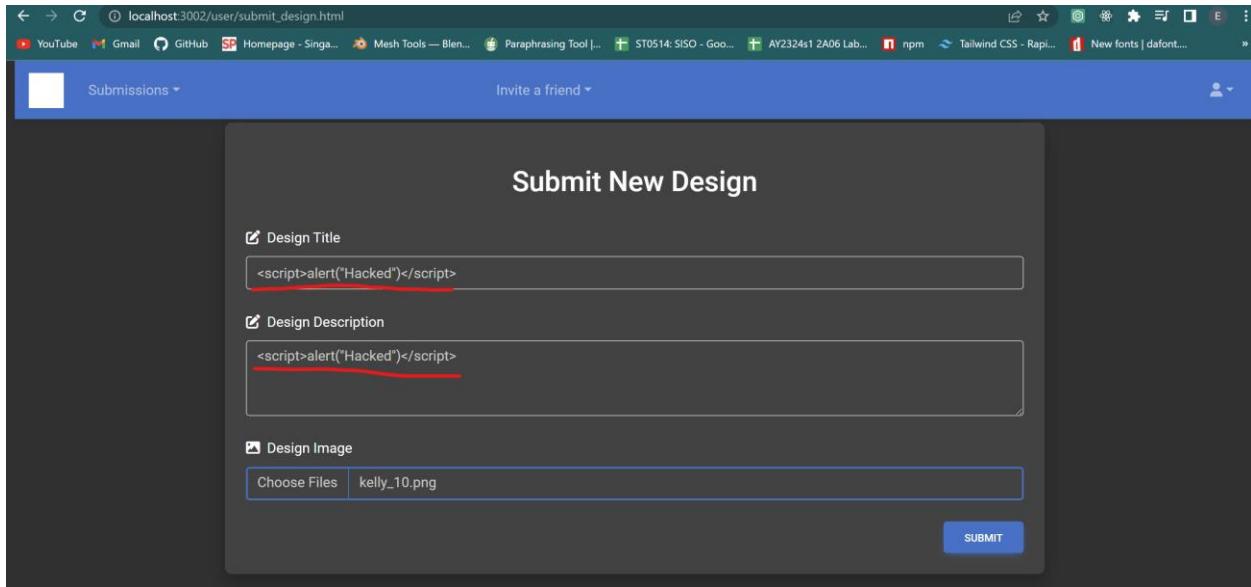
Go into the Submissions Tab at the top left corner of the website to submit a new design submission.



The screenshot shows a web browser window with the URL `localhost:3002/user/manage_submission.html` in the address bar. The page has a dark-themed header with a blue bar containing the word "Submissions" and a dropdown arrow. Below the header, there is a "Manage My Designs" section. This section includes a search bar with the placeholder "Search By Title" and a blue "SEARCH" button. On the left side of the main content area, there is a button labeled "Submit" with a red circle drawn around it. There is also a link labeled "Manage submission". In the top right corner of the browser window, there are several icons for various websites and tools.

Step 3:

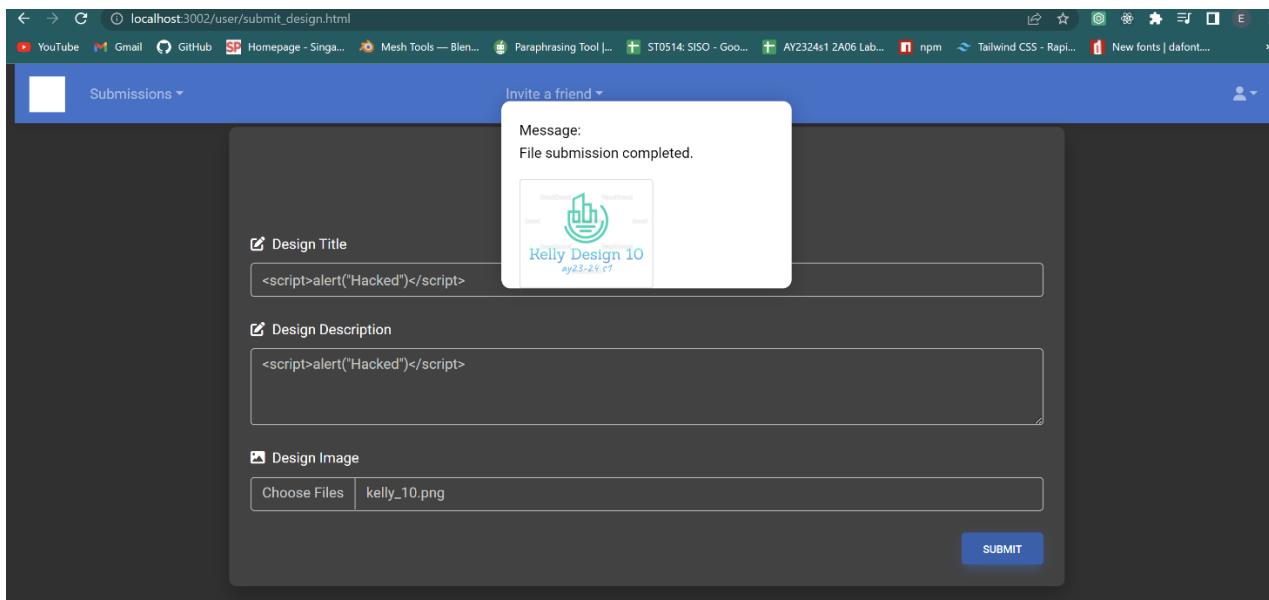
We will fill in the Design Title and Description with <script> tags to perform a Stored XSS Attack.



The screenshot shows a web page titled "Submit New Design". There are three input fields: "Design Title" containing "<script>alert('Hacked')</script>", "Design Description" containing "<script>alert('Hacked')</script>", and "Design Image" showing a file named "kelly_10.png". A red box highlights the contents of both the "Design Title" and "Design Description" fields.

Step 4:

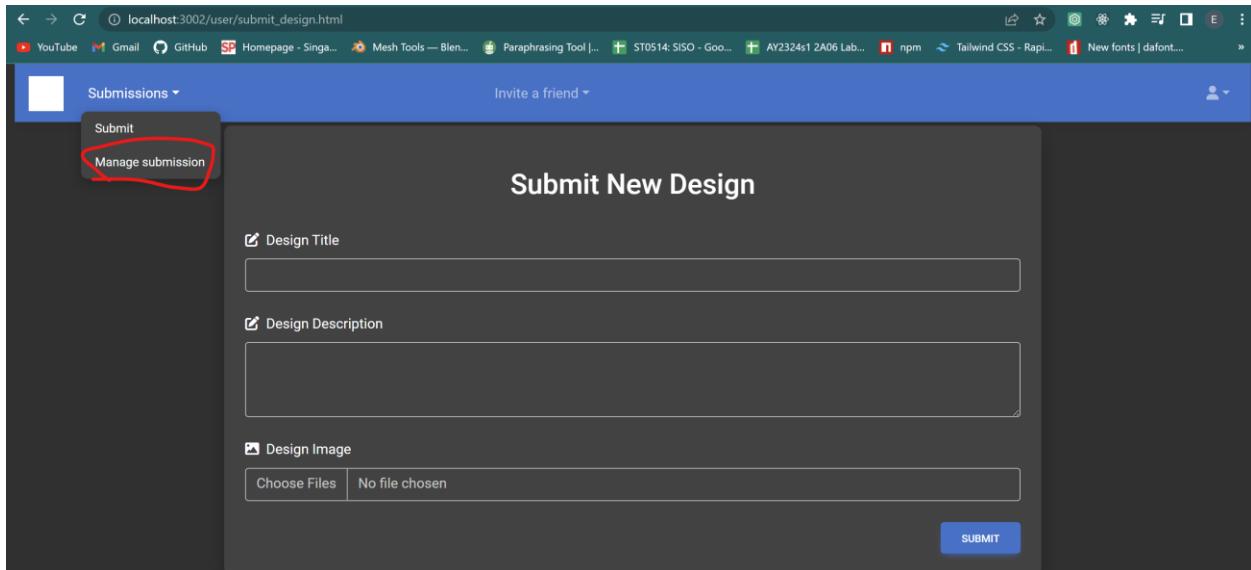
After submitting, the Design that we submitted successfully will be send and stored to the database.



The screenshot shows the same "Submit New Design" form after submission. A message box appears with the text "Message: File submission completed." and shows the uploaded image file "kelly_10.png". The image is a thumbnail of a design featuring a green circular logo and the text "Riley Design 10".

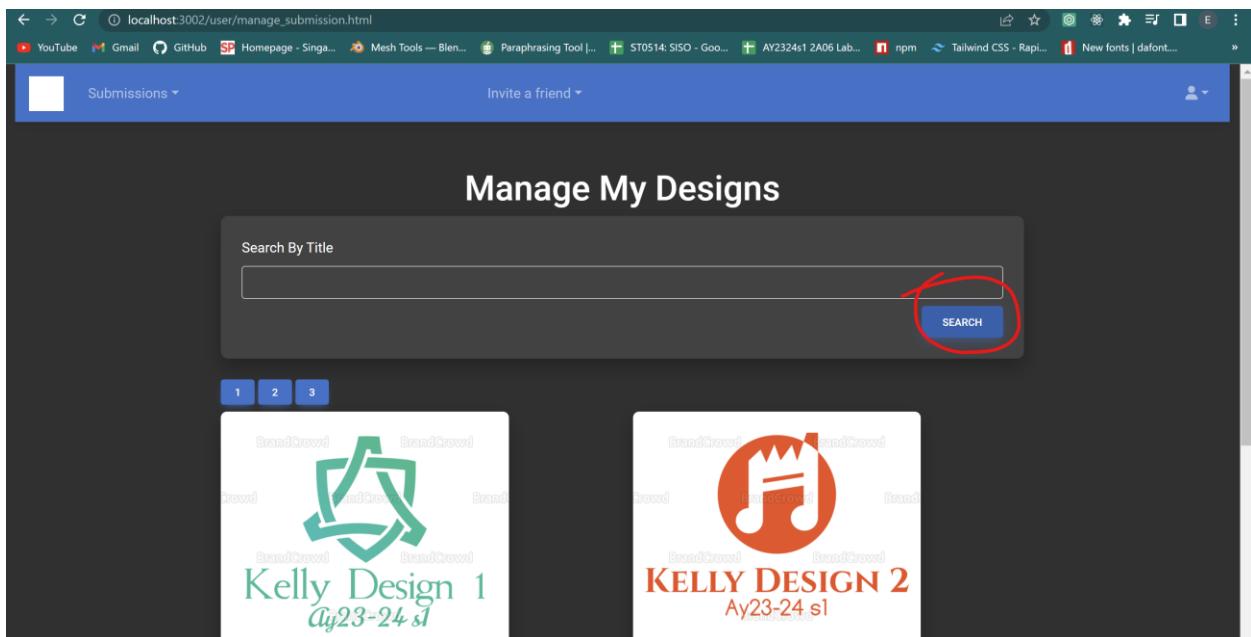
Step 5:

Next, we will go to Manage Submission to view the submission that we just submitted.



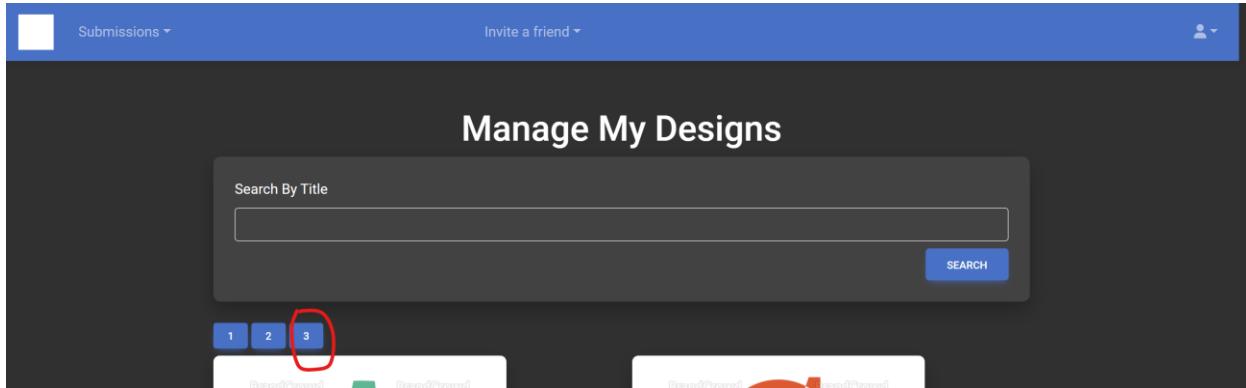
Step 6:

Click on the Search button to view the submissions of Kelly.



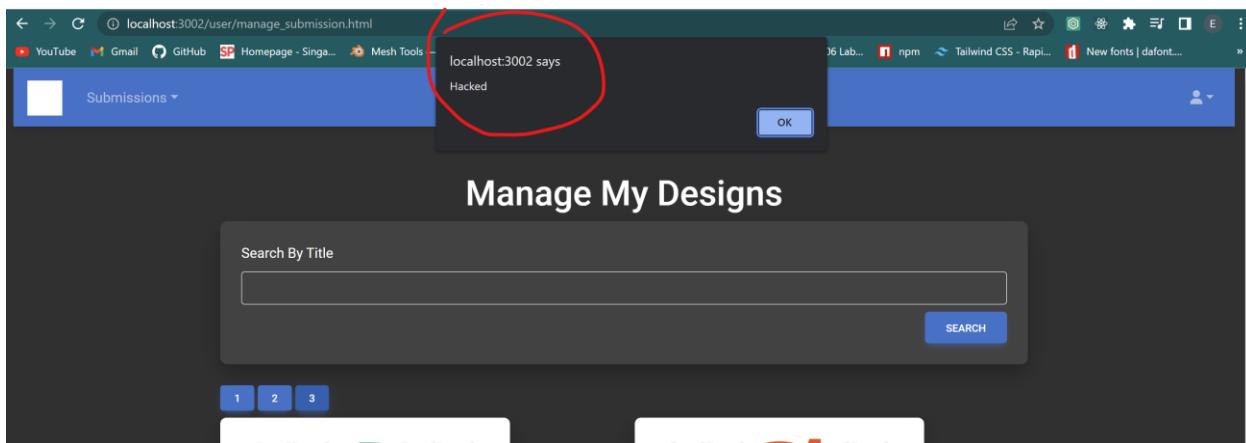
Step 7:

We will now go to view the recent submission that we submitted, click on the 3rd pagination to view.



Step 8:

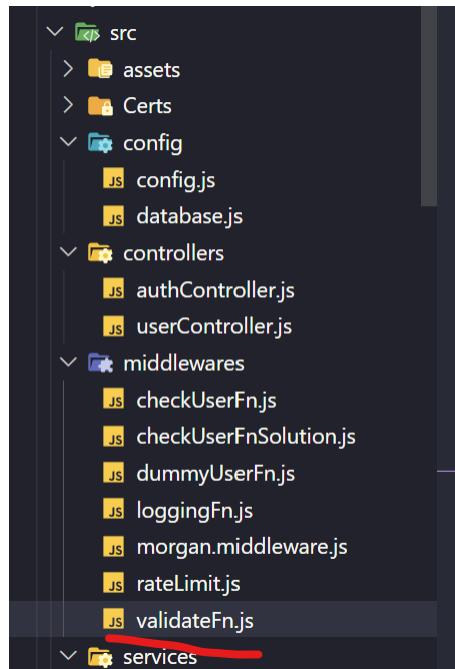
After clicking on pagination 3, you will be alerted with a Hacked message as shown below, which tells us that the Stored XSS Attack was successful.



1.3 How to rectify vulnerability

Step 1:

To prevent Stored XSS Attack, we will need to perform an Input Validation as well as an Output Sanitization. So, we will create a validateFn.js in the /backend/src/middlewares folder to validate the user input.



Step 2:

We will implement the following code in the validateFn.js file to validate the Design Title and Description before allowing the user to submit.

***Note declare the following code in validateFn.js first so that we can export it using module.exports to use this validateFn anywhere in the Backend Directory.**

```
const validateFn = {  
    // The code below will be here  
};  
  
module.exports = validateFn;
```

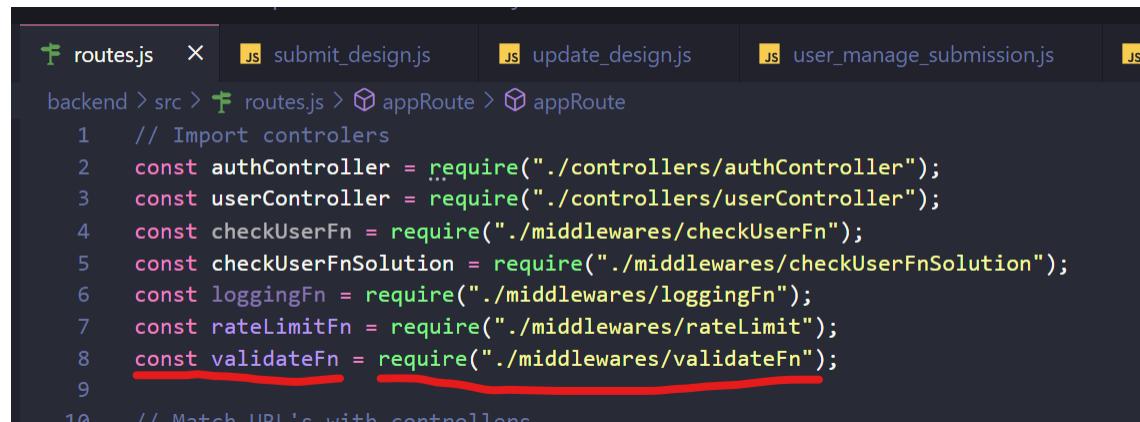
```

73 validateDesignTitleAndDesc: function (req, res, next) {
74     var title = req.body.designTitle;
75     var description = req.body.designDescription;
76
77     // Description contains only alphabet, spaces, digit, single quote
78     const ReTitle = new RegExp(`^[\w\d\s]+`);
79     // Description contains only alphabet, spaces, digit, single/double quote or comma
80     const ReDescription = new RegExp(`^[\w\d\s,"]+`);
81     if (ReTitle.test(title) && ReDescription.test(description)) {
82         next();
83     } else {
84         let message = "Unable to submit design, Title/Description is invalid.";
85         getIPAddress()
86             .then((ip) => {
87                 logger.error({
88                     method: req.method,
89                     route: req.originalUrl,
90                     ip: ip,
91                     error: message,
92                 });
93             })
94             .catch((error) => {
95                 console.log(error);
96             });
97         res.status(400).send({
98             message: message,
99         });
100    }
101 },
102

```

Step 3:

After implementing Input Validation, we will now use this middleware in the routes.js file under /backend/src. First, we will require the validateFn in the routes.js so let us bring it into routes.js.



```

routes.js  X  JS submit_design.js  JS update_design.js  JS user_manage_submission.js  JS
backend > src > routes.js > appRoute > appRoute
1 // Import controllers
2 const authController = require("./controllers/authController");
3 const userController = require("./controllers/userController");
4 const checkUserFn = require("./middlewares/checkUserFn");
5 const checkUserFnSolution = require("./middlewares/checkUserFnSolution");
6 const loggingFn = require("./middlewares/loggingFn");
7 const rateLimitFn = require("./middlewares/rateLimit");
8 const validateFn = require("./middlewares/validateFn");
9
10 // Match URL's with controllers

```

Step 4:

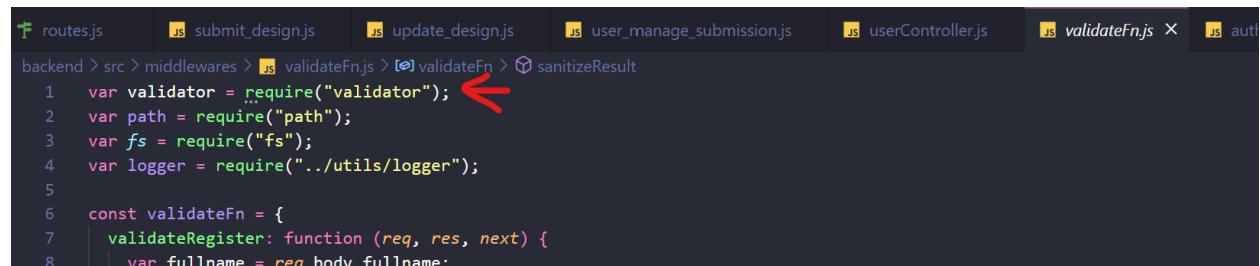
We will now bring the middleware “validateFn.validateDesignTitleAndDesc” into the following endpoint which will help us validate the user input.

```
37 // USER SUBMIT DESIGN
38 router.post(
39   "/api/user/process-submission",
40   loggingFn.logData,
41   checkUserFnSolution.checkForValidUser,
42   validateFn.validateDesignTitleAndDesc, ↖
43   validateFn.validateFileTypeAndSize,
44   rateLimitFn.designSubmissionCountLimit,
45   userController.processDesignSubmission
46 );
47 // USER UPDATE DESIGN (RETRIEVE PHASE)
48 router.get(
49   "/api/user/design/:fileId",
50   checkUserFnSolution.checkForValidUser,
51   userController.processGetOneDesignData
52 );
53 // USER UPDATE DESIGN (UPDATE PHASE)
54 router.put( ↖
55   "/api/user/design/",
56   loggingFn.logData,
57   checkUserFnSolution.checkForValidUser,
58   validateFn.validateDesignTitleAndDesc, ↖
59   userController.processUpdateOneDesign
60 );
```

Step 5:

We will now go to the validateFn.js and import a library called validator first.

*Note Use “npm install validator” in your backend directory in the terminal if you have not installed in your Backend folder directory.



```
routes.js submit_design.js update_design.js user_manage_submission.js UserController.js validateFn.js auth.js
backend > src > middlewares > validateFn.js > validateFn > sanitizeResult
1 var validator = require("validator"); ↖
2 var path = require("path");
3 var fs = require("fs");
4 var logger = require("../utils/logger");
5
6 const validateFn = {
7   validateRegister: function (req, res, next) {
8     var fullname = req.body.fullname;
```

Step 6:

Next, add the following code which we will use to sanitize the results that we retrieve from the database.

```
253     sanitizeResult: function (result) {
254         if (result.filedata) {
255             for (file of result.filedata) {
256                 file.design_title = validator.escape(file.design_title);
257                 file.design_description = validator.escape(file.design_description);
258             }
259         } else if (result.userdata) {
260             if (result.userdata.length >= 1) {
261                 for (user of result.userdata) {
262                     user.fullname = validator.escape(user.fullname);
263                     user.email = validator.escape(user.email);
264                     user.role_name = validator.escape(user.role_name);
265                 }
266             }
267         }
268         return result;
269     }, // end sanitizeResult
270 };
271
```

Step 7:

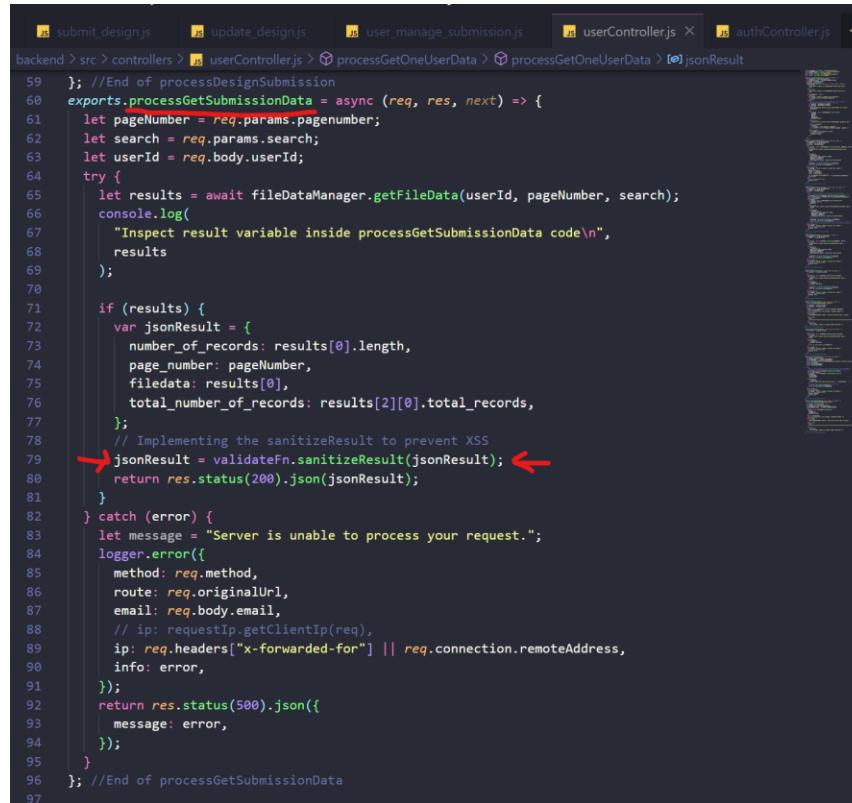
Let us bring in the validationFn.js file into the UserController.js file for output sanitization located under /backend/src/controllers/userController.js .



```
routes.js submit_design.js update_design.js user_manage_submission.js UserController.js X
backend > src > controllers > UserController.js > processUpdateOneDesign > processUpdateOneDesign
1 const userManager = require("../services/userService");
2 const fileDataManager = require("../services/fileService");
3 const config = require("../config/config");
4 const validateFn = require("../middlewares/validationFn");
```

Step 8:

We will now use the Output Sanitization to prevent any existing script tags from working. Go to userController.js and look for the following “exports.processGetSubmissionData” to add the validateFn.sanitizeResult.

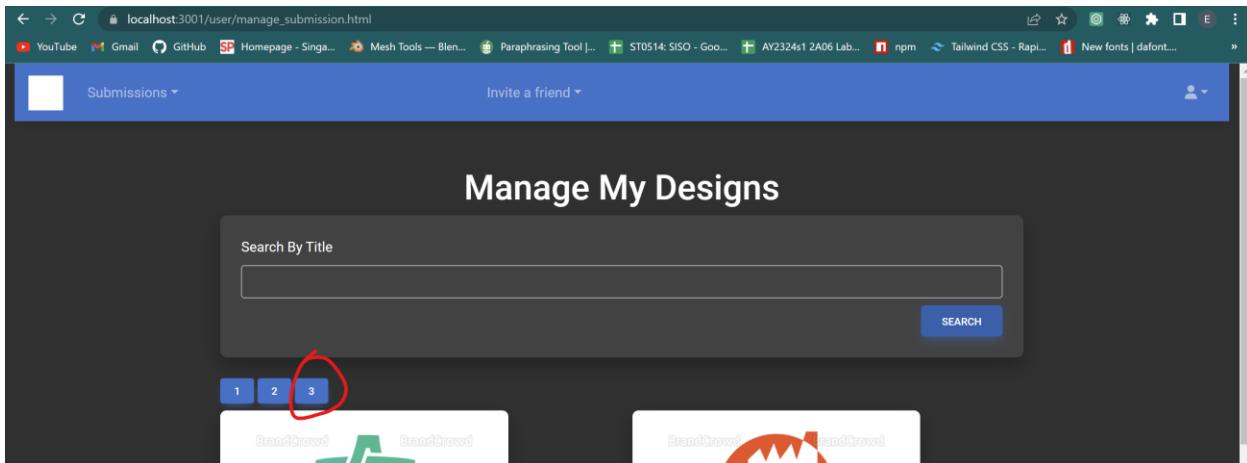


```
submit_design.js update_design.js user_manage_submission.js UserController.js authController.js
backend > src > controllers > UserController.js > processGetOneUserData > processGetOneUserData > JsonResult
59 }; //End of processDesignSubmission
60 exports.processGetSubmissionData = async (req, res, next) => {
61   let pageNumber = req.params.pageNumber;
62   let search = req.params.search;
63   let userId = req.body.userId;
64   try {
65     let results = await fileDataManager.getFileData(userId, pageNumber, search);
66     console.log(
67       "Inspect result variable inside processGetSubmissionData code\n",
68       results
69     );
70
71   if (results) {
72     var jsonResult = {
73       number_of_records: results[0].length,
74       page_number: pageNumber,
75       filedata: results[0],
76       total_number_of_records: results[2][0].total_records,
77     };
78     // Implementing the sanitizeResult to prevent XSS
79     jsonResult = validateFn.sanitizeResult(jsonResult); ←
80     return res.status(200).json(jsonResult);
81   }
82 } catch (error) {
83   let message = "Server is unable to process your request.";
84   logger.error({
85     method: req.method,
86     route: req.originalUrl,
87     email: req.body.email,
88     // ip: requestIp.getClientIp(req),
89     ip: req.headers["x-forwarded-for"] || req.connection.remoteAddress,
90     info: error,
91   });
92   return res.status(500).json({
93     message: error,
94   });
95 }
96 }; //End of processGetSubmissionData
97
```

1.4 Evidence that the vulnerability has been rectified

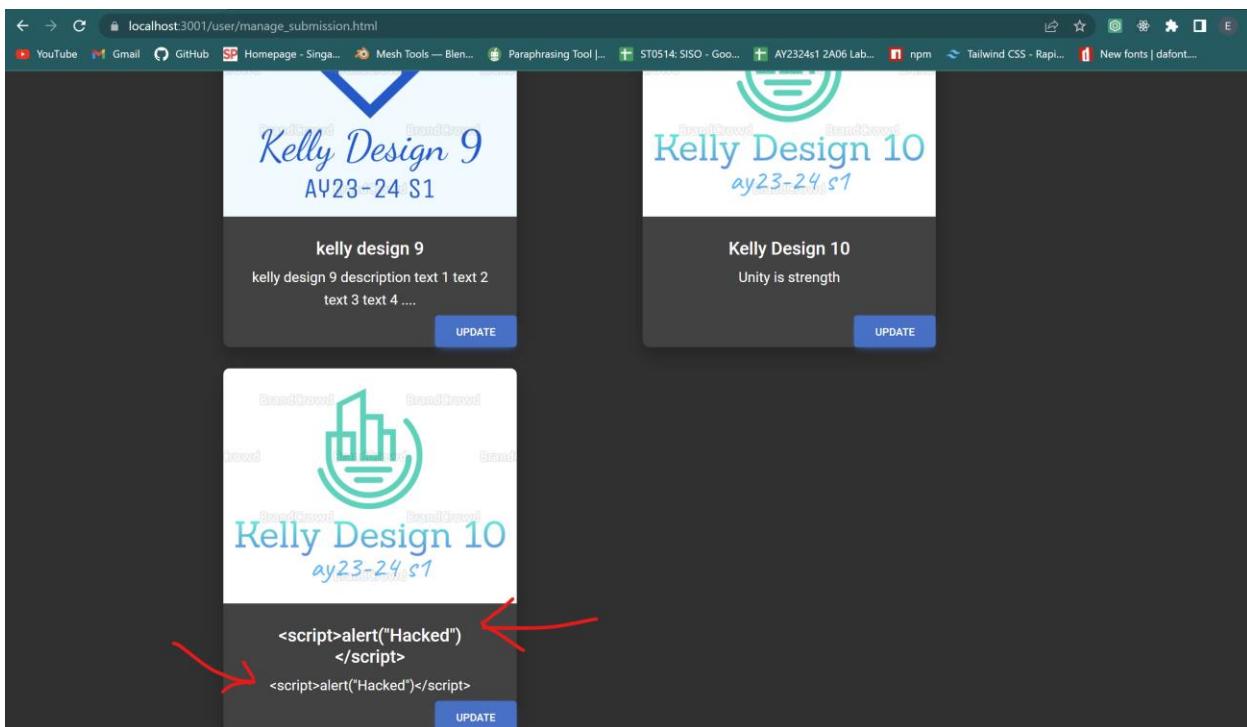
Step 1:

Go back to the Manage My Designs page to search for Kelly's submissions and click on the third pagination to see if the alert still shows.



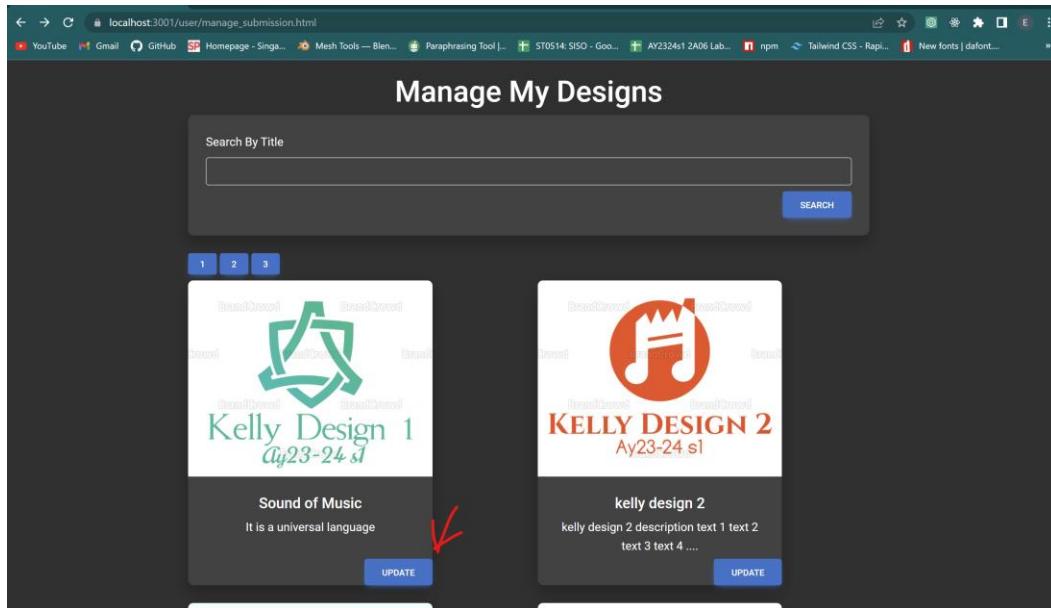
Step 2:

You should be able to see the escaped script tags in the title and description and no alert has popped up. This shows that we have successfully performed the output sanitization.



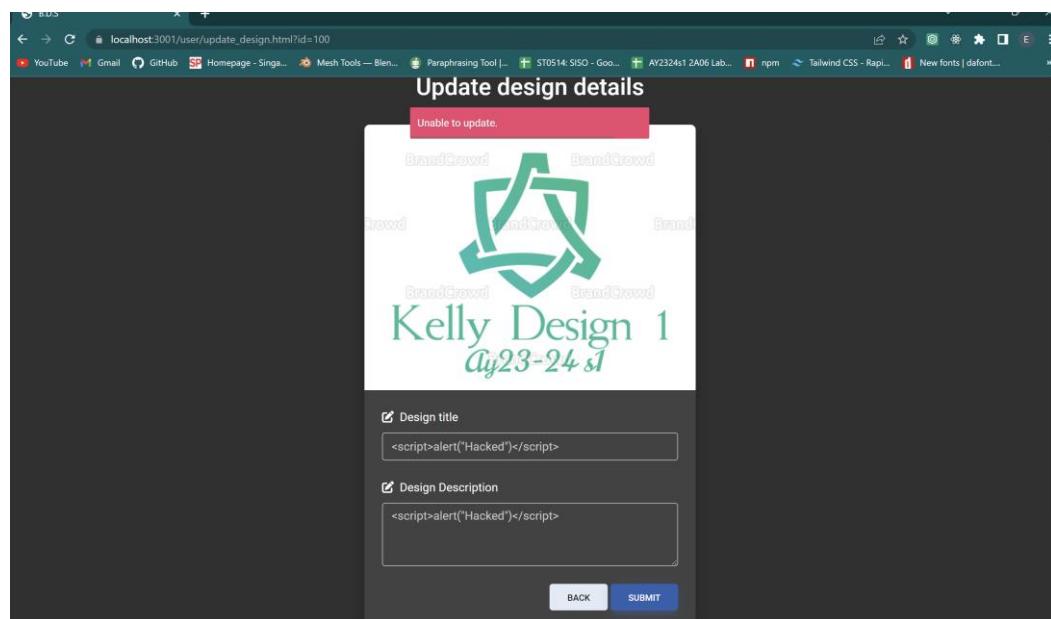
Step 3:

Lastly, let's try to update the design with script tags to see if our validation worked. Go to pagination 1 and click on the Update button.



Step 4:

Fill in the title and description with script tags to see if it will go through. Upon submitting you should see an Unable to update message which shows that our input validation is working.



2. Injection (SQL Injection)

2.1 Definition

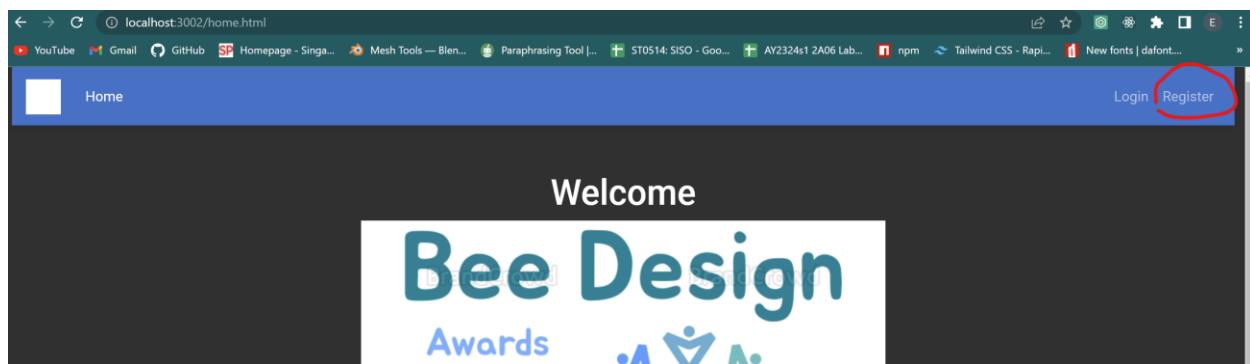
SQL Injection is a type of web application vulnerability that allows an attacker to manipulate a web application's database by injecting malicious SQL (Structured Query Language) code into user input fields or other vulnerable areas of the application. It occurs when the web application fails to properly validate or sanitize user input before incorporating it into SQL queries, allowing the attacker to execute arbitrary SQL commands.

The vulnerability arises when user-supplied data is concatenated directly into SQL statements without proper sanitization or parameterization. The attacker can exploit this vulnerability by submitting crafted input that includes SQL code as part of the user input. When the application processes this input, the malicious SQL code is executed by the database server, potentially leading to unauthorized access, data manipulation, or disclosure of sensitive information.

2.2 How to exploit vulnerability

Step 1:

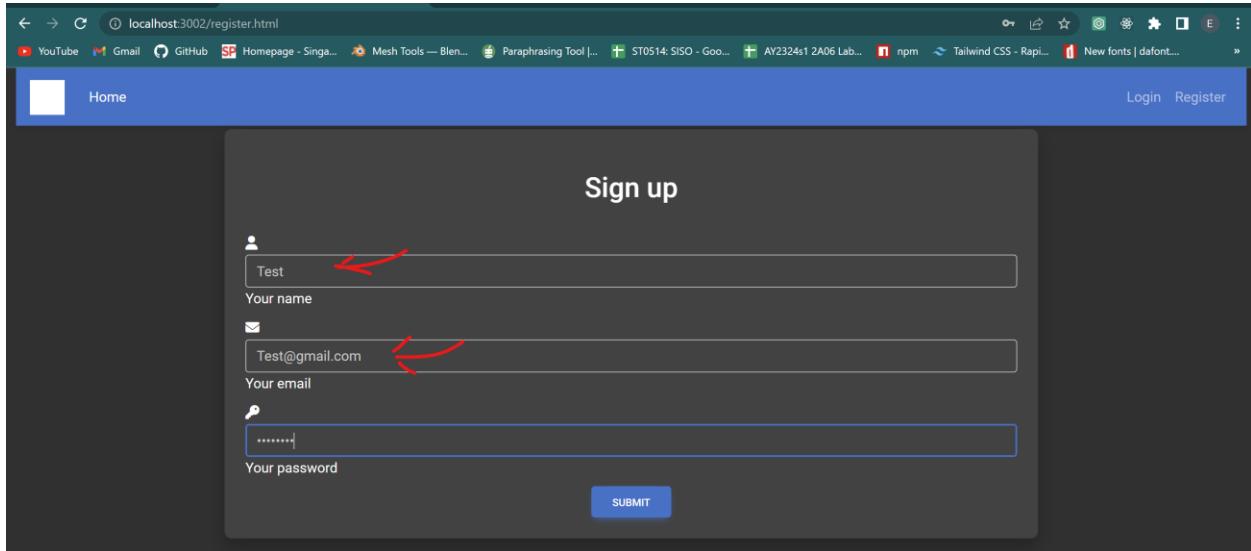
Go into the Register Page to Register as a new user by clicking on Register at the Navbar in the Welcome Page.



Step 2:

Fill in the necessary fields to register as a new user.

For this case - Name: Test, Email: Test@gmail.com, Password: password



A screenshot of a web browser window showing a 'Sign up' form. The URL in the address bar is 'localhost:3002/register.html'. The page has a dark background with a blue header bar. The header bar includes a 'Home' button, a 'Login' link, and a 'Register' link. The main content area is titled 'Sign up'. It contains three input fields: 'Your name' with the value 'Test', 'Your email' with the value 'Test@gmail.com', and 'Your password' with the value '.....'. A red arrow points to the 'Your name' field, and another red arrow points to the 'Your email' field. Below the fields is a blue 'SUBMIT' button.

Step 3:

You should see that you have successfully registered.

Login'. This message is circled with a red line." data-bbox="114 548 881 805"/>

A screenshot of a web browser window showing the same 'Sign up' form as the previous screenshot. The URL is again 'localhost:3002/register.html'. The page title is 'Sign up'. At the top, there is a green message box with the text 'You have registered. Please [Login](#)'. This message box is circled with a red line. Below the message box is the 'Sign up' form with its fields: 'Your name' (Test), 'Your email' (Test@gmail.com), and 'Your password' (.....). A blue 'SUBMIT' button is at the bottom. The rest of the page is identical to the first screenshot, including the header bar with 'Home', 'Login', and 'Register' links.

Step 4:

Go into MySQL Workbench and look for the user we just registered under the “user” table.

The screenshot shows the MySQL Workbench interface. In the left sidebar, under the 'Schemas' section, the 'ay2324s1_st0505_esde_db' schema is selected, and the 'user' table is chosen. The main pane displays the 'Result Grid' of the 'user' table. The grid has columns: user_id, fullname, email, user_password, and role_id. There are 12 rows of data, including a new entry for 'Test'. A red arrow points to the last row, which corresponds to the user 'Test'.

user_id	fullname	email	user_password	role_id
108	jacob	jacob@designer.com	\$2b\$10\$K.0Hwps0PDGaB/atFBmmXOGTw4ceeg...	2
109	eileen	eileen@designer.com	\$2b\$10\$K.0Hwps0PDGaB/atFBmmXOGTw4ceeg...	2
110	becky	becky@designer.com	\$2b\$10\$K.0Hwps0PDGaB/atFBmmXOGTw4ceeg...	2
111	dylan	dylan@designer.com	\$2b\$10\$K.0Hwps0PDGaB/atFBmmXOGTw4ceeg...	2
112	helen	helen@designer.com	\$2b\$10\$K.0Hwps0PDGaB/atFBmmXOGTw4ceeg...	2
113	elsie	elsie@designer.com	\$2b\$10\$K.0Hwps0PDGaB/atFBmmXOGTw4ceeg...	2
114	manfred	manfred@designer.com	\$2b\$10\$K.0Hwps0PDGaB/atFBmmXOGTw4ceeg...	2
115	hushien	hushien@designer.com	\$2b\$10\$K.0Hwps0PDGaB/atFBmmXOGTw4ceeg...	2
119	ss	ss@gmail.com	\$2b\$10\$I2TlppvMDtMyD/Xm1eR0o.qTT7hgRHE...	2
120	Test	Test@gmail.com	\$2b\$10\$QIN9UO/FLAk39bbz0qoIPe17sLGrqj9sE...	2
*	NULL	NULL	NULL	NULL

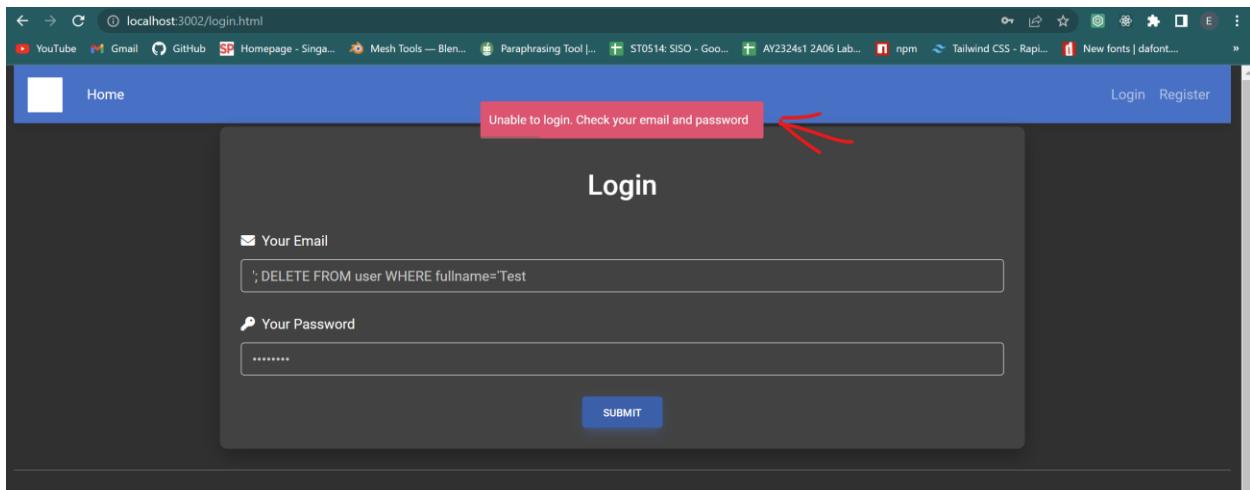
Step 5:

Now, go to the Vulnerable Bee Design Website Login Page we will add a SQL statement to try to delete the recent user that we just created.

The screenshot shows a web browser window with the URL 'localhost:3002/login.html'. The page is titled 'Login'. It has two input fields: 'Your Email' and 'Your Password'. The 'Your Email' field contains the value '; DELETE FROM user WHERE fullname='Test'. A red arrow points to this field. Below the fields is a 'SUBMIT' button.

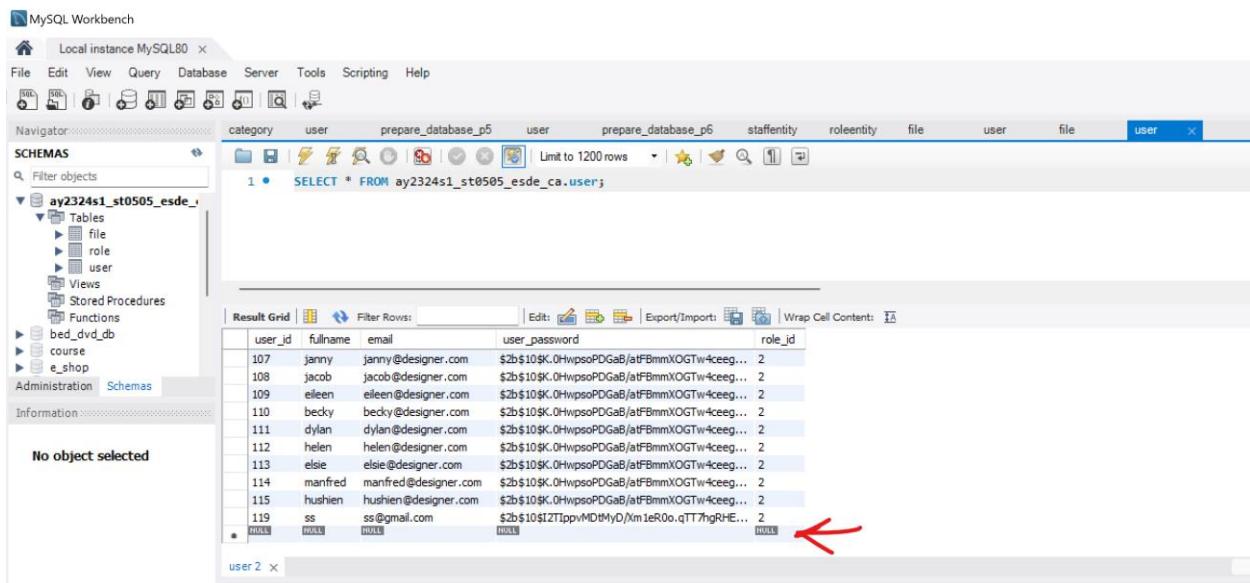
Step 6:

You should be able to see a message saying “Unable to Login. Check your email and password”.



Step 7:

Now, head over to MySQL Workbench to see if the user “Test” is still there. You will see that the user “Test” has been deleted. This shows that the Login page is susceptible to SQL Injection.

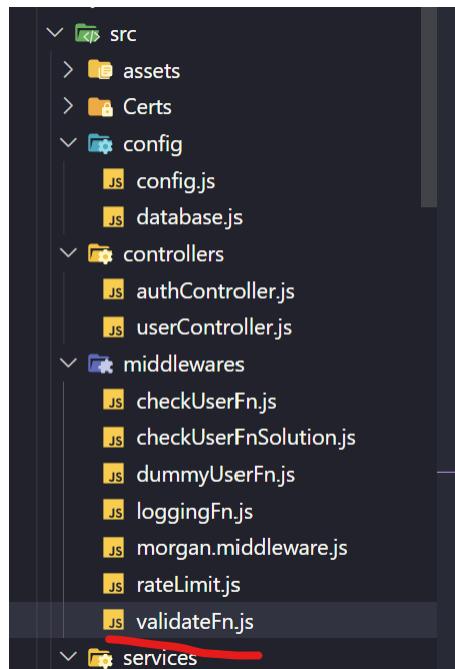


2.3 How to rectify vulnerability

Step 1:

To rectify this problem, we can do two different things, the first thing is input validation followed by using parameterized queries.

For Input Validation, head over to your VsCode and look for the middleware that we created from the 1st Vulnerability called validateFn.js.



Step 2:

Make sure that you have validator library being installed and imported into your validateFn.js file.

*Note Use “npm install validator” in your backend directory in the terminal if you have not installed in your Backend folder directory.

```
routes.js submit_design.js update_design.js user_manage_submission.js UserController.js validateFn.js auth
backend > src > middlewares > validateFn.js > validateFn > sanitizeResult
1 var validator = require("validator"); ←
2 var path = require("path");
3 var fs = require("fs");
4 var logger = require("../utils/logger");
5
6 const validateFn = {
7   validateRegister: function (req, res, next) {
8     var fullname = req.body.fullname;
```

Step 3:

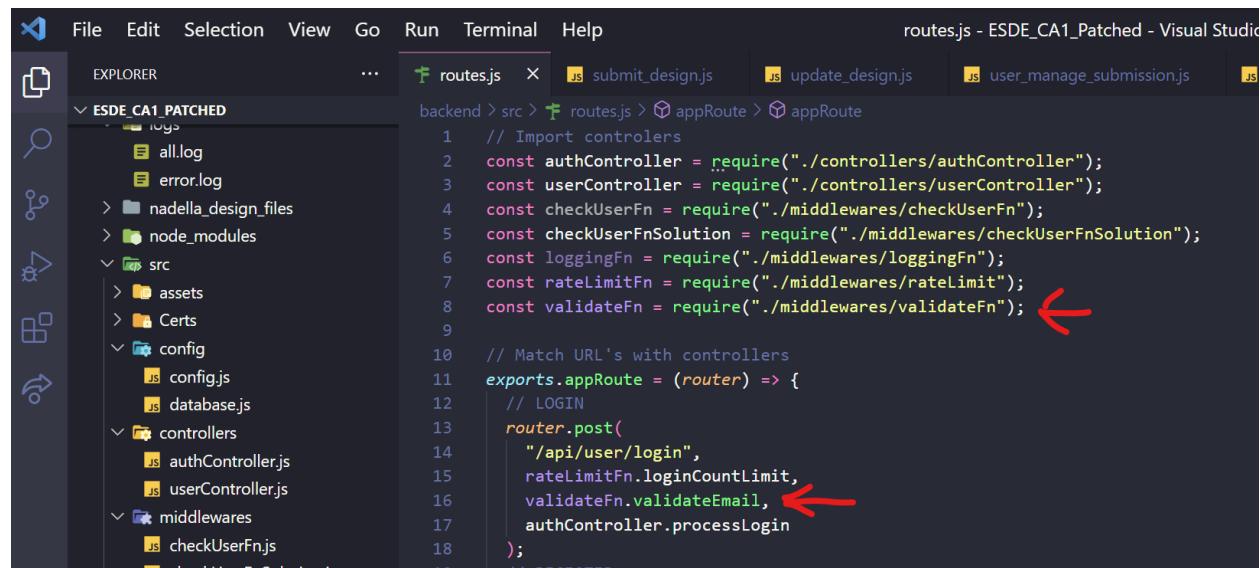
Add the following code into your validateFn.js file.

*Note: Ignore line 56-68 if you have not done the 6th Vulnerability Insufficient Logging and Monitoring.

```
50 validateEmail: function (req, res, next) {
51   var email = req.body.email;
52   if (validator.isEmail(email)) {
53     next();
54   } else {
55     let message = "Unable to Login, Please enter a valid Email";
56     getIPAddress()
57       .then((ip) => {
58         logger.error({
59           method: req.method,
60           route: req.originalUrl,
61           email: req.body.email,
62           ip: ip,
63           error: message,
64         });
65       })
66       .catch((error) => {
67         console.log(error);
68       });
69     res.status(400).send({ message: message });
70   }
71 },
```

Step 4:

Next, head over to routes.js to add this middleware to the Login Endpoint. Do remember to import the validateFn.js file into the routes.js file.



```
routes.js - ESDE_CA1_Patched - Visual Studio Code
File Edit Selection View Go Run Terminal Help
routes.js - ESDE_CA1_Patched - Visual Studio Code
backend > src > routes.js > appRoute > appRoute
1 // Import controllers
2 const authController = require("./controllers/authController");
3 const userController = require("./controllers/userController");
4 const checkUserFn = require("./middlewares/checkUserFn");
5 const checkUserFnSolution = require("./middlewares/checkUserFnSolution");
6 const loggingFn = require("./middlewares/loggingFn");
7 const rateLimitFn = require("./middlewares/rateLimit");
8 const validateFn = require("./middlewares/validateFn"); ←
9
10 // Match URL's with controllers
11 exports.appRoute = (router) => {
12   // LOGIN
13   router.post(
14     "/api/user/login",
15     rateLimitFn.loginCountLimit,
16     validateFn.validateEmail, ←
17     authController.processLogin
18   );
19   // REGISTER
```

Step 5:

Now, let us go to authService.js to look at the SQL Statement used to check the Login credentials. Located at /backend/src/services/authService.js.

```
File Edit Selection View Go Run Terminal Help
EXPLORER ... authController.js authService.js
backend > src > services > authService.js
1 config = require('../config/config');
2 const pool = require('../config/database')
3 module.exports.authenticate = (email, callback) => {
4   pool.getConnection((err, connection) => {
5     if (err) {
6       if (err) throw err;
7     } else {
8       try {
9         connection.query('SELECT user.user_id, fullname, email, user_password, role_name, user.role_id
10           FROM user INNER JOIN role ON user.role_id=role.role_id AND email=?', [email], (err, rows) => {
11           if (err) {
12             if (err) return callback(err, null);
13           } else {
14             if (rows.length == 1) {
15               console.log(rows);
16               return callback(null, rows);
17             } else {
18               return callback('Login has failed', null);
19             }
20           }
21         connection.release();
22       });
23     }
24   });
25 });
26 });
27 );
28 });
29 );
} catch (error) {
}
```

Step 6:

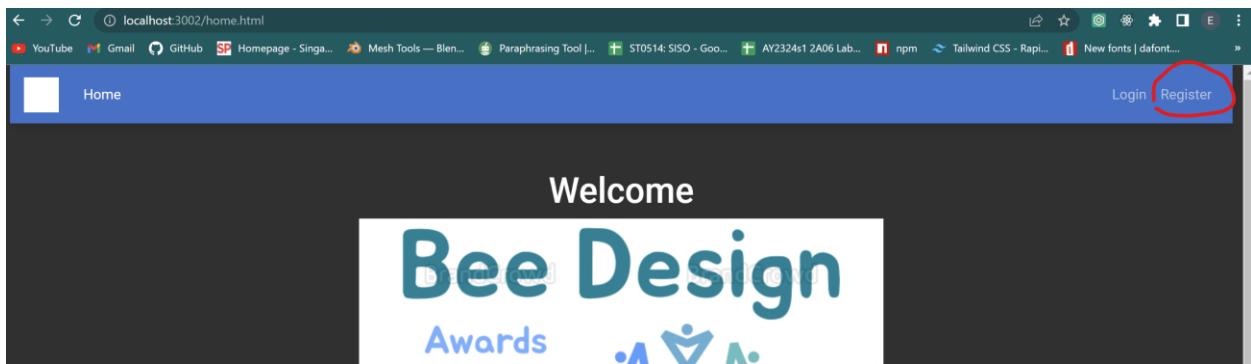
As you can see that the SQL Statement is not using Parameterized Query. So, let us change the following into the code shown below to use Parameterized Query.

```
File Edit Selection View Go Run Terminal Help
EXPLORER ... routes.js authService.js submit_design.js update_design.js user_manage_submission.js
backend > src > services > authService.js > authenticate > authenticate > pool.getConnection() callback
1 config = require("../config/config");
2 const pool = require("../config/database");
3 module.exports.authenticate = (email, callback) => {
4   pool.getConnection((err, connection) => {
5     if (err) {
6       if (err) throw err;
7     } else {
8       try {
9         connection.query(
10           'SELECT user.user_id, fullname, email, user_password, role_name, user.role_id
11             FROM user INNER JOIN role ON user.role_id=role.role_id AND email=?',
12           [email],
13           (err, rows) => {
14             if (err) {
15               if (err) return callback(err, null);
16             } else {
17               if (rows.length == 1) {
18                 console.log(rows);
19                 return callback(null, rows);
20               } else {
21                 return callback("Login has failed", null);
22               }
23             }
24           connection.release();
25         );
26       });
27     }
28   });
29 });
30 );
} catch (error) {
  return callback(error, null);
}
```

2.4 Evidence that the vulnerability has been rectified

Step 1:

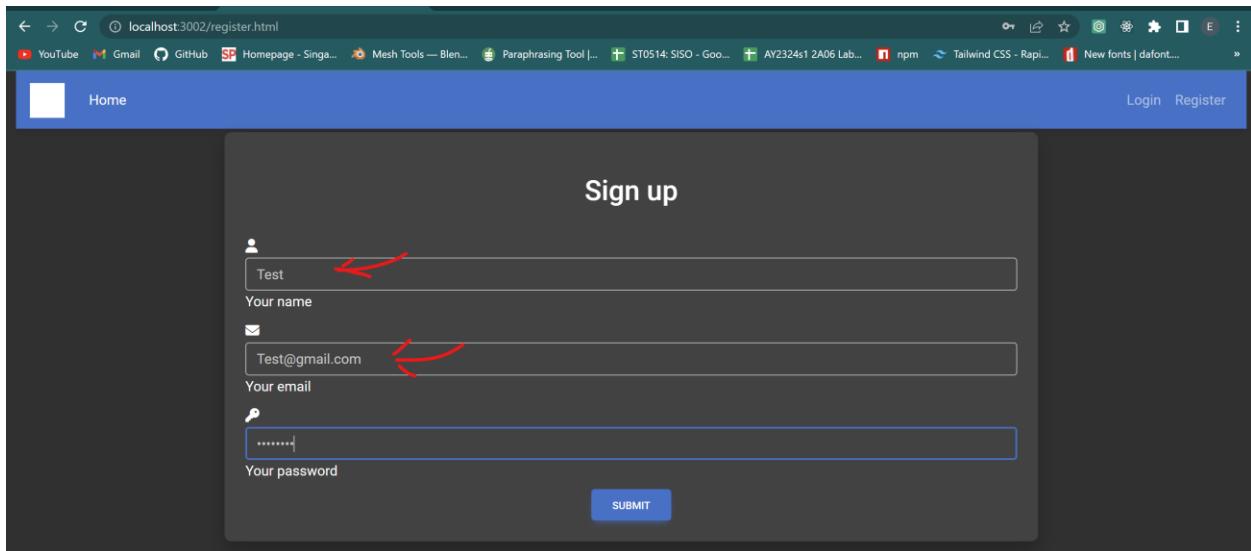
Go into the Register Page to Register as a new user by clicking on Register at the Navbar in the Welcome Page.



Step 2:

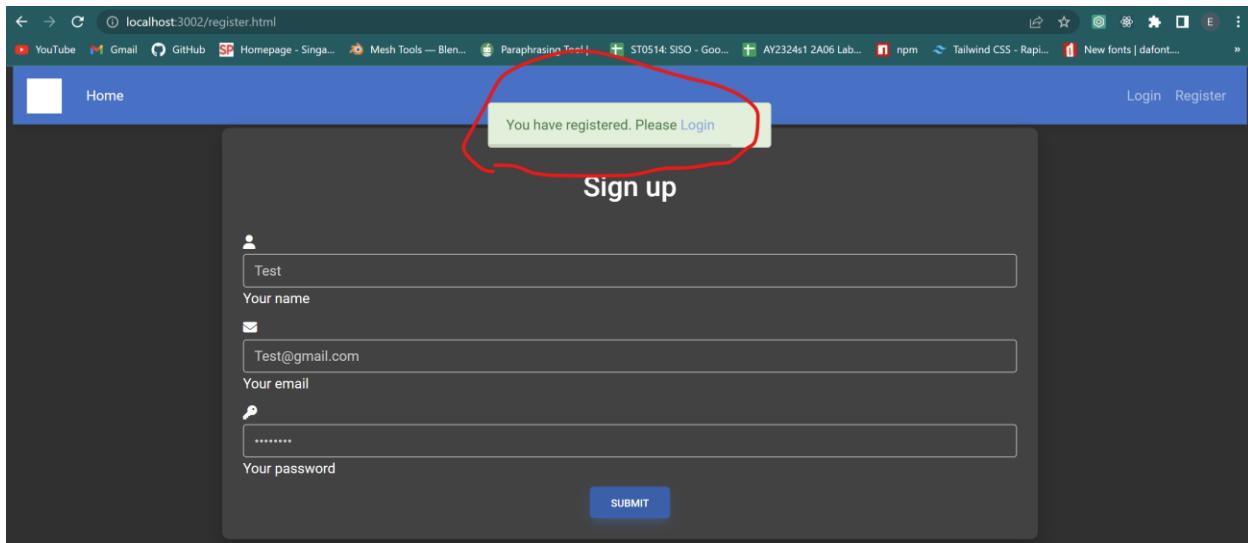
Fill in the necessary fields to register as a new user.

For this case - Name: Test, Email: Test@gmail.com, Password: password



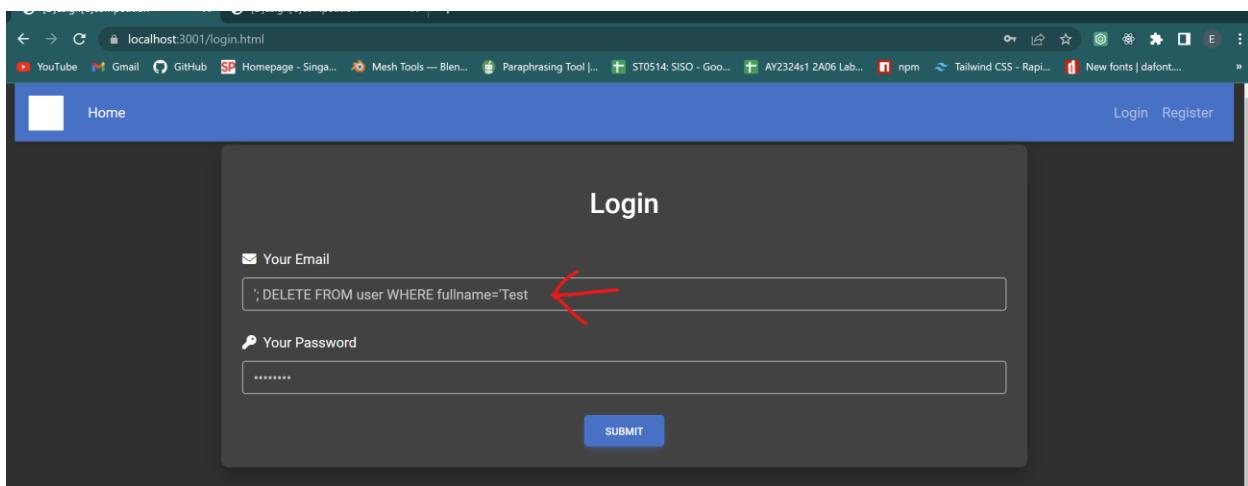
Step 3:

You should see that you have successfully registered.



Step 4:

We will now see if the Input Validation and Parameterized Query has removed the SQL Injection Vulnerability.



Step 5:

You should still see the same pop up showing invalid email and password.

A screenshot of a web browser window showing a login form. The URL bar indicates the page is at `localhost:3001/login.html`. The browser's address bar shows various tabs and icons. The main content is a dark-themed login page with a red error message box at the top center containing the text "Unable to login. Check your email and password". Below the message is a "Login" title and two input fields: one for "Your Email" containing the value "; DELETE FROM user WHERE fullname=Test" and another for "Your Password" containing the value ".....". At the bottom is a blue "SUBMIT" button.

Step 6:

Go over to see if the “Test” user is being deleted, you should be able to still see the “Test” user in your MySQL Workbench under the user table. This shows that we have successfully rectified the SQL Injection Vulnerability for the Login Page.

A screenshot of the MySQL Workbench application. The interface includes a toolbar, a menu bar, and a central workspace. In the workspace, the "Schemas" tree on the left shows a database named "ay2324s1_st0505_esde_ca". Under this database, the "Tables" node is expanded, showing "file", "role", "user", "Views", "Stored Procedures", and "Functions". The "user" table is selected. A query editor at the top displays the SQL command: `SELECT * FROM ay2324s1_st0505_esde_ca.user;`. Below the query editor is a "Result Grid" showing the data from the "user" table. The grid has columns: user_id, fullname, email, user_password, and role_id. There are 12 rows of data, including the "Test" user entry at the bottom. The "role_id" column for the "Test" user entry contains the value "NULL", which is highlighted with a red arrow. The "email" column for the "Test" user entry contains the value "Test@gmail.com".

user_id	fullname	email	user_password	role_id
108	jacob	jacob@designer.com	\$2b\$10\$K.0Hwps0PDGkB/atfBmmXOGTw4ceeg...	2
109	ileen	ileen@designer.com	\$2b\$10\$K.0Hwps0PDGkB/atfBmmXOGTw4ceeg...	2
110	becky	becky@designer.com	\$2b\$10\$K.0Hwps0PDGkB/atfBmmXOGTw4ceeg...	2
111	dylan	dylan@designer.com	\$2b\$10\$K.0Hwps0PDGkB/atfBmmXOGTw4ceeg...	2
112	helen	helen@designer.com	\$2b\$10\$K.0Hwps0PDGkB/atfBmmXOGTw4ceeg...	2
113	elsie	elsie@designer.com	\$2b\$10\$K.0Hwps0PDGkB/atfBmmXOGTw4ceeg...	2
114	manfred	manfred@designer.com	\$2b\$10\$K.0Hwps0PDGkB/atfBmmXOGTw4ceeg...	2
115	hushien	hushien@designer.com	\$2b\$10\$K.0Hwps0PDGkB/atfBmmXOGTw4ceeg...	2
119	ss	ss@gmail.com	\$2b\$10\$12TppvMDtMyD/Xm1eR0o.qTT7hgRHE...	2
121	Test	Test@gmail.com	\$2b\$10\$InSVx2gJdYdhws176NOpfOmcXGFqI...	2
*	NULL	NULL	NULL	NULL

3. Broken Authentication

3.1 Definition

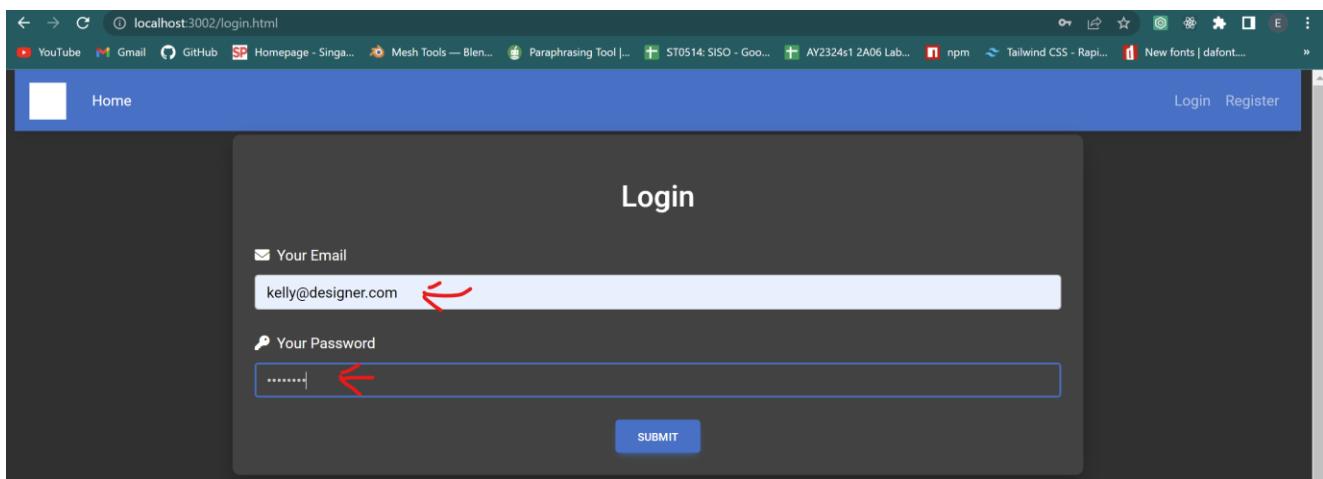
Broken Authentication is a security vulnerability that occurs when the authentication mechanisms of a web application are improperly implemented or configured, allowing attackers to bypass or compromise user authentication and gain unauthorized access to privileged resources or accounts.

When authentication mechanisms are broken or flawed, it can lead to various types of attacks like Credential Stuffing, Brute Force Attacks, Session Hijacking, Account Enumeration, and Password Reset Attacks.

3.2 How to exploit vulnerability

Step 1:

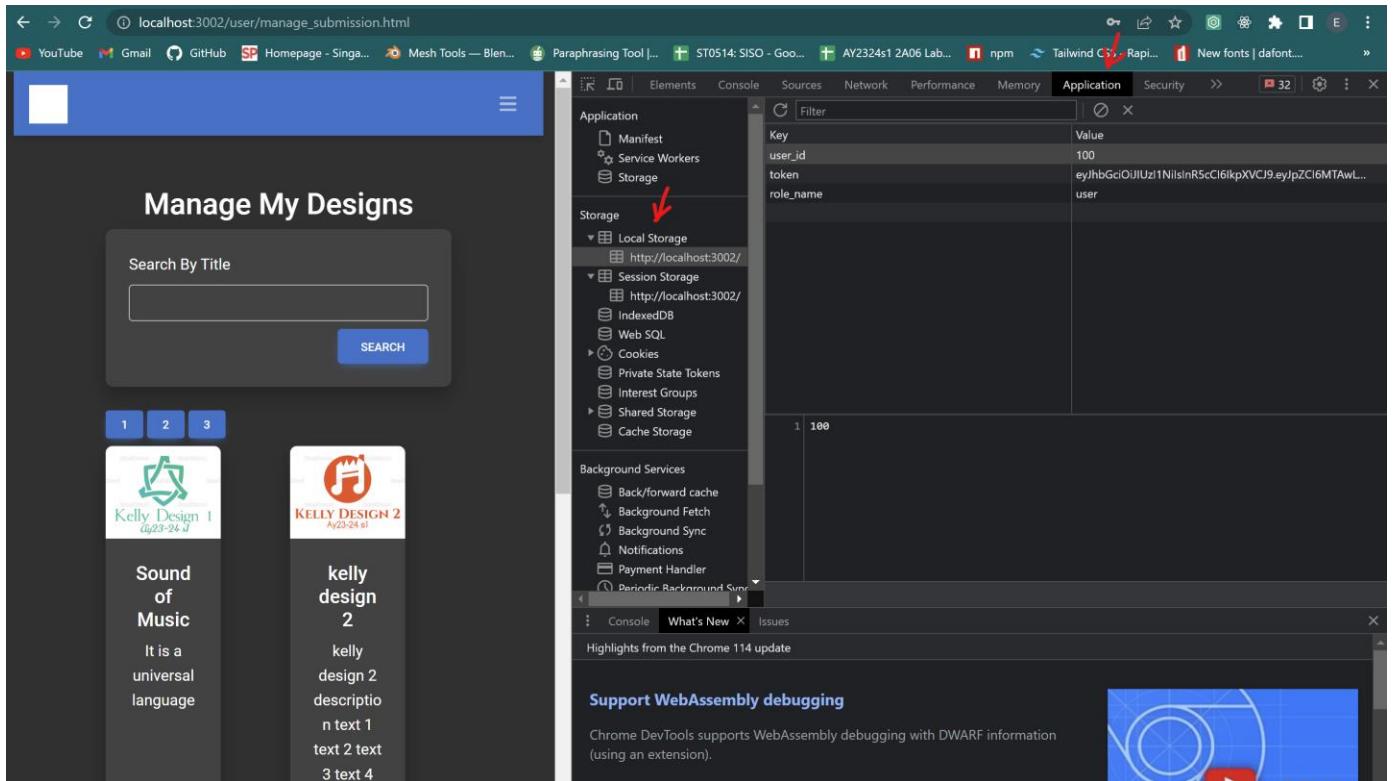
Login to the vulnerable website using Kelly's email and password.



A screenshot of a web browser displaying a login form. The URL in the address bar is 'localhost:3002/login.html'. The page has a dark background with a central 'Login' button. There are two input fields: one for 'Your Email' containing 'kelly@designer.com' and one for 'Your Password' containing '.....'. Red arrows point to both of these input fields, highlighting them as potential targets for exploitation.

Step 2:

Right Click and open the inspect element as shown below and navigate to Application > Local Storage.



Step 3:

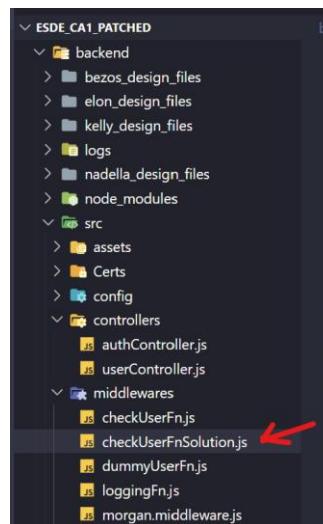
Let's modify the user_id from 100 to 101 to see if we can view Bezos's submissions. After modifying, click on search and you should be able to see Bezos's submissions.

The screenshot shows a web browser window with the URL `localhost:3002/user/manage_submission.html`. The main content area displays a list of designs under the heading "Manage My Designs". There are two items visible: "Bezos Design 1" and "Bezos Design 2". Below each design is its title and author. A red arrow points from the "SEARCH" button on the left side of the page to the "Storage" section in the DevTools Application tab. In the DevTools, the "Local Storage" section is expanded, showing a key-value pair for "user_id": "101". Another red arrow points from the "101" value in the DevTools to the "user_id" key in the list.

3.3 How to rectify vulnerability

Step 1:

Let's go over to the backend folder and look for `checkUserFnSolution.js` under `/backend/src/middlewares/checkUserFnSolution.js`.



Step 2:

Add the following code so that we will use the token to get the user id rather than retrieving from the storage where we can manipulate the value.

```
5  module.exports.checkForValidUser = (req, res, next) => {
6    const authHeader = req.headers["authorization"];
7    // check if authorization header is null, undefined or does not contain Bearer
8    if (
9      authHeader === null ||
10     authHeader === undefined ||
11     !authHeader.startsWith("Bearer "))
12   ) {
13     console.log("req.headers.authorization is undefined");
14     return res
15       .status(403)
16       .send({ message: "Unauthorised! You are not logged in as user." });
17   } else {
18     // Retrieve the authorization header and parse out the
19     // JWT using the split function
20     console.log("retrieving authorization header");
21     let token = authHeader.split(" ")[1];
22
23     jwt.verify(token, config.JWTKey, (err, data) => {
24       console.log("data extracted from token \n", data);
25       if (err) {
26         console.log(err);
27         return res.status(403).send({ message: "Unauthorised Access" });
28       } else {
29         req.body.userId = data.id;
30         next();
31       }
32     });
33   }
34 };

```

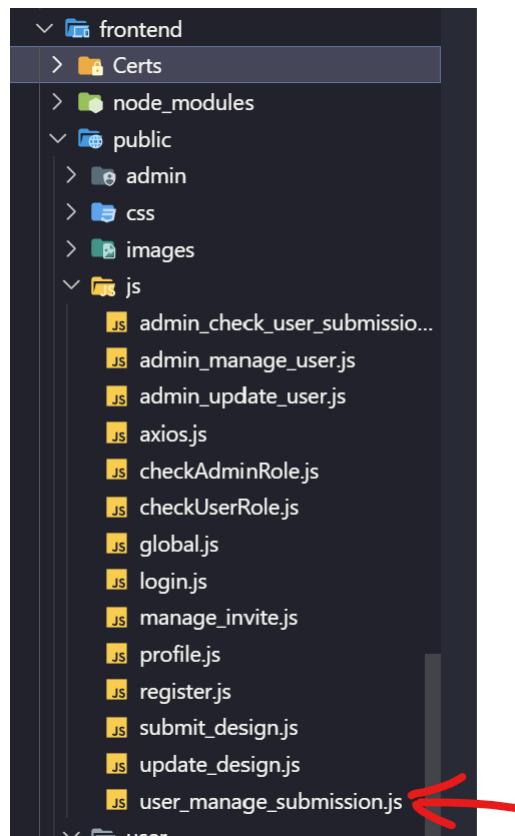
Step 3:

Head over to routes.js and we will implement this middleware to process-search-design endpoint. This is so that we can make sure that only the logged in user can view their own submission.

```
27 // USER SEARCH SUBMISSION
28 router.get(
29   "/api/user/process-search-design/:pagenumber/:search?",
30   checkUserFnSolution.checkForValidUser, ↖
31   validateFn.validateUserSearchOwnSubmission,
32   userController.processGetSubmissionData
33 );
```

Step 4:

Go to the front end folder and look for user_manage_submission.js under frontend/public/js/user_manage_submission.js.



Step 5:

Add the following into the headers for us to retrieve and use the token from the storage from the frontend.

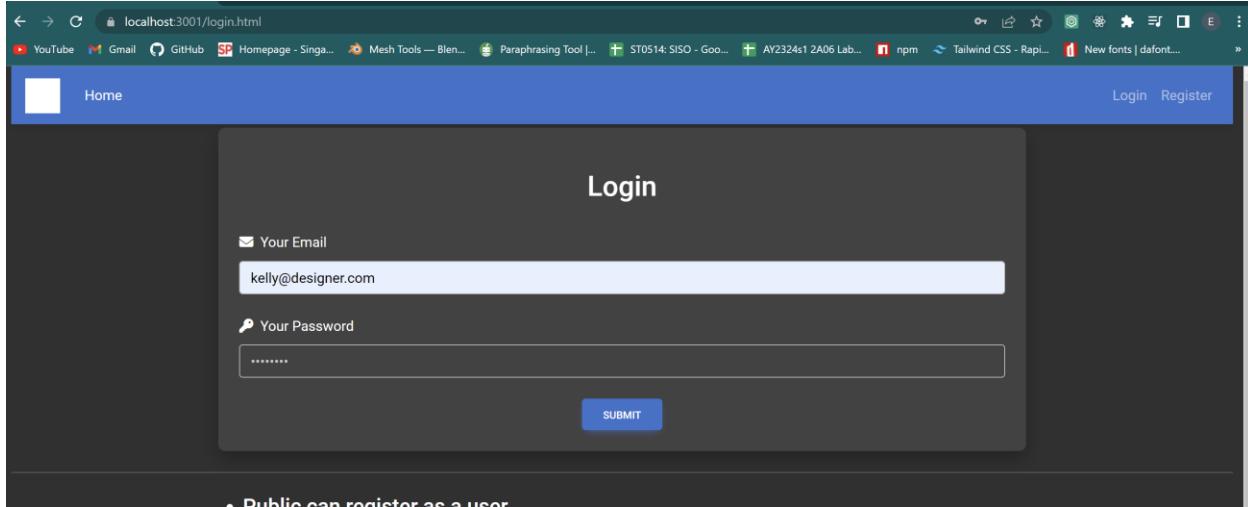
```
routes.js      js checkAdminRole.js      js checkUserRole.js      js checkUserFnSolution.js      js validateFn.js      js submit_design.js      js update_design.js      js user_manage_submission.js X
frontend > public > js > js user_manage_submission.js > clickHandlerForPageButton > token
1  let $searchDesignFormContainer = $("#searchDesignFormContainer");
2  if ($searchDesignFormContainer.length != 0) {
3    console.log(
4      "Search design form detected in user manage submission interface. Binding event handling logic to form elements."
5    );
6    //If the jQuery object which represents the form element exists,
7    //the following code will create a method to send key-value pair information to do record searching
8    //to server-side api when the #submitButton element fires the click event.
9    $("#submitButton").on("click", function (event) {
10      event.preventDefault();
11      const baseUrl = "https://localhost:5000";
12      let searchInput = $("#searchInput").val();
13      let userId = sessionStorage.getItem("user_id");
14      let token = sessionStorage.getItem("token"); ←
15      axios({
16        headers: {
17          user: userId,
18          authorization: `Bearer ${token}` ←
19        },
20        method: "get",
21        url: baseUrl + "/api/user/process-search-design/1/" + searchInput,
22      })
23        .then(function (response) {
24          //console.log(response)
25          //Using the following to inspect the response.data data structure
26          //before deciding the code which dynamically generates cards.
27          //Each card describes a design record.
28      })
29    });
30  });
31  //I have hard code 3 buttons for the manage-submission interface (user role).
32  //to cut down the JavaScript code for this file.
33  //If the jQuery object which represents the form element exists,
34  //the following code will create a method to make a HTTP GET
35  //to server-side api.
36  function clickHandlerForPageButton(event) {
37    event.preventDefault();
38    const baseUrl = "https://localhost:5000";
39    let userId = sessionStorage.getItem("user_id");
40    let token = sessionStorage.getItem("token"); ←
41    let pageNumber = $(event.target).text().trim();
42    let searchInput = $("#searchInput").val();
43    console.log(pageNumber);
44    axios({
45      headers: {
46        user: userId,
47        authorization: `Bearer ${token}` ←
48      },
49      method: "get",
50      url:
51        baseUrl +
52        "/api/user/process-search-design/" +
53        pageNumber +
54        "/" +
55        searchInput,
56    })
57    .then(function (response) {
58      //Using the following to inspect the response.data data structure
59      //before deciding the code which dynamically generates cards.
60      //Each card describes a design record.
61    })
62  };
63
```

```
routes.js      js checkAdminRole.js      js checkUserRole.js      js checkUserFnSolution.js      js validateFn.js      js submit_design.js      js update_design.js      js user_manage_submission.js X
frontend > public > js > js user_manage_submission.js > clickHandlerForPageButton > token
106    }).show();
107  });
108  });
109  //I have hard code 3 buttons for the manage-submission interface (user role)
110  //to cut down the JavaScript code for this file.
111  //If the jQuery object which represents the form element exists,
112  //the following code will create a method to make a HTTP GET
113  //to server-side api.
114  function clickHandlerForPageButton(event) {
115    event.preventDefault();
116    const baseUrl = "https://localhost:5000";
117    let userId = sessionStorage.getItem("user_id");
118    let token = sessionStorage.getItem("token"); ←
119    let pageNumber = $(event.target).text().trim();
120    let searchInput = $("#searchInput").val();
121    console.log(pageNumber);
122    axios({
123      headers: {
124        user: userId,
125        authorization: `Bearer ${token}` ←
126      },
127      method: "get",
128      url:
129        baseUrl +
130        "/api/user/process-search-design/" +
131        pageNumber +
132        "/" +
133        searchInput,
134    })
135    .then(function (response) {
136      //Using the following to inspect the response.data data structure
137      //before deciding the code which dynamically generates cards.
138      //Each card describes a design record.
139    })
140  };
141
```

3.4 Evidence that the vulnerability has been rectified

Step 1:

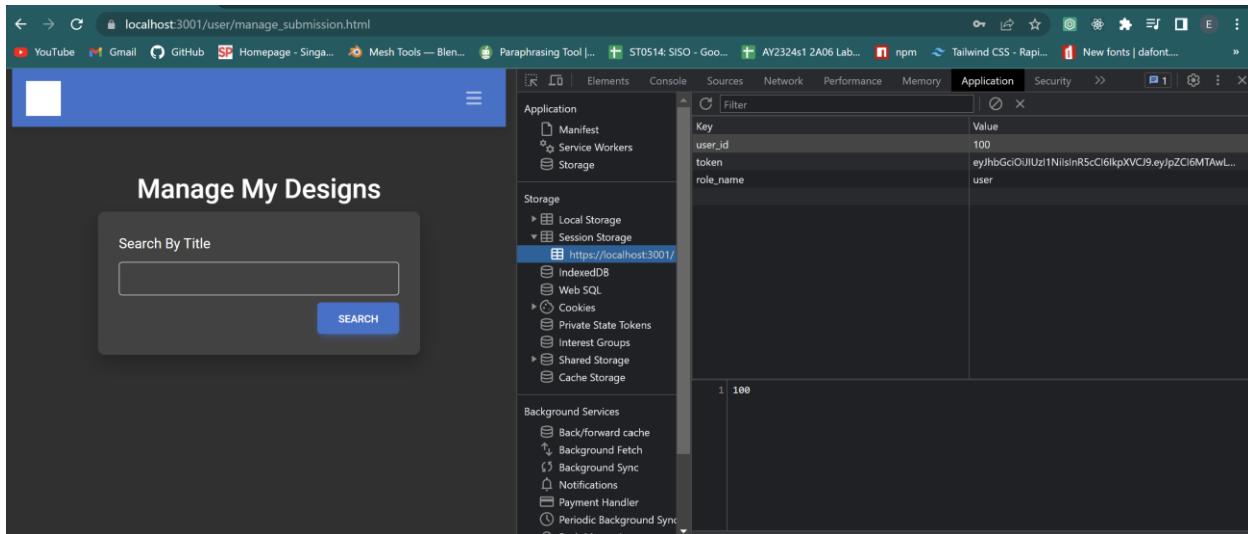
Login to the vulnerable website using Kelly's email and password.



Step 2:

Right Click and open the inspect element as shown below and navigate to Application > Local Storage.

*Note: I used Session Storage for my patched code, but you can use local storage.



Step 3:

Modify the user_id to 101 to see if we can view bezos's submissions this time.

You will see that we are still retrieving Kelly's Submissions as we used the token for the user id and not the user_id from the storage. This shows that we have successfully rectified the Broken Authentication Vulnerability.

The screenshot shows a browser window with the URL `localhost:3001/user/manage_submission.html`. On the left, there is a web application titled "Manage My Designs" with a search bar and two design thumbnails: "Kelly Design 1" and "Kelly Design 2". A red arrow points from the "Kelly Design 2" thumbnail towards the developer tools on the right. On the right, the developer tools Application tab is open, showing the Session Storage. The "user_id" key has a value of "101", which is highlighted with a red arrow. The "token" key has a long string of characters, and the "role_name" key has a value of "user".

4. Broken Access Control

4.1 Definition

Broken Access Control is a web application vulnerability that occurs when access controls and authorization mechanisms are improperly implemented, allowing unauthorized users to access restricted resources or perform actions beyond their intended privileges. It involves the failure to properly enforce restrictions on what authenticated users can access or modify within the application.

When access controls are broken or flawed, it can lead to various security issues like Unauthorized Data Access, Horizontal Privilege Escalation, Vertical Privilege Escalation, Insecure Direct Object Reference, and Mass Assignment.

4.3.1 How to exploit vulnerability - About Role access control

Step 1:

Change the search URL above to the following.

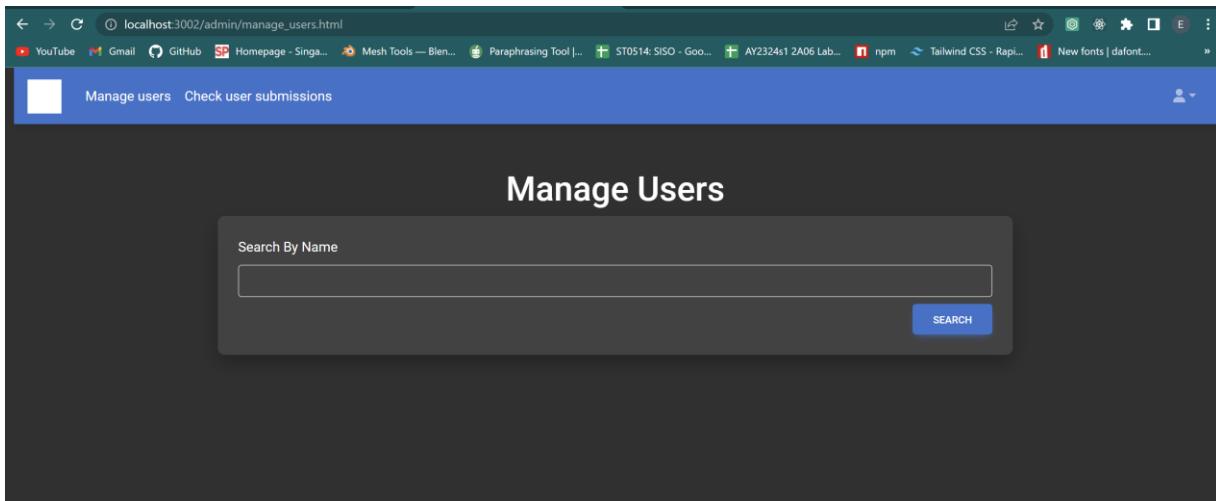
yourLocalhostPort/admin/manage_users.html



Step 2:

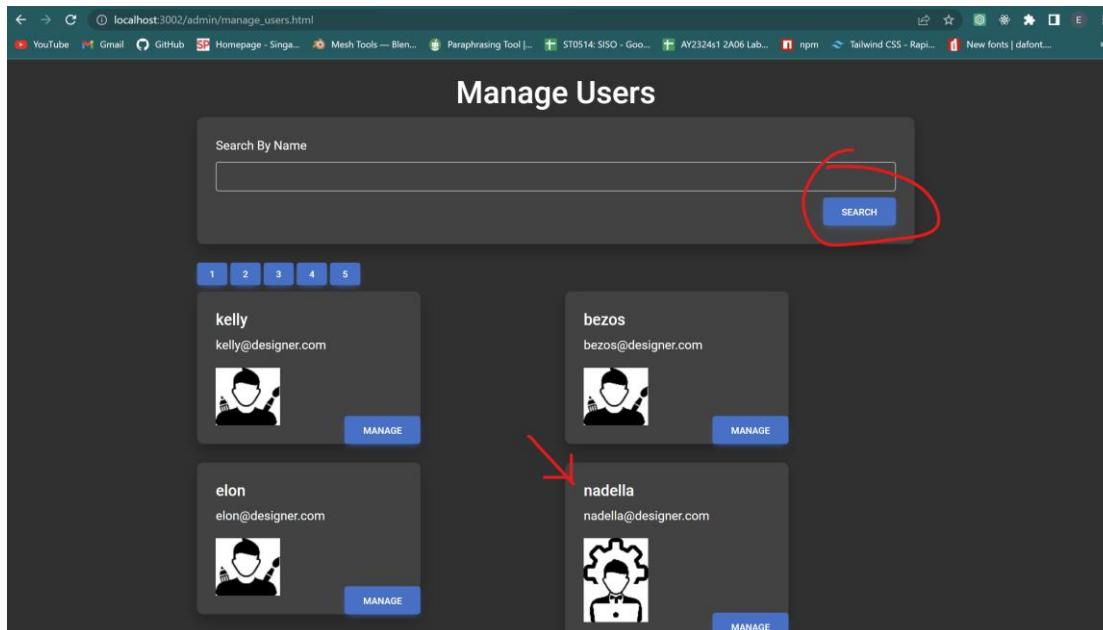
You should be brought to the admin page which we are not allowed to.

*Note: this is a 2nd vulnerability for Broken Access Control which is being rectified above under Bonus 7.2.0.



Step 3:

Click on the search button and look for a user called Nadella, click on Manage Button for Nadella to see her role.



Step 4:

You should be able to see Nadella's role which is an administrator for my case.

A screenshot of a web browser window titled "localhost:3002/admin/update_user.html?id=103". The main content area is titled "Update user". It displays a user profile for "nadella" with the email "nadella@designer.com". A dropdown menu shows the current role as "Administrator". Below the dropdown are two buttons: "BACK" and "SUBMIT". Two red arrows point to the "Administrator" role in the dropdown and the "SUBMIT" button.

About this interface: The administrator role can make changes to the role so that he can set another person as an

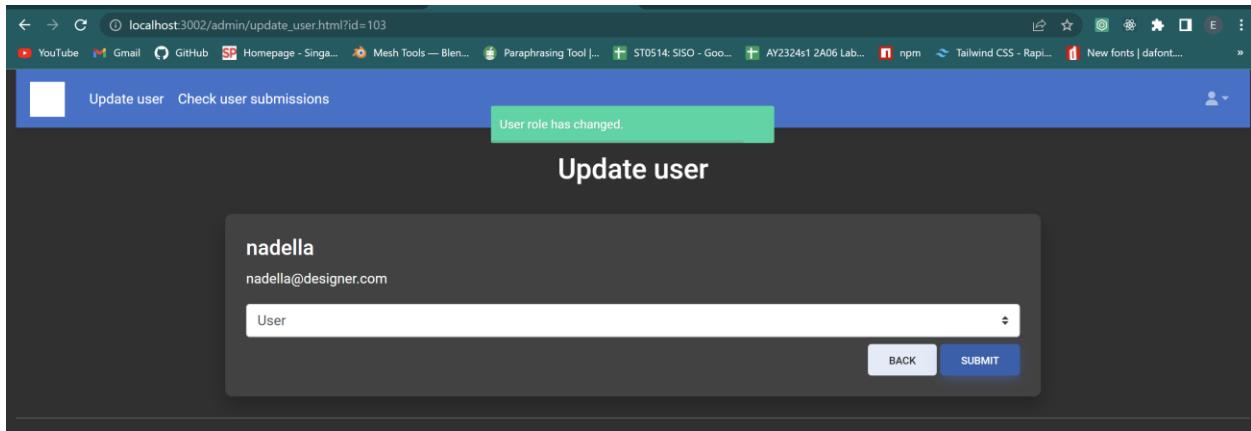
Step 5:

Let's change her role to User and submit.

A screenshot of a web browser window titled "localhost:3002/admin/update_user.html?id=103". The main content area is titled "Update user". It displays a user profile for "nadella" with the email "nadella@designer.com". The dropdown menu now shows the role as "User". Below the dropdown are two buttons: "BACK" and "SUBMIT". The "SUBMIT" button is circled in red.

Step 6:

You should see that you are able to change the role of Nadella even though we are not logged in.

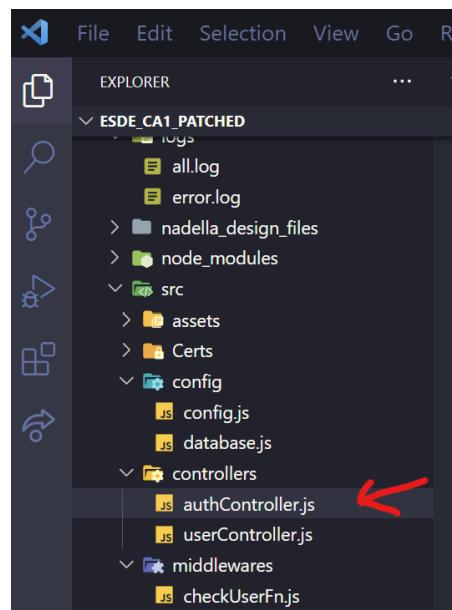


4.3.2 How to rectify vulnerability - About Role access control

Step 1:

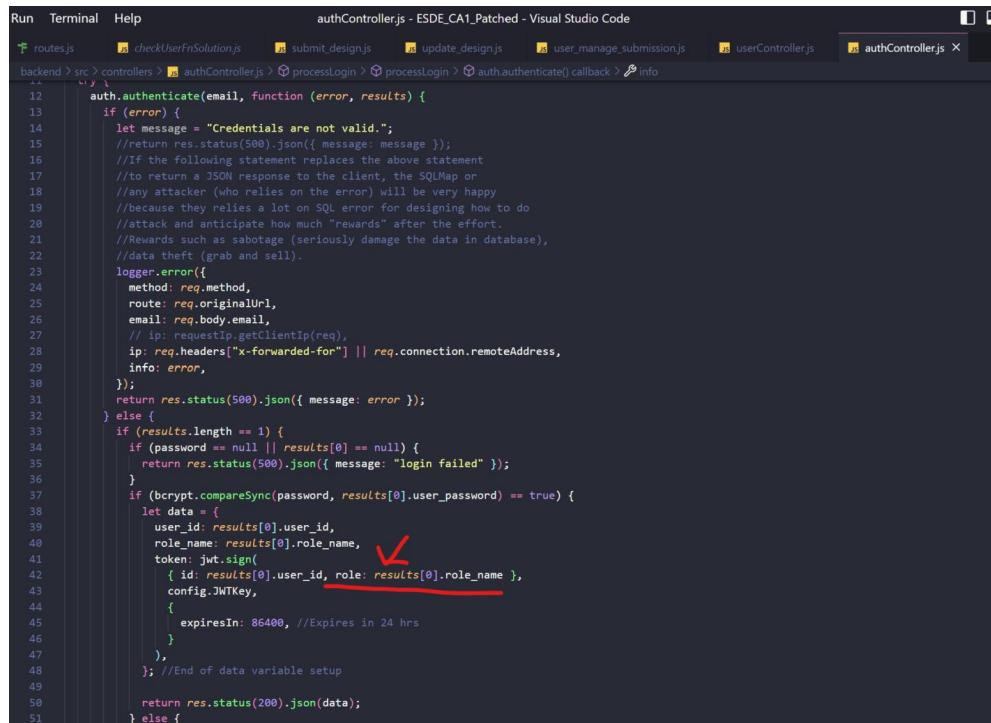
To rectify this vulnerability, let us first go to authController.js to do some changes to the cookie.

Under /src/controllers/authController.js



Step 2:

Apply the following changes to the exports.processLogin under /src/controllers/authController.js. This will allow us to check the role of the current user to see if they are valid to perform any admin actions from the token.

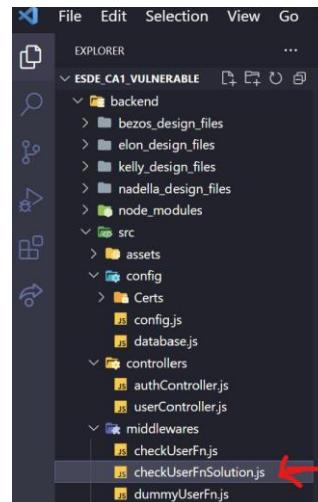


```
authController.js - ESDE_CA1_Patched - Visual Studio Code
Run Terminal Help
backend > src > controllers > authController.js > processLogin > processLogin > auth.authenticate() callback > info
routes.js authController.js submit_designs update_design.js user_manage_submission.js UserController.js authController.js

12     auth.authenticate(email, function (error, results) {
13       if (error) {
14         let message = "Credentials are not valid.";
15         //If the following statement replaces the above statement
16         //to return a JSON response to the client, the SQLMap or
17         //any attacker (who relies on the error) will be very happy
18         //because they relies a lot on SQL error for designing how to do
19         //attack and anticipate how much "rewards" after the effort.
20         //Rewards such as sabotage (seriously damage the data in database),
21         //data theft (grab and sell).
22         logger.error({
23           method: req.method,
24           route: req.originalUrl,
25           email: req.body.email,
26           ip: req.ip || req.headers['x-forwarded-for'] || req.connection.remoteAddress,
27           info: error,
28         });
29       }
30       return res.status(500).json({ message: error });
31     } else {
32       if (results.length == 1) {
33         if (password == null || results[0] == null) {
34           return res.status(500).json({ message: "login failed" });
35         }
36         if (bcrypt.compareSync(password, results[0].user_password) == true) {
37           let data = {
38             user_id: results[0].user_id,
39             role_name: results[0].role_name,
40             token: jwt.sign(
41               { id: results[0].user_id, role: results[0].role_name },
42               config.JWTKey,
43               {
44                 expiresIn: 86400, //Expires in 24 hrs
45               }
46             ),
47           };
48           return res.status(200).json(data);
49         } else {
50       }
51     }
52   }
53 }
```

Step 3:

Head over to checkUserFnSolution.js to add a check admin role middleware. Under /src/middlewares/ checkUserFnSolution.js



Step 4:

Add the following code to create a middleware to check if the current logged in user is an admin or not by using the token.

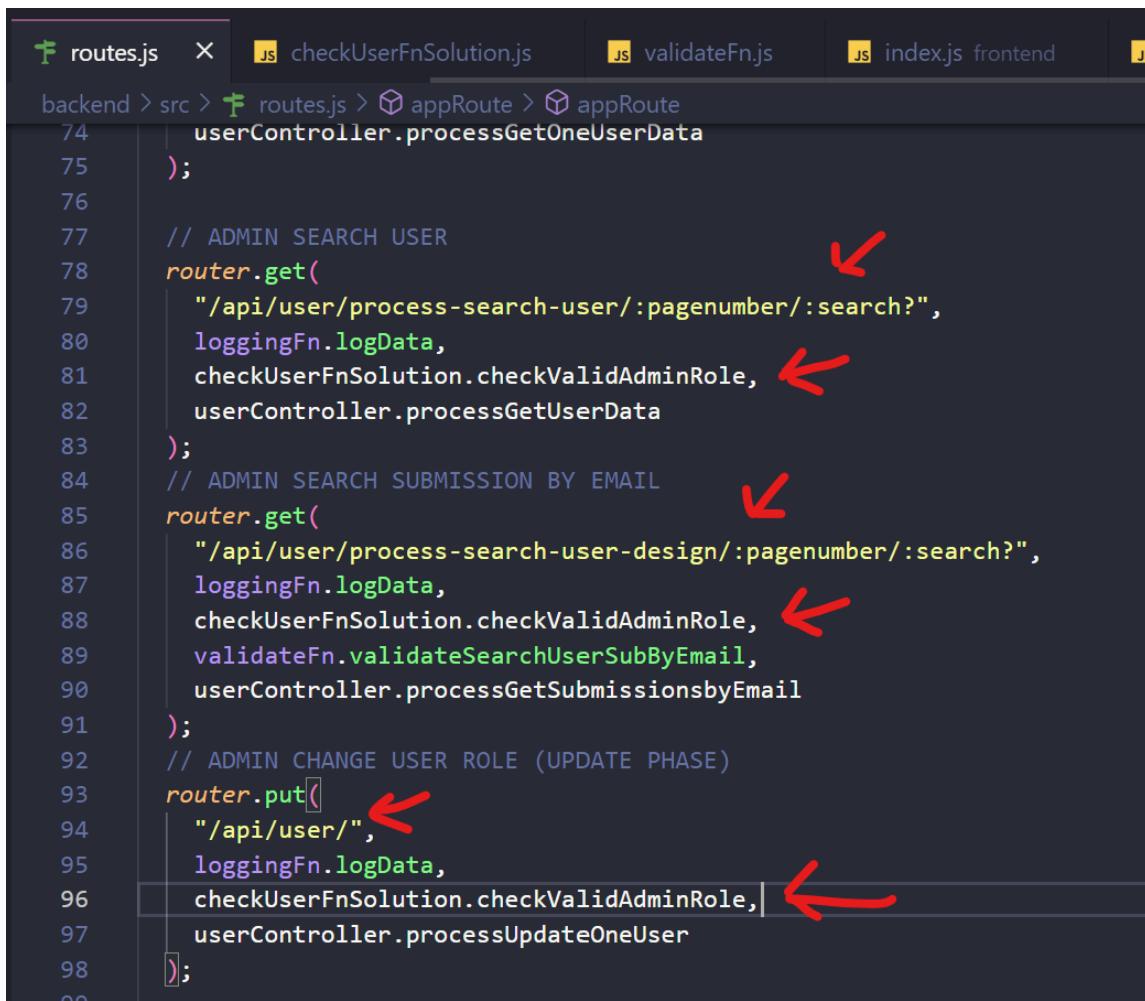


```
routes.js          checkUserFnSolution.js X      submit_design.js      update_design.js      user_manage
backend > src > middlewares > checkUserFnSolution.js > checkValidAdminRole > checkValidAdminRole

70
71 module.exports.checkValidAdminRole = (req, res, next) => {
72   console.log("http header - admin ", req.headers["user"]);
73   console.log("authHeader : " + req.headers["authorization"]);
74   const authHeader = req.headers["authorization"];
75   if (
76     authHeader === null ||
77     authHeader === undefined ||
78     !authHeader.startsWith("Bearer "))
79   ) {
80     console.log(
81       "authHeader is null or undefined or does not start with Bearer"
82     );
83     return res.status(403).send({ message: "Unauthorized access" });
84   } else {
85     //Retrieve authorization header and parse out the JWT using the split function
86     console.log("Retrieving authorization header");
87     let token = authHeader.split(" ")[1];
88     console.log("Check for received token from frontend : \n");
89     console.log(token);
90
91     jwt.verify(token, config.JWTKey, (err, data) => {
92       console.log("data extracted from token \n", data);
93       if (err) {
94         console.log(err);
95         return res.status(403).send({ message: "Unauthorized access" });
96       } else {
97         console.log("data: " + data.role);
98         if (data.role == "admin") {
99           console.log("User is admin");
100          next();
101        } else {
102          console.log("User is not admin");
103          return res.status(403).send({
104            message: "Unauthorized access! You are not an admin.",
105          });
106        }
107      }
108    });
109  }
110}; //End of checkValidAdminRole
111
```

Step 5:

Next, head over to routes.js under /backend/src/routes.js where we will use the middleware we just created to the admin endpoints. Check that you have added the middleware to the correct endpoints with checkUserFnSolution.checkValidAdminRole.



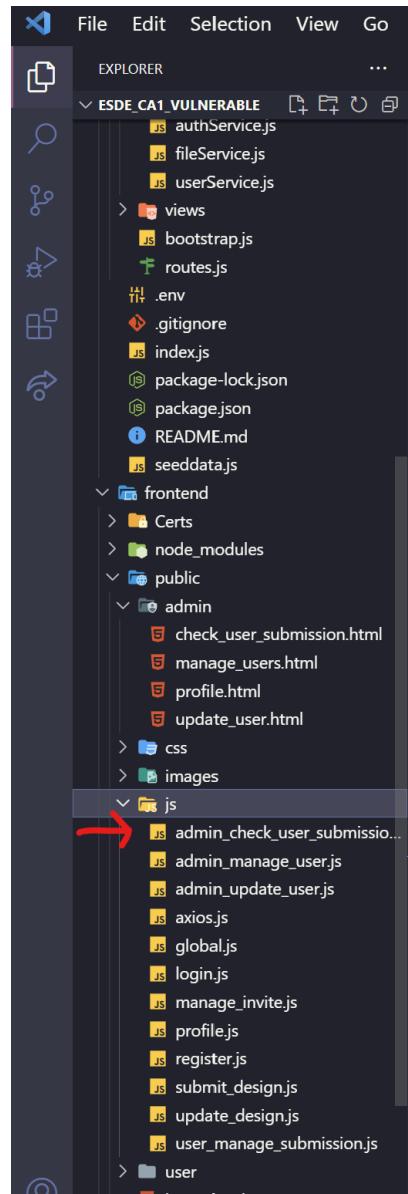
```
routes.js  X  checkUserFnSolution.js  validateFn.js  index.js frontend  JS

backend > src > routes.js > appRoute > appRoute
74     userController.processGetOneUserData
75   );
76
77   // ADMIN SEARCH USER
78   router.get(
79     "/api/user/process-search-user/:pagenumber/:search?",
80     loggingFn.logData,
81     checkUserFnSolution.checkValidAdminRole, ←
82     userController.processGetUserData
83   );
84   // ADMIN SEARCH SUBMISSION BY EMAIL
85   router.get(
86     "/api/user/process-search-user-design/:pagenumber/:search?",
87     loggingFn.logData,
88     checkUserFnSolution.checkValidAdminRole, ←
89     validateFn.validateSearchUserSubByEmail,
90     userController.processGetSubmissionsByEmail
91   );
92   // ADMIN CHANGE USER ROLE (UPDATE PHASE)
93   router.put(
94     "/api/user/",
95     loggingFn.logData,
96     checkUserFnSolution.checkValidAdminRole, ←
97     userController.processUpdateOneUser
98   );
99
```

Step 6:

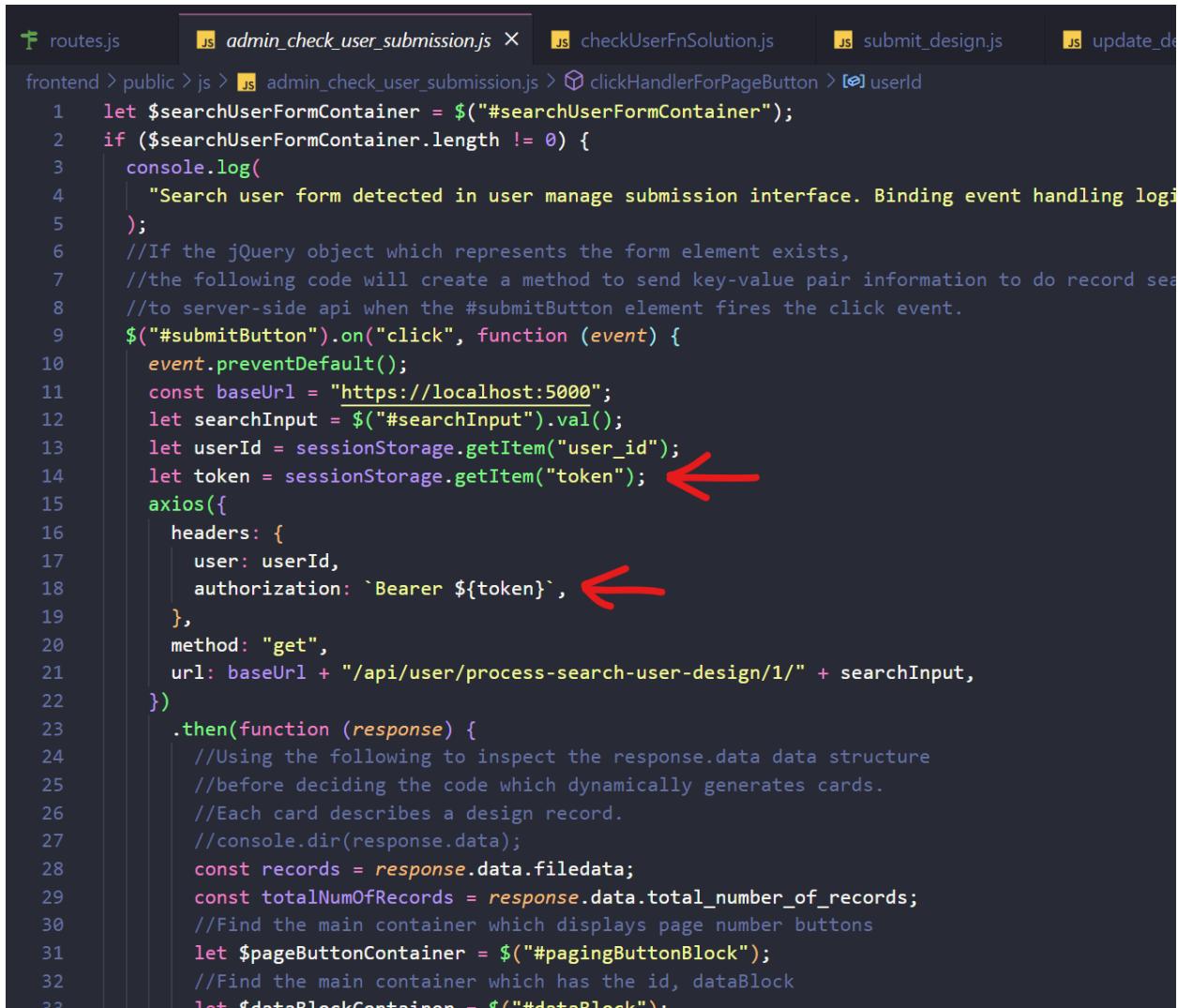
Next, go over to the Frontend Directory and look for admin_check_user_submission.js.

Under /frontend/public/js/admin_check_user_submission.js



Step 7:

We will now add a code to retrieve the token from the local storage and place it in the headers as shown below.



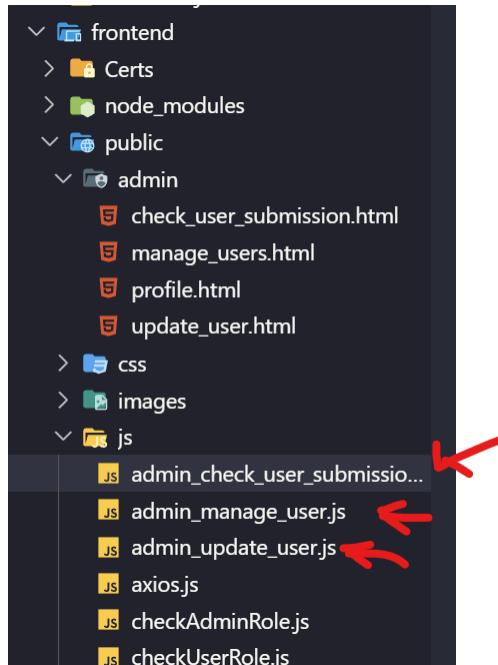
```
routes.js      admin_check_user_submission.js X      checkUserFnSolution.js      submit_designjs      update_de
frontend > public > js > admin_check_user_submission.js > clickHandlerForPageButton > [o] userId
1  let $searchUserFormContainer = $("#searchUserFormContainer");
2  if ($searchUserFormContainer.length != 0) {
3    console.log(
4      "Search user form detected in user manage submission interface. Binding event handling logic."
5    );
6    //If the jQuery object which represents the form element exists,
7    //the following code will create a method to send key-value pair information to do record search
8    //to server-side api when the #submitButton element fires the click event.
9    $("#submitButton").on("click", function (event) {
10      event.preventDefault();
11      const baseUrl = "https://localhost:5000";
12      let searchInput = $("#searchInput").val();
13      let userId = sessionStorage.getItem("user_id");
14      let token = sessionStorage.getItem("token"); ←
15      axios({
16        headers: {
17          user: userId,
18          authorization: `Bearer ${token}` , ←
19        },
20        method: "get",
21        url: baseUrl + "/api/user/process-search-user-design/1/" + searchInput,
22      })
23      .then(function (response) {
24        //Using the following to inspect the response.data data structure
25        //before deciding the code which dynamically generates cards.
26        //Each card describes a design record.
27        //console.dir(response.data);
28        const records = response.data.filidata;
29        const totalNumOfRecords = response.data.total_number_of_records;
30        //Find the main container which displays page number buttons
31        let $pageButtonContainer = $("#pagingButtonBlock");
32        //Find the main container which has the id, dataBlock
33        let $dataBlockContainer = $("#dataBlock");
```

Step 8:

Continue doing the above changes for all the admin js files that uses axios like admin_check_user_submission, admin_manage_user, and admin_update_user.

*Note: There should be a total of six changes to the 3 files including the above changes.

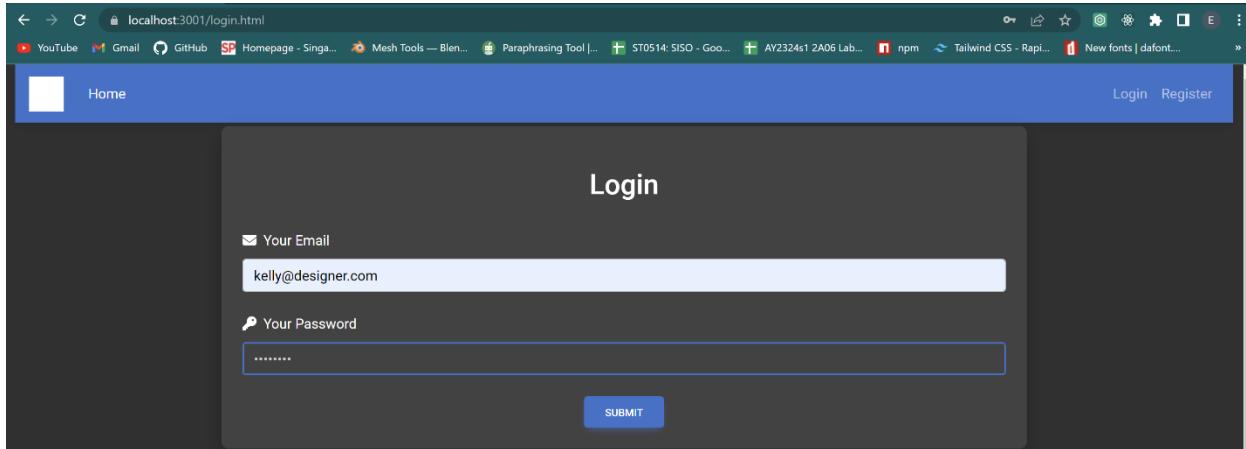
Under /frontend/public/js/.



4.4.2 Evidence that the vulnerability has been rectified - About Role access control

Step 1:

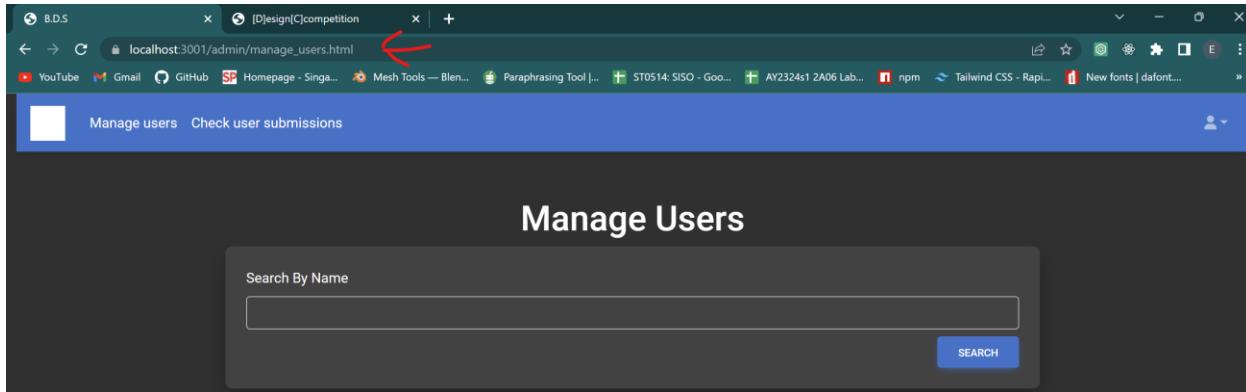
Login using Kelly's email and password in the Login Page.



The screenshot shows a web browser window with the URL `localhost:3001/login.html`. The page has a dark background with a blue header bar. In the header, there is a 'Home' button and links for 'Login' and 'Register'. The main content area is titled 'Login'. It contains two input fields: one for 'Your Email' with the value `kelly@designer.com` and another for 'Your Password' with redacted text. Below the password field is a 'SUBMIT' button.

Step 2:

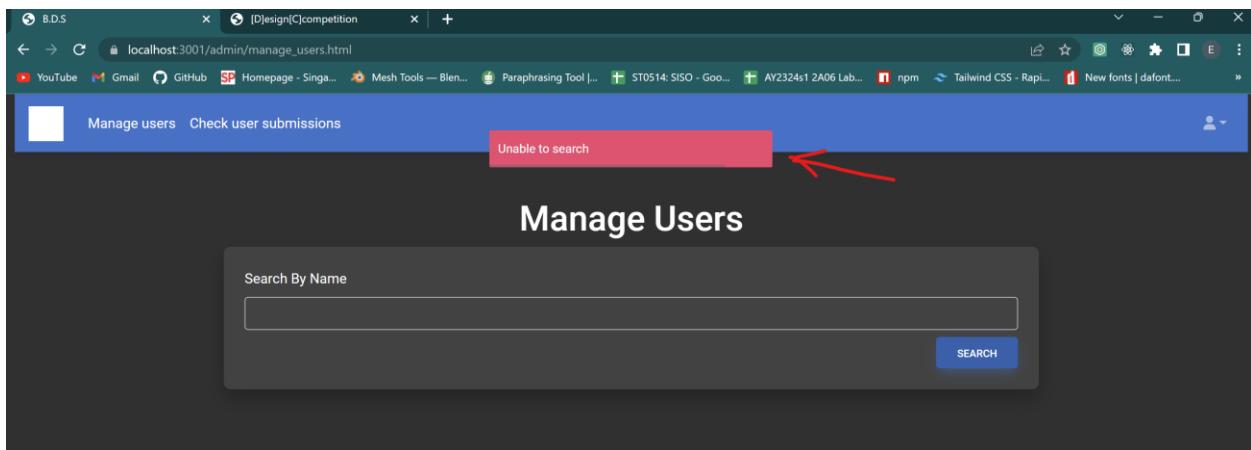
Change the above URL link to `yourLocalhostPort/admin/manage_users.html`.



The screenshot shows a web browser window with the URL `localhost:3001/admin/manage_users.html`. The address bar is highlighted with a red arrow. The page has a dark background with a blue header bar. In the header, there is a 'Manage users' button and a user icon. The main content area is titled 'Manage Users'. It contains a search bar with the placeholder text 'Search By Name' and a 'SEARCH' button.

Step 3:

Now, after applying the checkValidAdminRole middleware to all the admin endpoints, you will immediately see that upon pressing the search button it will show an alert saying, “Unable to search”. This shows that we have successfully validated the role of the user and not allow non-admin users to do any admin right actions even though they can access the admin page.



5. Sensitive Data Exposure

5.1 Definition

Sensitive Data Exposure is a security vulnerability where confidential or sensitive information is exposed to unauthorized individuals or entities due to inadequate protection or improper handling of the data. It occurs when sensitive data, such as personal information, financial details, passwords, or intellectual property, is stored, transmitted, or processed without appropriate security measures in place.

This can occur through various means like Insecure Data Storage, Inadequate Transmission Security, Weak Authentication and Authorization, Poorly Configured Security Controls, and Vulnerabilities in Applications or Systems.

5.2 How to exploit vulnerability

Step 1:

Install Wireshark as we will be using Wireshark for this vulnerability.

Link: <https://www.wireshark.org/download.html>

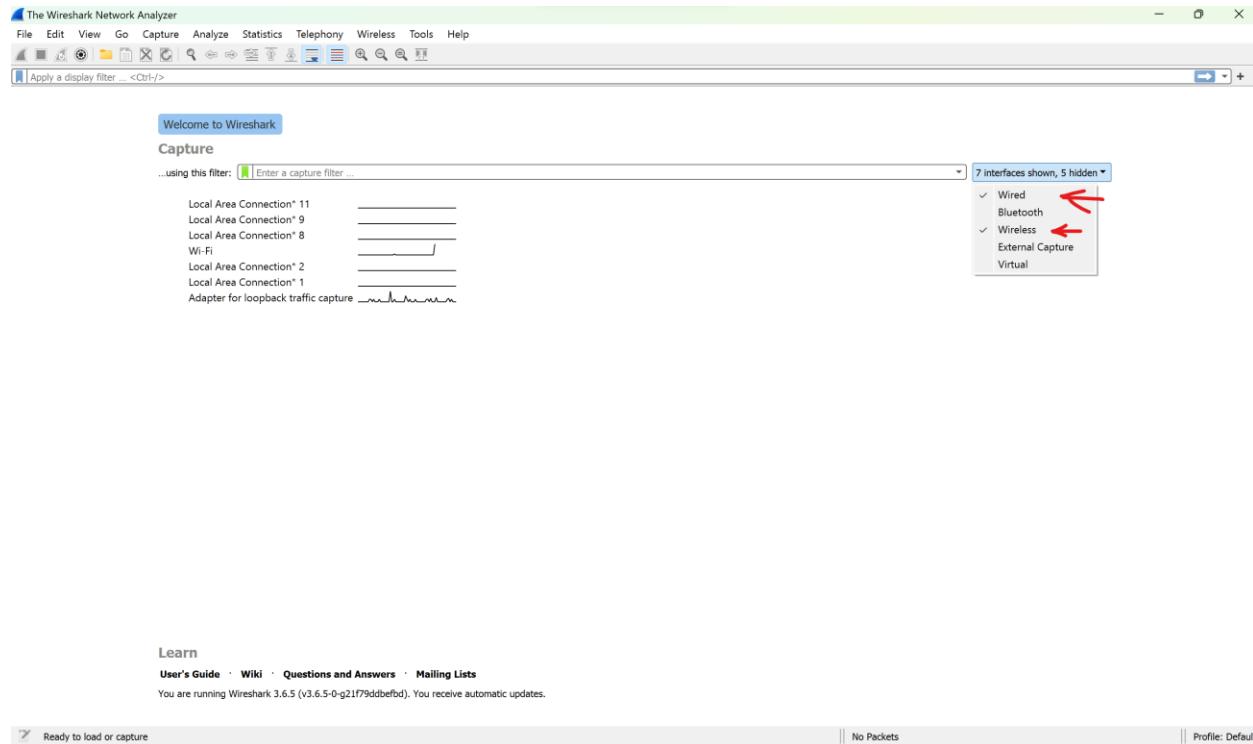
Step 2:

Head over to the vulnerable website fill in Kelly's Email and Password as shown below **but do not press submit.**

The screenshot shows a web browser window with the URL `localhost:3002/login.html`. The page title is "Login". It features a form with two text input fields: one for "Your Email" containing "kelly@designer.com" and one for "Your Password" containing "*****". Below the password field is a blue "SUBMIT" button. At the top of the page, there is a navigation bar with "Home", "Login", and "Register" links. The browser's address bar also displays the URL.

Step 3:

Head over to Wireshark where you will first see the image shown below and make sure that you have checked Wired and Wireless.



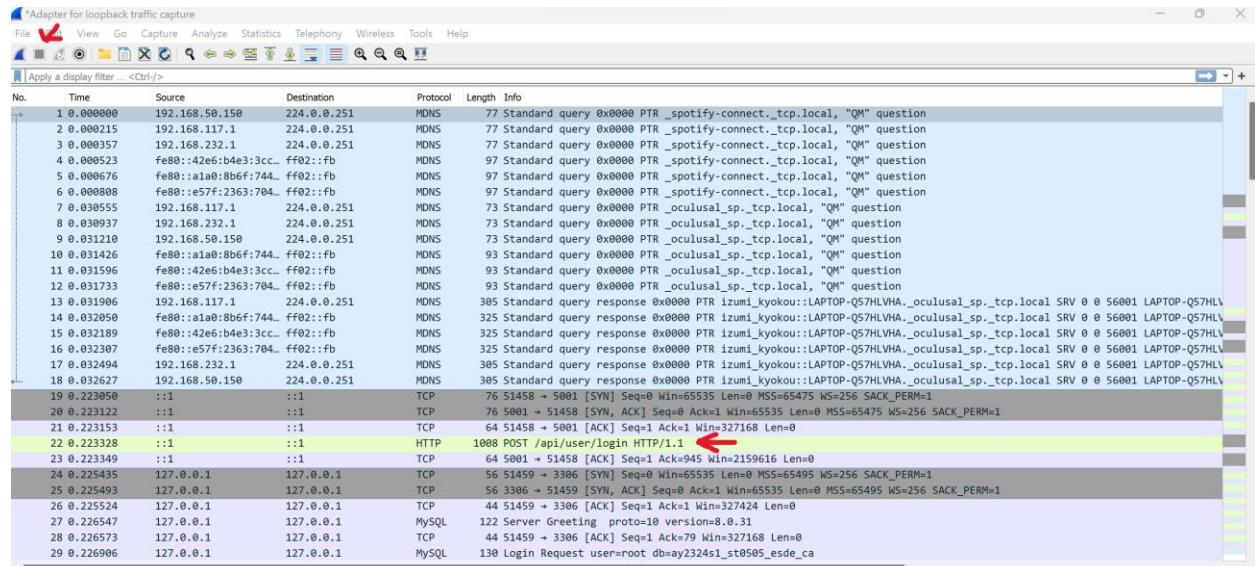
Step 4:

Next click on Adapter for loopback traffic capture and press the shark fin looking icon on the top left to begin capturing the traffic.



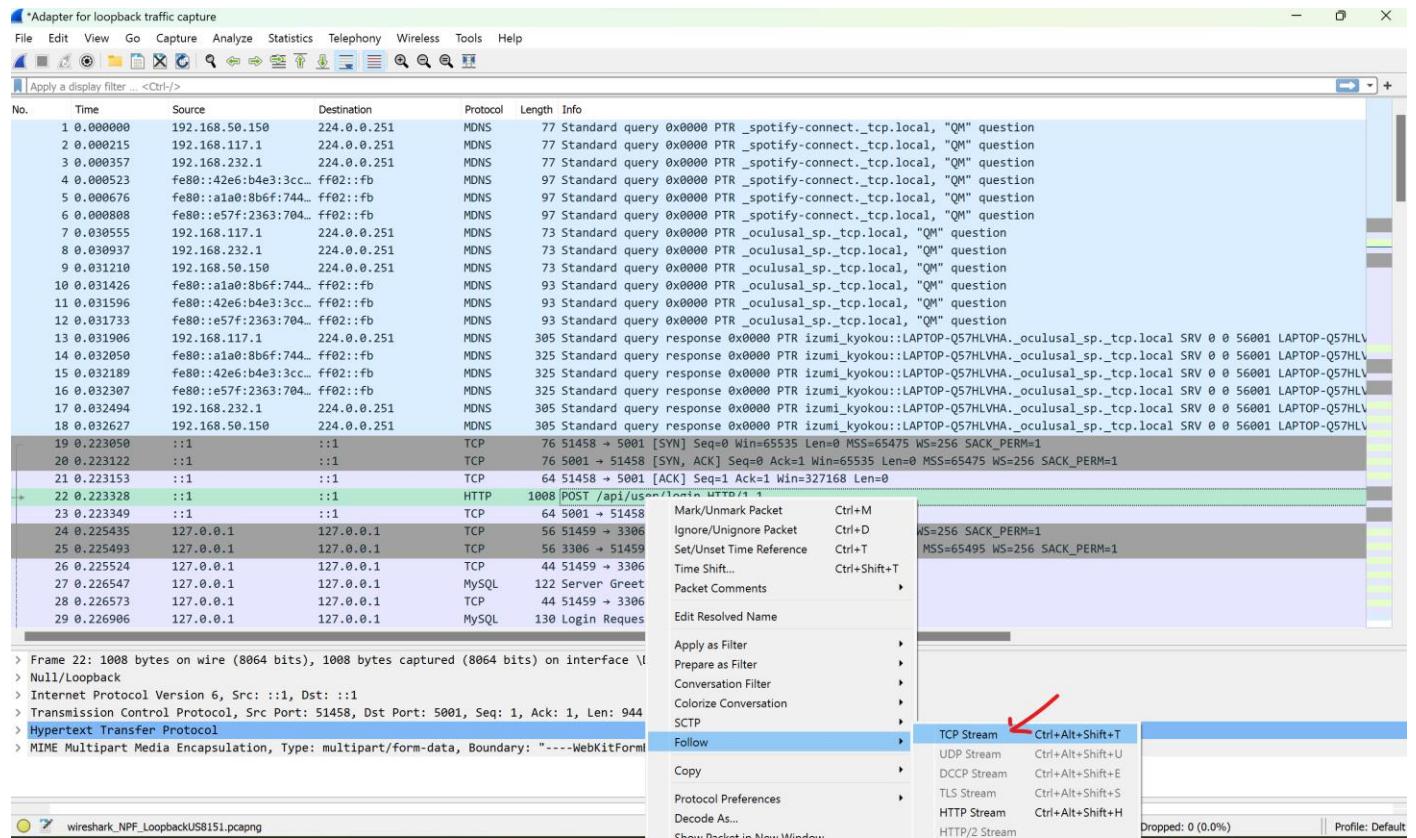
Step 5:

Next, go back to the vulnerable website and press submit to log in and go to Wireshark and press the stop button beside the shark fin looking button. You should see a lot of traffic being captured and one of it being the POST request for our login.



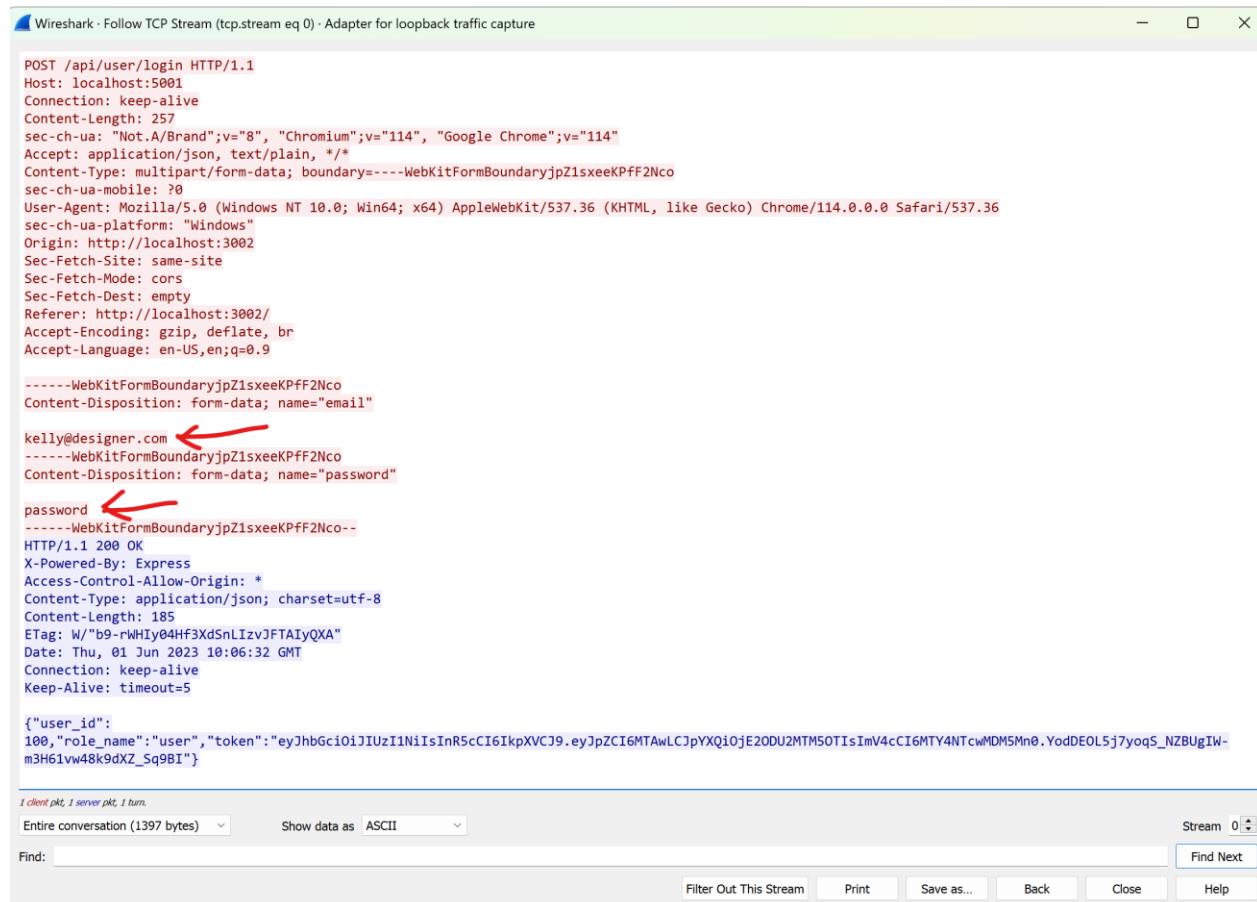
Step 6:

Next, right click on the POST request and select Follow > TCP Stream.



Step 7:

You should be able to see the login details in plain text as shown below.



The screenshot shows a Wireshark capture of an HTTP POST request to '/api/user/login'. The request includes user credentials in plain text:

```
POST /api/user/login HTTP/1.1
Host: localhost:5001
Connection: keep-alive
Content-Length: 257
sec-ch-ua: "Not_A/Brand";v="8", "Chromium";v="114", "Google Chrome";v="114"
Accept: application/json, text/plain, */*
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryjpZ1sxeeKPff2Nco
sec-ch-ua-mobile: ?
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0 Safari/537.36
sec-ch-ua-platform: "Windows"
Origin: http://localhost:3002
Sec-Fetch-Site: same-site
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:3002/
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9

-----WebKitFormBoundaryjpZ1sxeeKPff2Nco
Content-Disposition: form-data; name="email"
kelly@designer.com ←

-----WebKitFormBoundaryjpZ1sxeeKPff2Nco
Content-Disposition: form-data; name="password"
password ←

-----WebKitFormBoundaryjpZ1sxeeKPff2Nco--
HTTP/1.1 200 OK
X-Powered-By: Express
Access-Control-Allow-Origin: *
Content-Type: application/json; charset=utf-8
Content-Length: 185
ETag: W/"b9-rWHy04HF3XdSnLIZvJFTAIyQXA"
Date: Thu, 01 Jun 2023 10:06:32 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{"user_id": 100, "role_name": "user", "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MTAwLCJpYXQiOjE2ODU2MTM5OTIsImV4cCI6MTY4NTcwMDM5Mn0.YodDEOL5j7yoqS_NZB0gIW-m3H61vw48k9dXZ_Sq9BI"}
```

Two red arrows point to the email address 'kelly@designer.com' and the password 'password' in the captured data.

5.3 How to rectify vulnerability

Step 1:

Let us install chocolatey so that we can install mkcert to make our site secure.

Type the following into your **Administrative Command Prompt**.

```
@ "%SystemRoot%\System32\WindowsPowerShell\v1.0\powershell.exe" -
NoProfile -InputFormat None -ExecutionPolicy Bypass -Command "
[System.Net.ServicePointManager]::SecurityProtocol = 3072; iex ((New-Object
System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))" &&
SET "PATH=%PATH%;%ALLUSERSPROFILE%\chocolatey\bin"
```

Reference: <https://www.liquidweb.com/kb/how-to-install-chocolatey-on-windows/>

Step 2:

Check that you have chocolatey installed by typing “choco –version”. For me, the version I am using is 1.1.0.

```
C:\Administrator: Command Prompt  
Microsoft Windows [Version 10.0.22621.1702]  
(c) Microsoft Corporation. All rights reserved.  
  
C:\Windows\System32>choco --version  
1.1.0
```

Step 3:

Next let us install mkcert to generate a self-signed certificate for us to use later.

Enter the following in order into your command prompt.

1. choco install mkcert
2. mkcert -install
3. mkcert localhost

Your screen should look something like the picture below.

Reference: <https://technixleo.com/create-locally-trusted-ssl-certificates-with-mkcert-on-windows/>

*Note: The following image has extra command used to check for the install location of the mkcert like “mkcert -CAROOT”. Please use this command if you cannot find the directory of the installed mkcert.

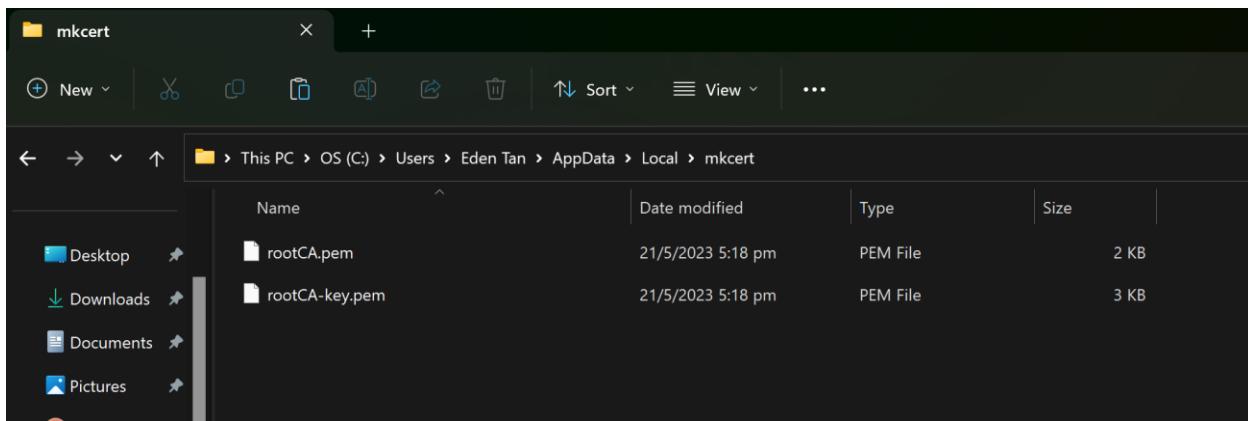
```
Command Prompt + - Microsoft Windows [Version 10.0.22621.1702] (c) Microsoft Corporation. All rights reserved. C:\Users\Eden Tan>mkcert --version v1.4.4 C:\Users\Eden Tan>mkcert -install Created a new local CA * The local CA is now installed in the system trust store! ⚡ Note: Firefox support is not available on your platform. ⓘ C:\Users\Eden Tan>mkcert -CAROOT C:\Users\Eden Tan\AppData\Local\mkcert C:\Users\Eden Tan>mkcert ESDE_CAI localhost 127.0.0.1 ::1 Created a new certificate valid for the following names 🌐 - "ESDE_CAI" - "localhost" - "127.0.0.1" - "::1" The certificate is at "./ESDE_CAI+3.pem" and the key at "./ESDE_CAI+3-key.pem" ✅ It will expire on 21 August 2025 ⓘ C:\Users\Eden Tan>
```

Step 4:

Let us go to the file path listed after typing the command `mkcert -CAROOT`.

You should see a `rootCA.pem` file and a `rootCA-key.pem` file that we created from the command “`mkcert -install`”.

Remember the location of the `rootCA.pem` as we will be using it for our trusted certificate in google chrome for my case.



Step 5:

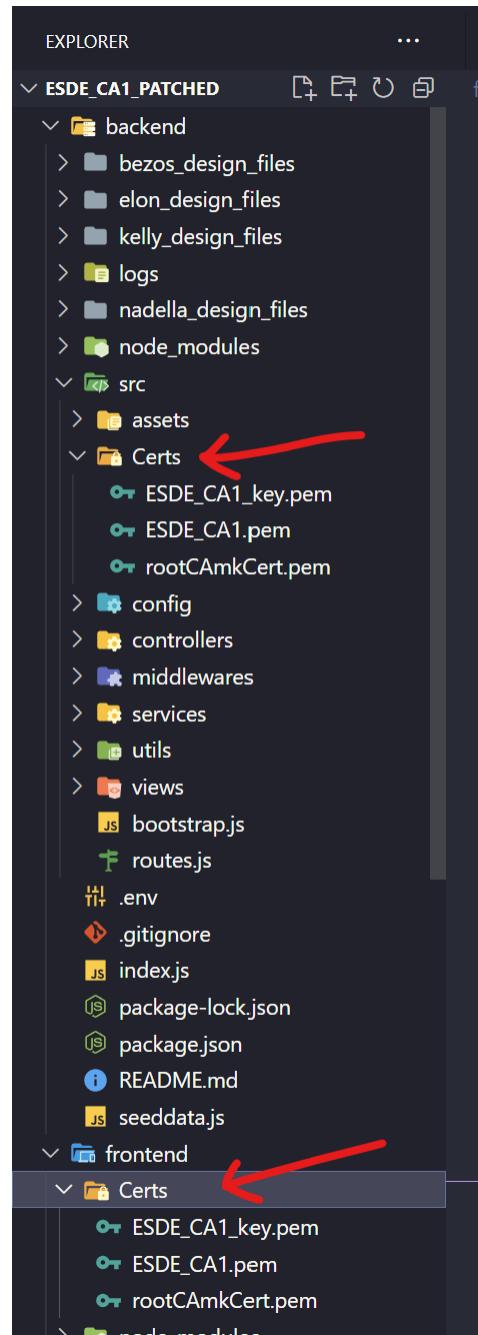
Next, we will go to the directory for my name in my case as shown below and we will see two .pem files that we can use to make our website secure which we got from the command “mkcert localhost”.

Name	Date modified	Type	Size
miniconda3	21/4/2023 3:32 pm	File folder	
Music	8/3/2023 10:00 am	File folder	
OneDrive	1/6/2023 12:26 pm	File folder	
Pictures	8/3/2023 10:00 am	File folder	
Postman	17/10/2022 1:40 pm	File folder	
Saved Games	8/3/2023 10:00 am	File folder	
Searches	8/3/2023 10:00 am	File folder	
source	18/10/2022 11:08 am	File folder	
Videos	8/3/2023 10:00 am	File folder	
.emulator_console_auth_token	17/10/2022 4:46 pm	EMULATOR_CONSOLE...	1 KB
.gitconfig	7/4/2022 6:46 pm	Git Config Source File	1 KB
.node_repl_history	22/4/2023 4:59 pm	NODE_REPL_HISTOR...	1 KB
.npmrc	26/11/2022 6:00 pm	NPMRC File	1 KB
ESDE_CA1+3.pem	21/5/2023 5:22 pm	PEM File	2 KB
ESDE_CA1+3-key.pem	21/5/2023 5:22 pm	PEM File	2 KB
lively_setup_x86_full_v1742	3/4/2022 12:45 pm	Application	199,749 KB

Step 6:

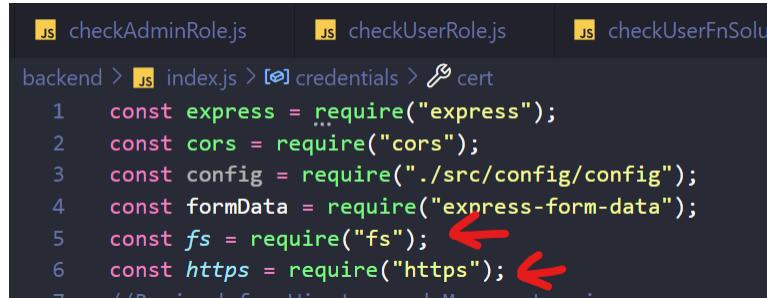
Copy the two .pem files and create a folder called Certs in both your backend and frontend folder and paste the .pem files inside.

*Note: I renamed the Cert and Key and added the rootCA.pem file here for convenience.



Step 7:

Next, go into index.js under /backend/src/index.js where we will use the certs we got. We will first use the built-in fs and https library from node.js.



```
checkAdminRole.js          checkUserRole.js          checkUserFnSolu...
backend > index.js > [e] credentials > cert
1  const express = require("express");
2  const cors = require("cors");
3  const config = require("./src/config/config");
4  const formData = require("express-form-data");
5  const fs = require("fs");           ←
6  const https = require("https");    ←
```

Step 8:

Next, let us add the following code to read the key and cert files.

```
23 //Read private key and certificate for HTTPS ~ Sensitive Data Exposure Prevention
24 const credentials = {
25   key: fs.readFileSync("./src/Certs/ESDE_CA1_key.pem"),
26   cert: fs.readFileSync("./src/Certs/ESDE_CA1.pem"),
27 };
28 //End of Sensitive Data Exposure Prevention Settings
29
```

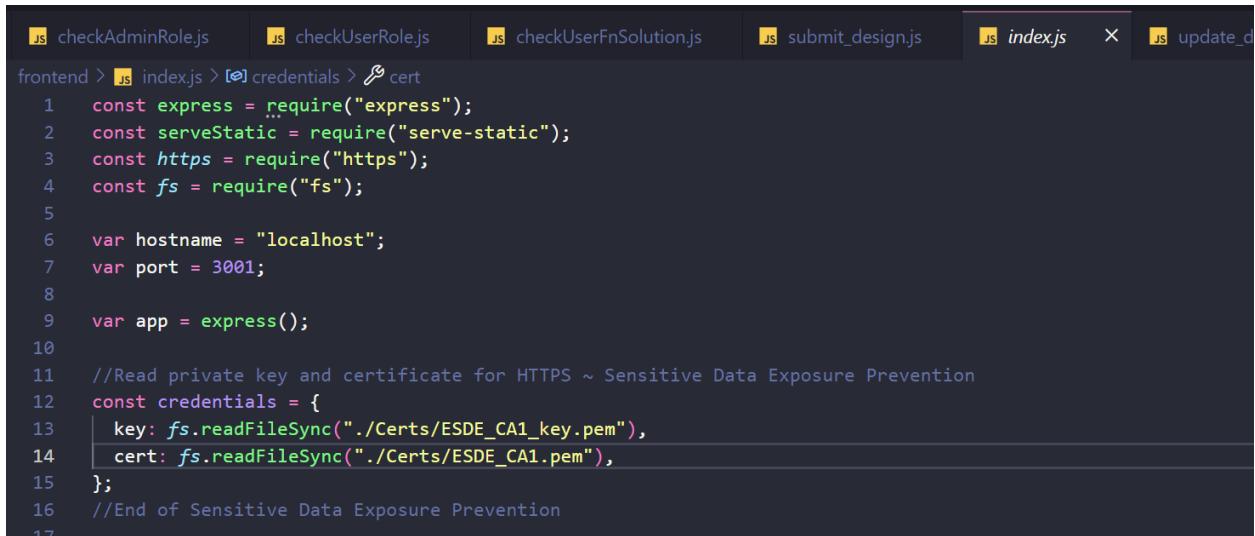
Step 9:

Next, we will add the following code at the bottom of the index.js file to create a https server for the backend.

```
92 // Create an HTTPS server with the 'credentials' for a secure connection
93 var httpsServer = https.createServer(credentials, app);
94 httpsServer.listen(PORT, (err) => {
95   if (err) return console.log(`Cannot Listen on PORT: ${PORT}`);
96   console.log(`Server is Listening on: https://localhost:${PORT}/`);
97 });
98
```

Step 10:

Next, head over to the index.js file under /frontend/public/index.js. We will also use the built in https and fs library and create the credentials like above.



```
frontend > index.js > [e] credentials > cert
1  const express = require("express");
2  const serveStatic = require("serve-static");
3  const https = require("https");
4  const fs = require("fs");
5
6  var hostname = "localhost";
7  var port = 3001;
8
9  var app = express();
10
11 //Read private key and certificate for HTTPS ~ Sensitive Data Exposure Prevention
12 const credentials = {
13   key: fs.readFileSync("./Certs/ESDE_CA1_key.pem"),
14   cert: fs.readFileSync("./Certs/ESDE_CA1.pem"),
15 };
16 //End of Sensitive Data Exposure Prevention
17
```

Step 11:

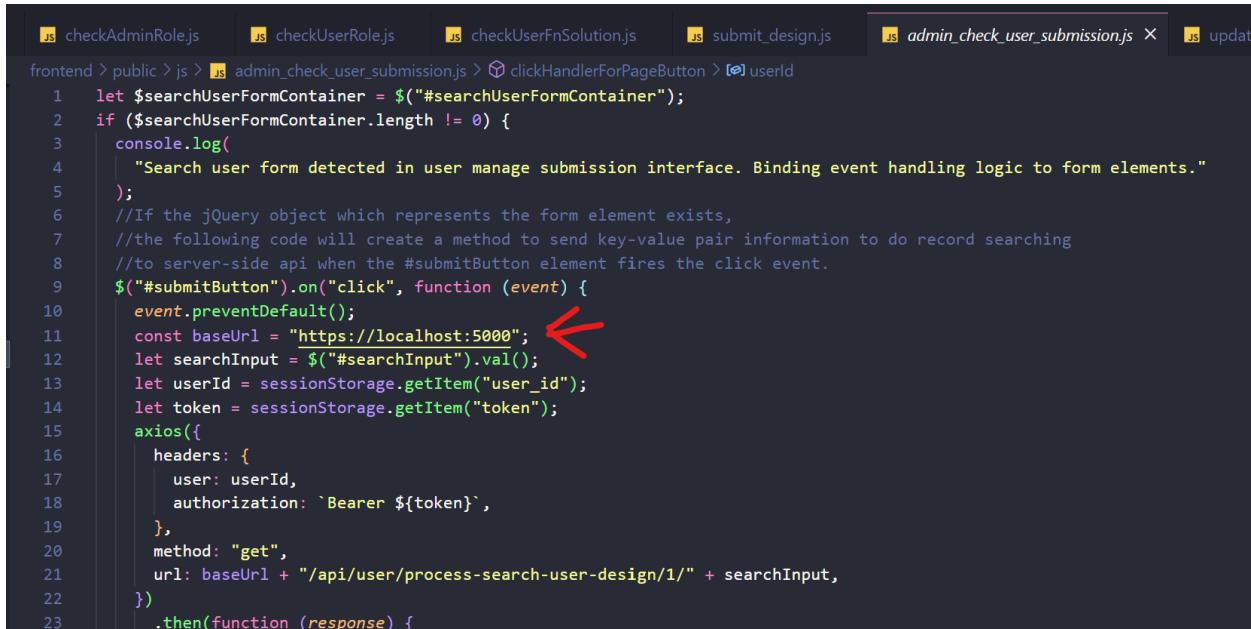
We will now add the following code at the bottom of the index.js file to secure the frontend.

```
44  // Create an HTTPS server with the 'credentials' for a secure connection
45  const server = https.createServer(credentials, app);
46  server.listen(port, hostname, function () {
47    console.log(`Server hosted at https://\$ {hostname} : \$ {port}`);
48  });
49
```

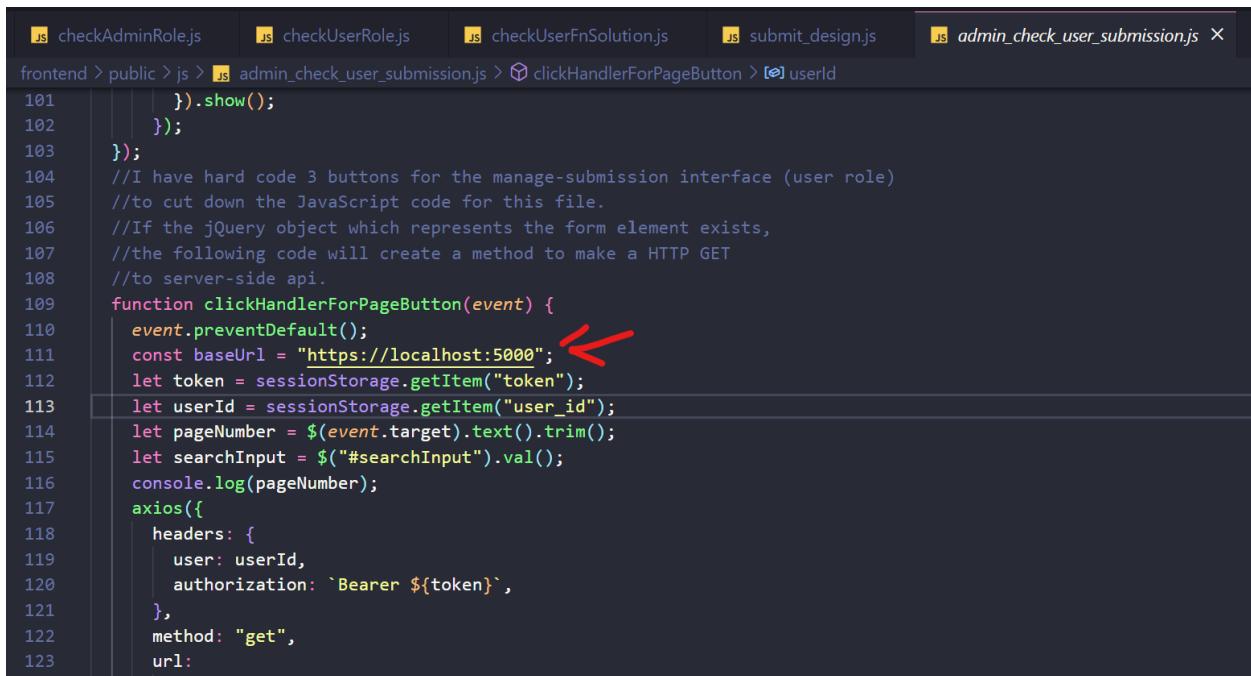
Step 12:

We will now need to change all request using http to https in the js folder in the frontend folder. Located under /frontend/public/js/.

As shown below for admin_check_user_submission.js we will change all the request to https. Do the same for the other js files using http request under /frontend/public/js/.



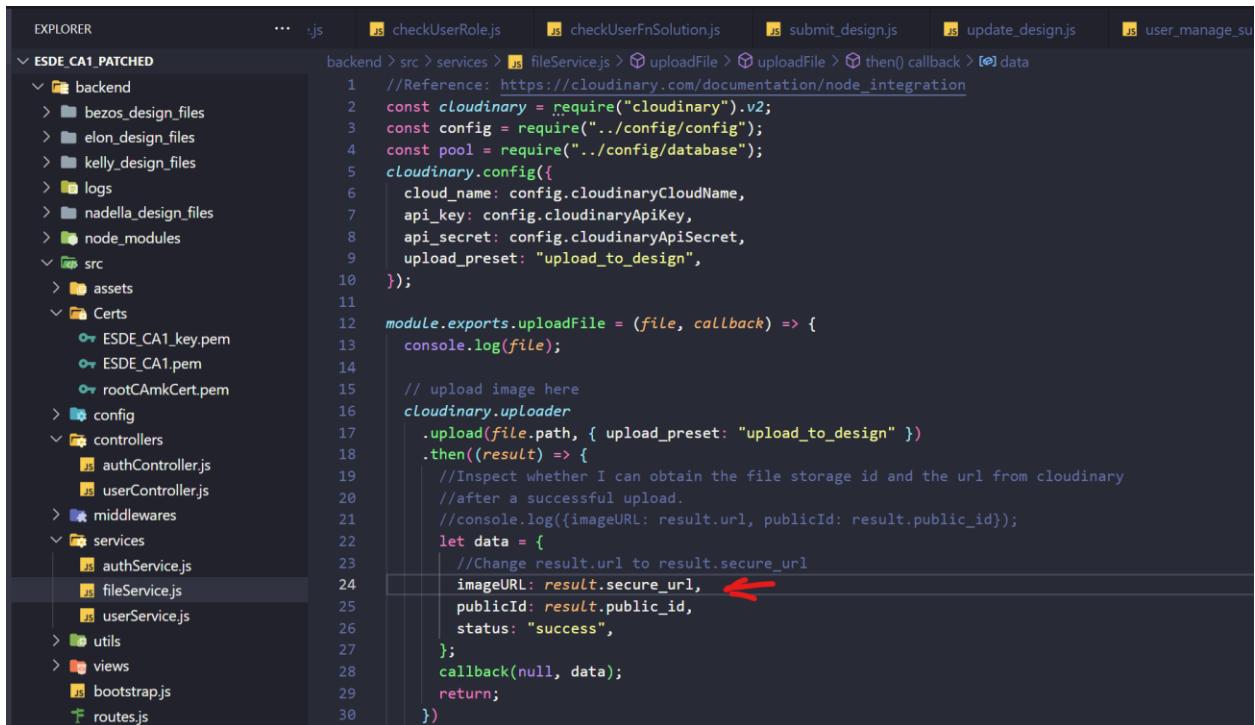
```
frontend > public > js > admin_check_user_submission.js > clickHandlerForPageButton > [userId]
1  let $searchUserFormContainer = $("#searchUserFormContainer");
2  if ($searchUserFormContainer.length != 0) {
3    console.log(
4      "Search user form detected in user manage submission interface. Binding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



```
frontend > public > js > admin_check_user_submission.js > clickHandlerForPageButton > [userId]
101   | }).show();
102   });
103   });
104   //I have hard code 3 buttons for the manage-submission interface (user role)
105   //to cut down the JavaScript code for this file.
106   //If the jQuery object which represents the form element exists,
107   //the following code will create a method to make a HTTP GET
108   //to server-side api.
109   function clickHandlerForPageButton(event) {
110     event.preventDefault();
111     const baseUrl = "https://localhost:5000"; ←
112     let token = sessionStorage.getItem("token");
113     let userId = sessionStorage.getItem("user_id");
114     let pageNumber = $(event.target).text().trim();
115     let searchInput = $("#searchInput").val();
116     console.log(pageNumber);
117     axios({
118       headers: {
119         user: userId,
120         authorization: `Bearer ${token}`,
121       },
122       method: "get",
123       url:
```

Step 13:

Let us go to fileService.js located under /backend/src/services/fileService.js and change the image link from result.url to result.secure_url to use https for our image upload too as shown below.

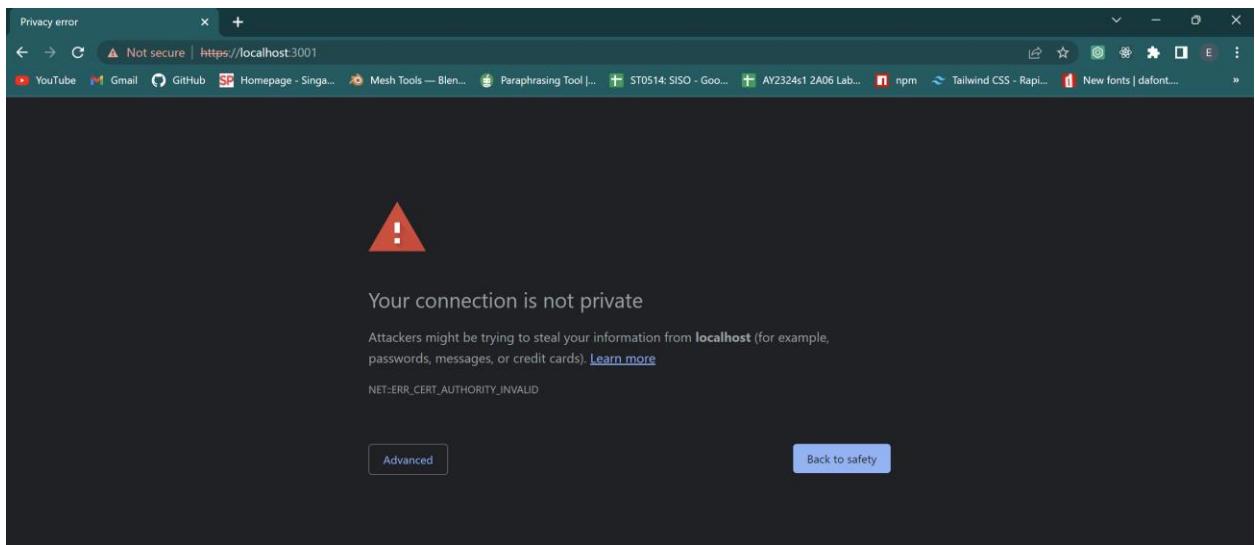


```
EXPLORER          ... .js    checkUserRole.js    checkUserFnSolution.js    submit_design.js    update_design.js    user_manage_su
ESDE CA1 PATCHED
  backend
    bezos_design_files
    elon_design_files
    kelly_design_files
    logs
    nadella_design_files
    node_modules
  src
    assets
  Certs
    ESDE_CA1_key.pem
    ESDE_CA1.pem
    rootCAmkCert.pem
  config
  controllers
    authController.js
    userController.js
  middlewares
  services
    authService.js
    fileService.js
    userService.js
  utils
  views
    bootstrap.js
  routes.js

1 //Reference: https://cloudinary.com/documentation/node_integration
2 const cloudinary = require("cloudinary").v2;
3 const config = require("../config/config");
4 const pool = require("../config/database");
5 cloudinary.config({
6   cloud_name: config.cloudinaryCloudName,
7   api_key: config.cloudinaryApiKey,
8   api_secret: config.cloudinaryApiSecret,
9   upload_preset: "upload_to_design",
10 });
11
12 module.exports.uploadFile = (file, callback) => {
13   console.log(file);
14
15   // upload image here
16   cloudinary.uploader
17     .upload(file.path, { upload_preset: "upload_to_design" })
18     .then(result) => {
19       //Inspect whether I can obtain the file storage id and the url from cloudinary
20       //after a successful upload.
21       //console.log({imageURL: result.url, publicId: result.public_id});
22       let data = {
23         //Change result.url to result.secure_url
24         imageURL: result.secure_url, ←
25         publicId: result.public_id,
26         status: "success",
27       };
28       callback(null, data);
29       return;
30     }
}
```

Step 14:

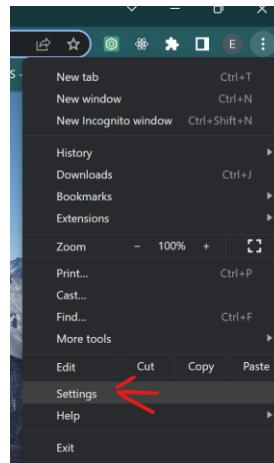
You will then see this problem when you type the https localhost link into the url.



Step 15:

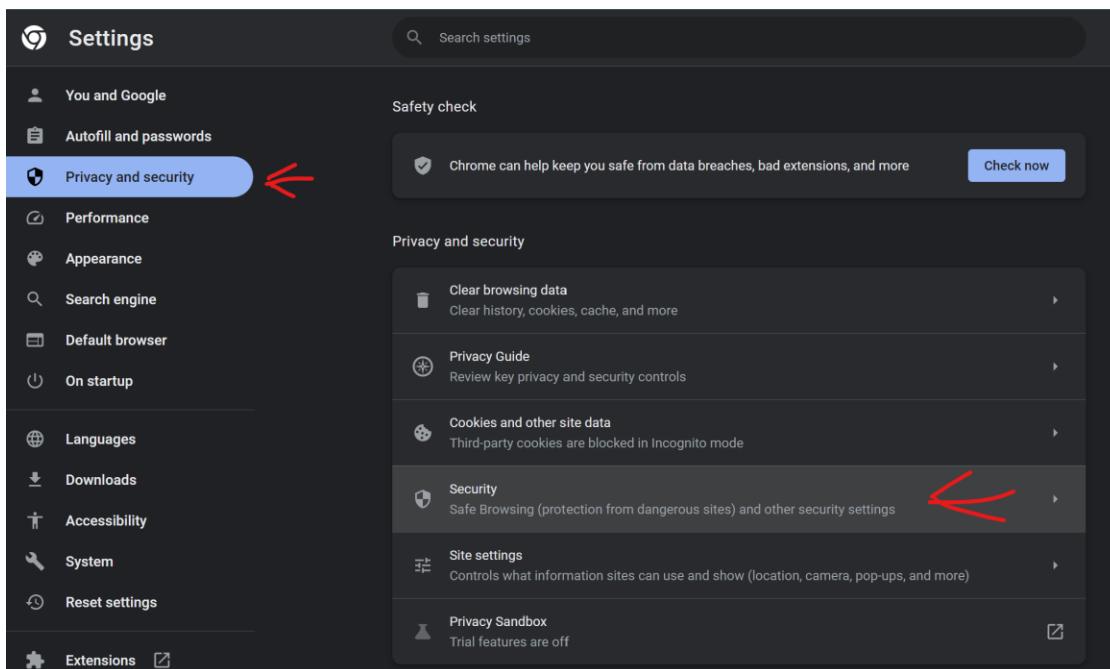
To solve this problem, we will import the rootCA.pem file into the trusted certs in google chrome.

First, go to settings after clicking on the 3 dot on the top right corner.



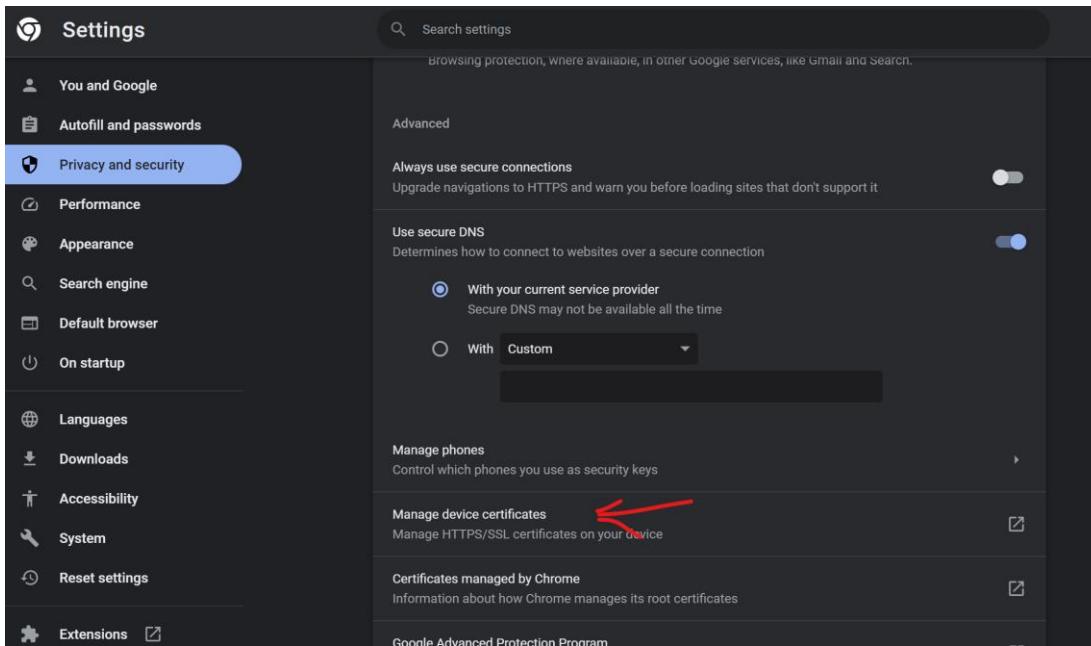
Step 16:

Go to Privacy and security and click on Security as shown below.



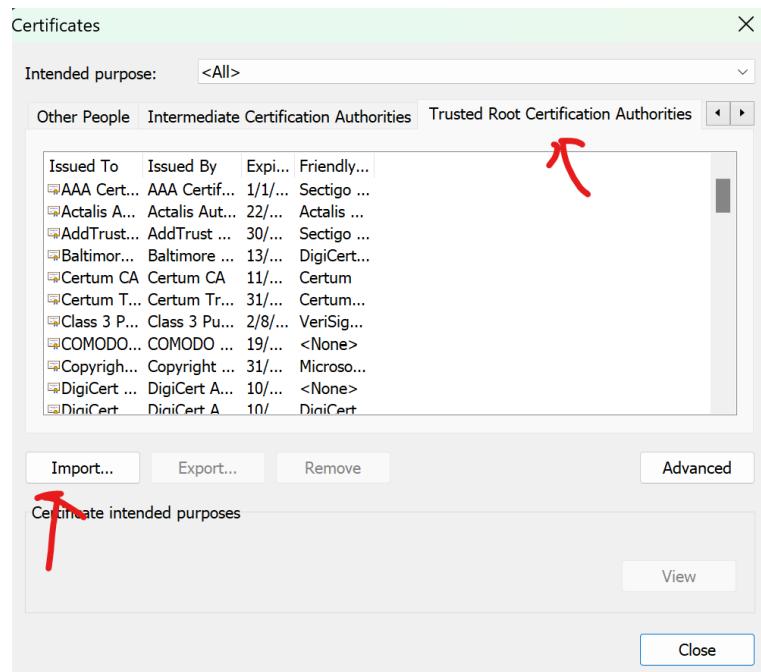
Step 17:

Scroll down and you will see a “Manage device certificates”. Click on it.



Step 18:

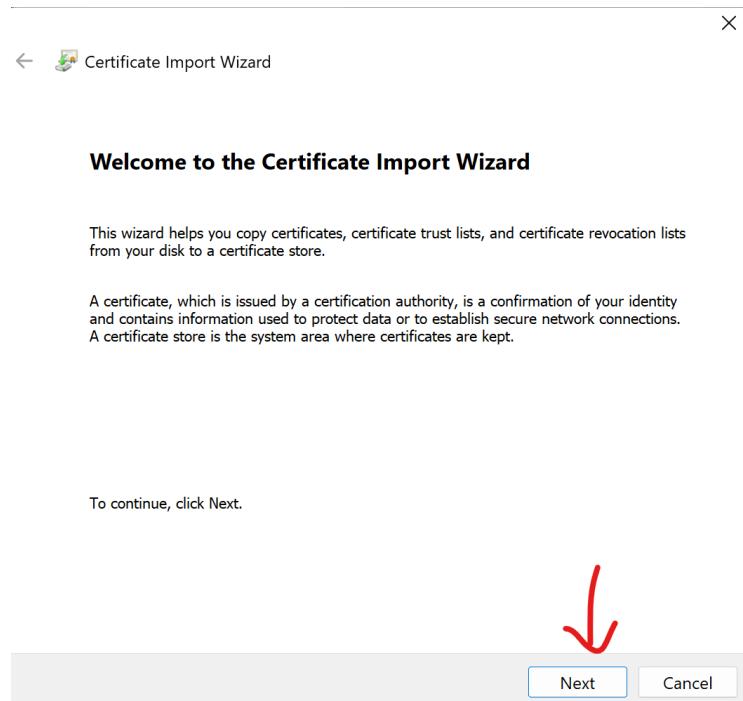
Next, you will see a pop-up go and click on Trusted Root Certification Authorities and import the rootCA.pem file that we got earlier.



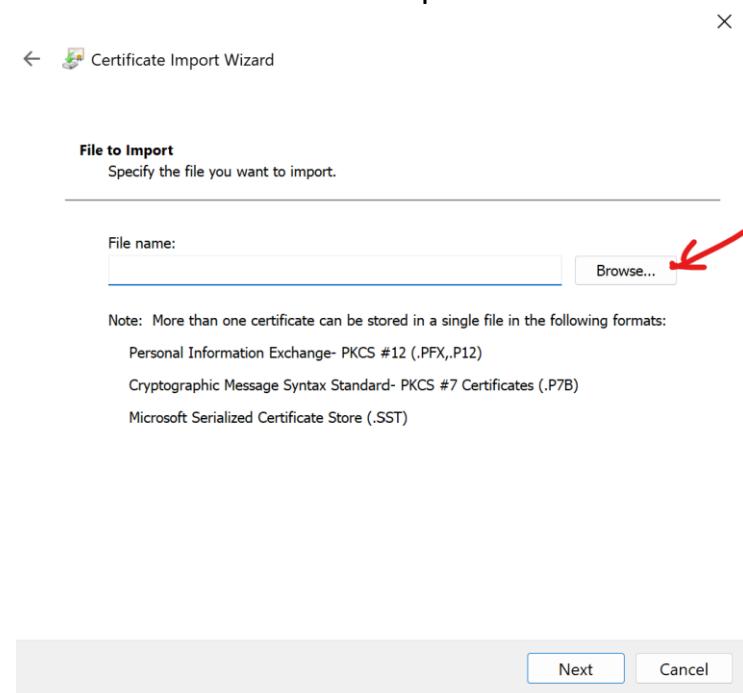
Step 19:

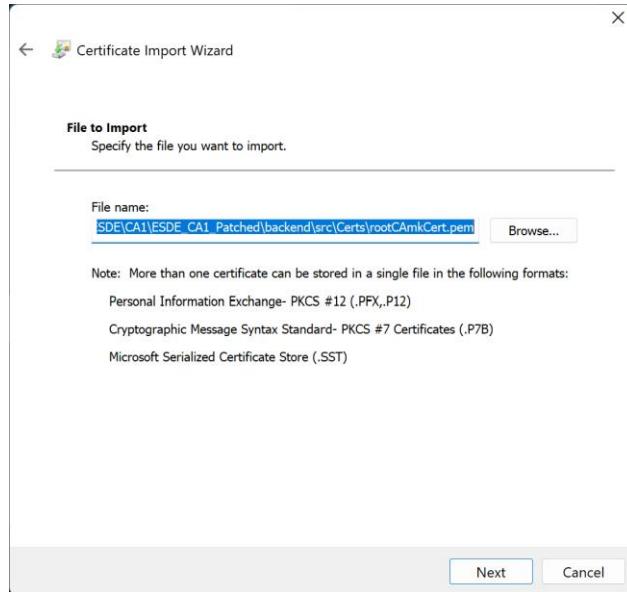
Follow the steps below to import the rootCA.pem file.

1. Click on next

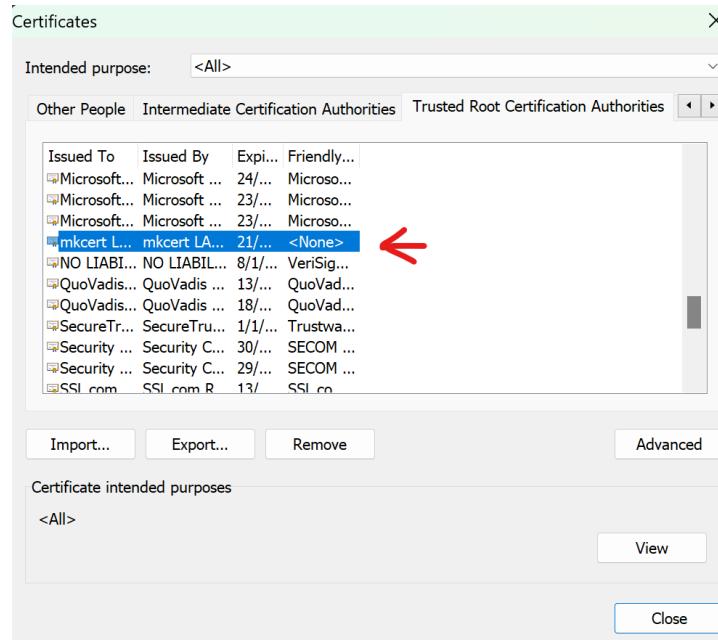


2. Click on Browse and look for the rootCA.pem file



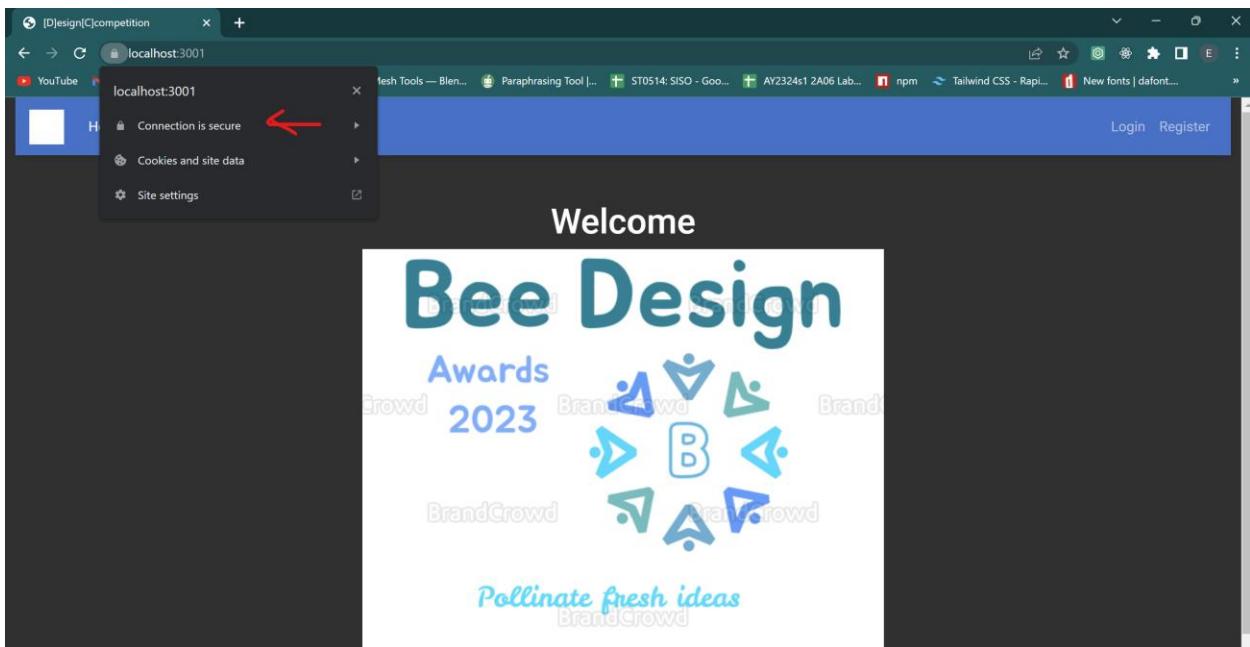


3. Click on Next, Yes and Finish and you have successfully imported the self-signed cert. You should see a mkcert under the Issued By column.



Step 20:

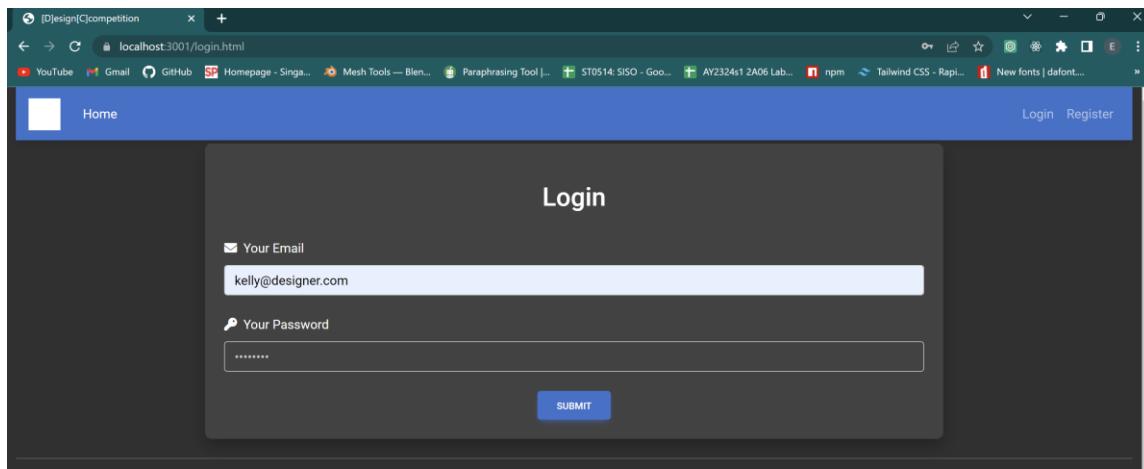
Now close the browser and reopen it and type the https localhost link again and the error should be removed. You should see a lock icon indicating that your site is now secure.



5.4 Evidence that the vulnerability has been rectified

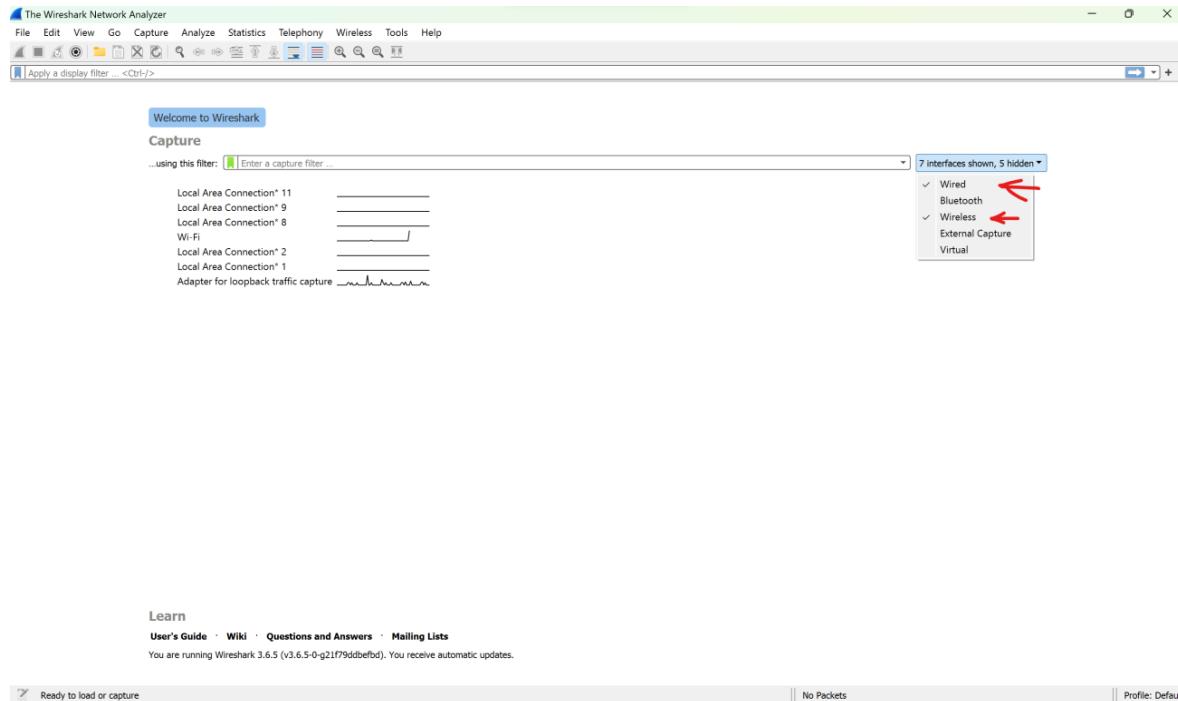
Step 1:

Now go into the login page and fill in Kelly's email and password **but do not press the submit button.**



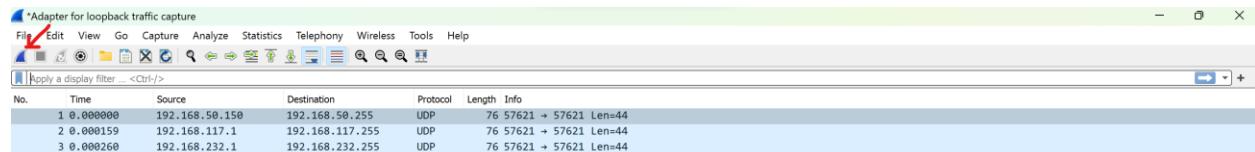
Step 2:

Head over to Wireshark and we will test if we are still capturing traffic showing plain text data and make sure that you have checked Wired and Wireless.



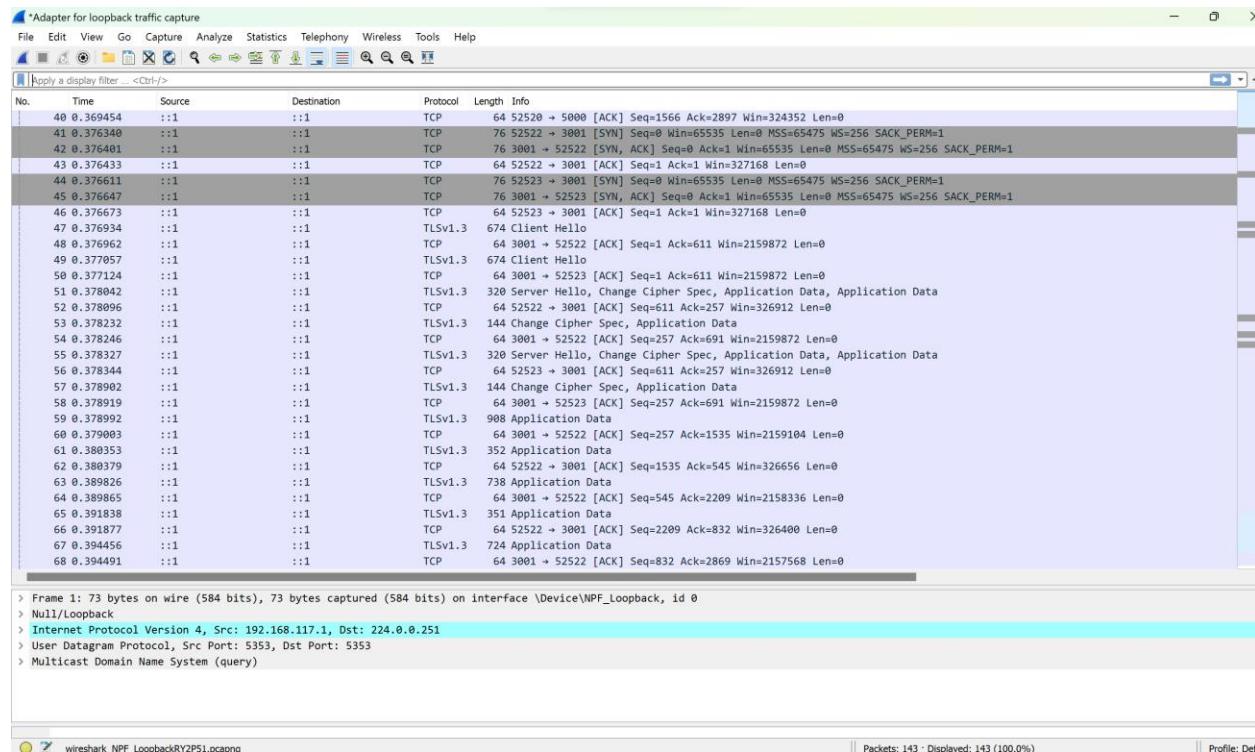
Step 3:

Next click on Adapter for loopback traffic capture and press the shark fin looking icon on the top left to begin capturing the traffic.



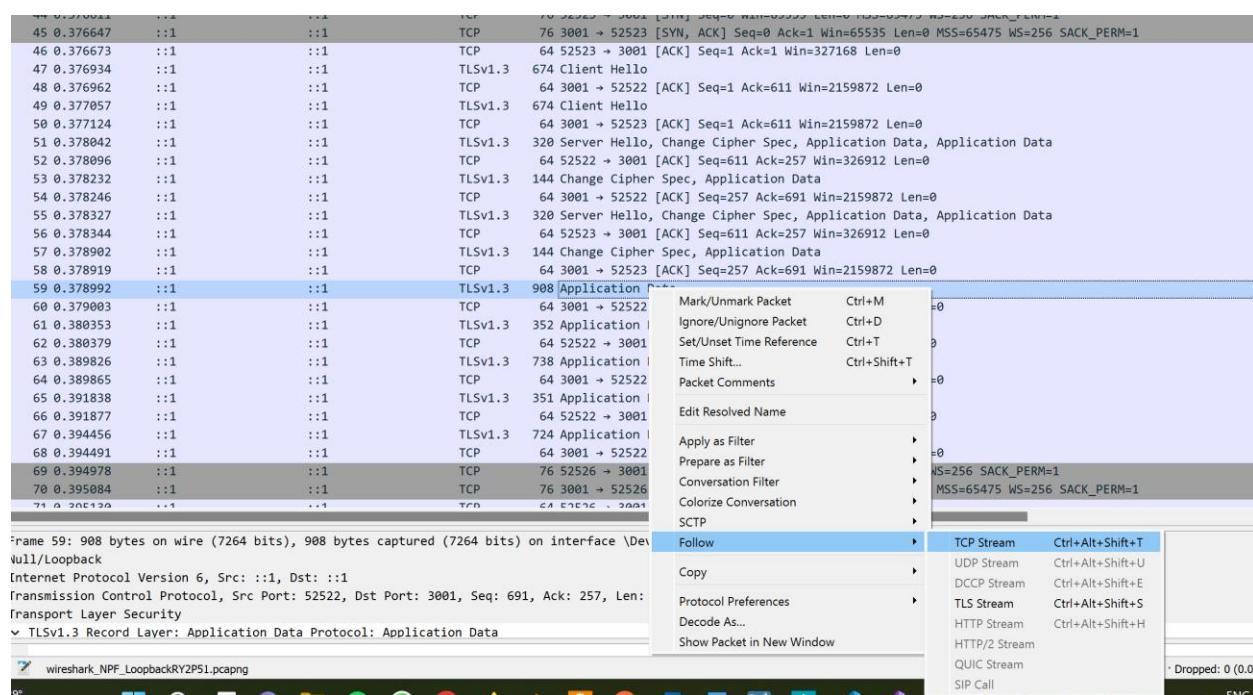
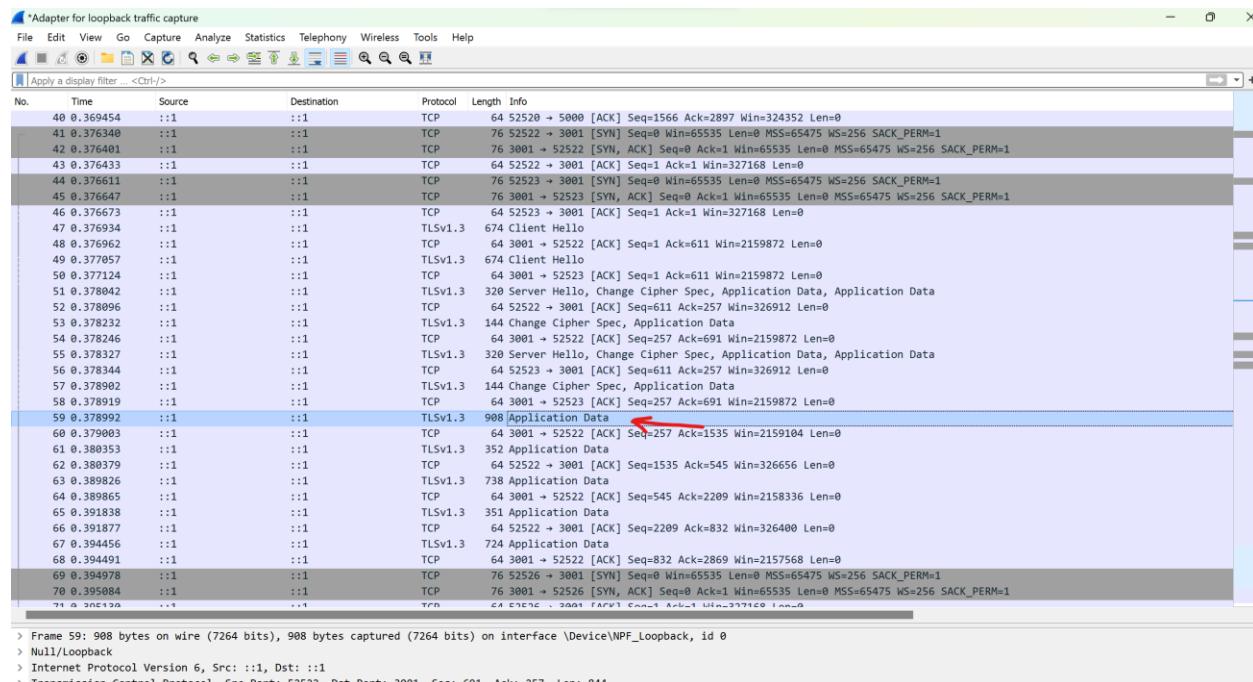
Step 4:

Click on the submit button to capture the traffic for our login request. And stop the Wireshark capture by clicking on the red square shaped button. You should not be able to see any request anymore compared to when our site is insecure.



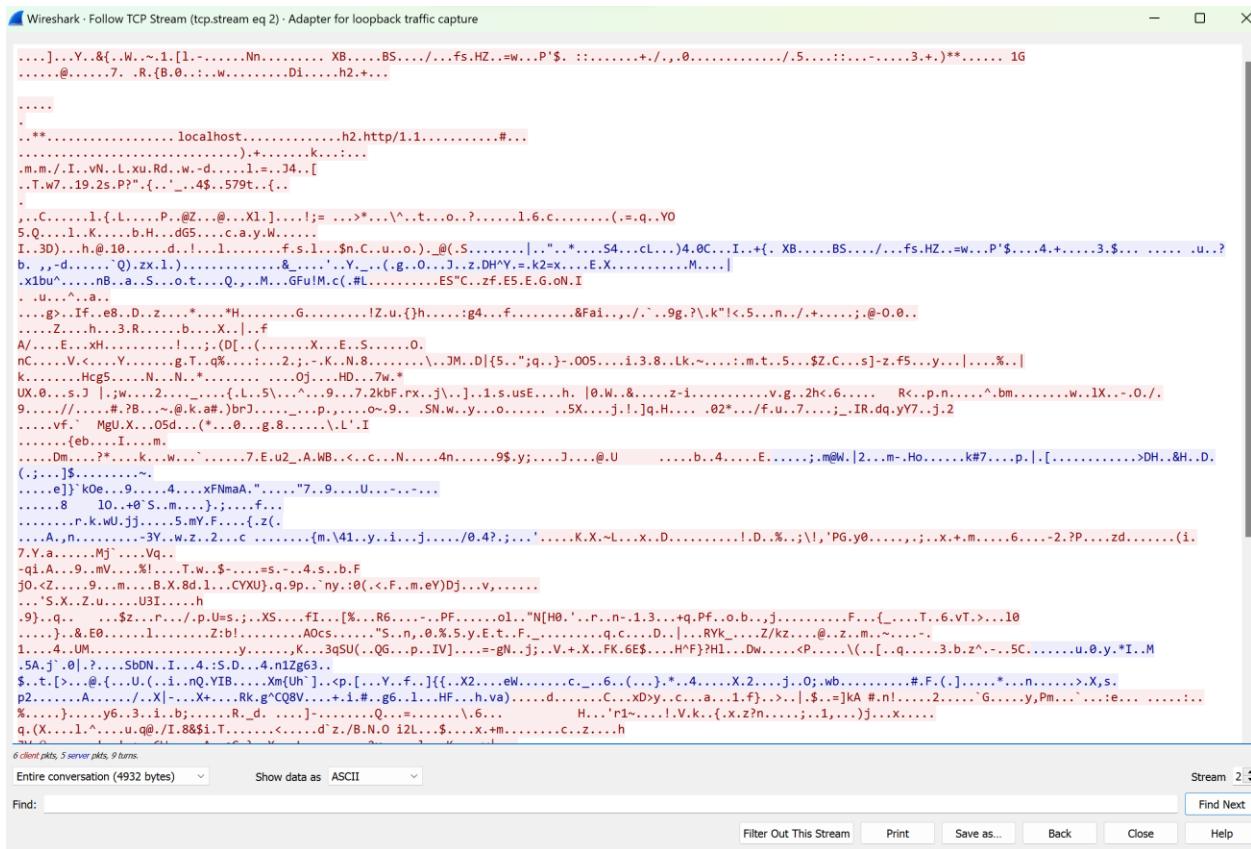
Step 5:

You will realize that our data is protected by a TLS protocol which helps us encrypt the data that is in transit as shown below. Click on an Application Data as shown below and right click to go into Follow > TCP Stream.



Step 6:

You should be able to see encrypted data as shown below. This shows that our cert has helped us to encrypt all the data in transit and we have successfully rectified this sensitive data exposure vulnerability.



The screenshot shows a Wireshark capture of an encrypted TCP stream (tcp.stream eq 2) on an Adapter for loopback traffic capture. The packet list view displays several encrypted frames, with the content pane showing redacted data. The status bar at the bottom indicates "6 client pkts, 5 server pkts, 9 items". The interface includes standard Wireshark controls like "Entire conversation (4932 bytes)", "Show data as ASCII", "Stream 2", "Find", "Find Next", and "Help".

```
....]..Y..&{..W..~.1.[.~.....Nn..... XB.....BS.....fs.HZ..=w...P'$.. :.....+/.0...../.5.....3.+.)**..... 1G
.....@.....7..R.{B.0..:.w.....Di.....h2.+.+
.....
....*.....localhost.....h2.http/1.1.....#...
.....
....*.localhost.....)+.....k.....
.m.m./.I..vN..L.xu.Rd..w.-d....l..,.34..[
..T.w7..19.25.P?"..45..579c..{..
.
..C.....1.{.L....P..@Z...@...X1.]....!;= ...>...^\t...o.,?.....1.6.c.....(.=q..Y0
5.Q.....1..K.....b.H.....dGS.....c.a.y.W...
I..3D)..h@.10....d..l....f.s.l...$n.C.u.o.o)...@(.S.....|..".....S4...cL...)4.0C...I..+(. XB.....BS.....fs.HZ..=w...P'$....4....3.$.....u.?
b...,,d.....`Q).zx.l...).....&..'.Y..(.g..0...J..z.DH.Y..=k2=x...E.X.....M...
.xbu^....nB..a...S...o.t...Q...M...6Fu!M.c(..#.....ES*C..zf.E5.E.G.O.I
..u...^..a...
....g>..If..e8..D..z...*..*H.....G.....|Z.u.{}h.....g4..f.....&Fa1.../.~.9g.?\\k"!<.5..n.../.+....;.@-0..0..
....Z...h...3.R...b...X..|.f
A/....E...xH.....!..;..(D|..(.X..E..S.....0.
nC.....V.<....Y.....g.T..q%.....2.;-K..N.8.....\..JM..D[5..";q..]-005..i..3.8..Lk..~...:m..t..5..$Z.C..s]-z.f5..y...|....%..|
k.....Hg5.....N..N..*.....0j...HD..7w.*
UX..0..s.J |.;w....2....({.L..5..^..9....9..7..2kbF.rx..j\...].1.s.usE.....h..|0..W..&....z-i.....v.g..2h<.6..... R<..p.n.....^..bm.....w..1X..~..O.?
9....//.:#.PB..~..@.k.#..)brJ.....p...o...o..9...SN.w..y..o.....5X..j..!..q.H..=.02*..f.u..7....;_IR.dq.yY7..j.2
....vf..MgU.X...05d...(*...g.B...\\'L'.I
....{eb...I...m
....Dm...?*..k..w...`.....7.E.u2..A.WB...c..N.....4n.....9$.y;....J....@.U .....b..4....E.....;..m@W..[2...m..Ho.....k#7....p.|.[.....>DH..&H..D
(.j...)]$....~
....e]}KO..9....4...xFNmaA....."...."7..9...U..~-..-+-
....8 ..10..+0'S..m...),..,f...
....r.k.wU.jj...5.mY.F...{.z.
....A..n.....3Y..w.z..2..c.....{m..41..y..i..j.....0.4?;..'.....K.X..~..x..D.....|..D..%..;\\l..'PG.y0.....;..x.+..m.....6...-2..?P...zd.....(i.
7.Y.a.....Mj'....V.q..
-q1.A...9..mV...%!.T.w..$-...=s...4..s..b.F
j0..<....9..m...B.X.8d.l...CYXU).q..9p..ny.:0(<.F..m.eY)Dj..v,.....
....S.X..Z.u..._U3I...h
....9..q...$.z..r.../..p.U=s..;..X5...fI...[%.R6.....PF.....o1..N[H0.'..r..n..-1..3...+q.Pf..o.b..,j.....F..{.....T..6.VT.>..10
....)&..E0.....1.....Z:b!.....AOcs.....S..n..0..%..5..y.E.t..F.....q..c..D..|..RVk.....Z/kz...@..z..m..~..-;
1....4..UM.....y.....K..3gSU(..Q6..p..IV)..=..gN..j..V..+..X..FK..6E$....H'F)?H1..DW.....<..[..q..3..b..z^..-..5C.....u..@..y..*I..M
..5A..j'..0..?..SBDN..I..4..5..D..4..n12g63..
$..t..[...@..{..U..{..i..NQ..YB.....Xn{Uh}..cp..[...Y..f..]{...X2..eW.....c..6..{..}..*..4..X..2..j..0..wb.....#.F..{..}..*..n..>..X..s.
p2.....A...../..X|-..X..Rk..g..CQBV..+..1..#.g6..1..HF..h.v)a.....d.....C..xDy..c..a..1..f)..>..|..$.=-)kA..#..n!.....2....`G..y..Pm...`..:e.....
%..}...y6..3..i..b;...R..d..)-.....Q..=.....\..6...
....H..'r1~...l..V..k..{..x..z?n..};..1..)j..x..-.
q..(X..1.^..u..q@..I..88$..T.....<..d..z..B..N..0..i2L..$.x..+..m.....c..z....h
```

6. Insufficient Logging & Monitoring

6.1 Definition

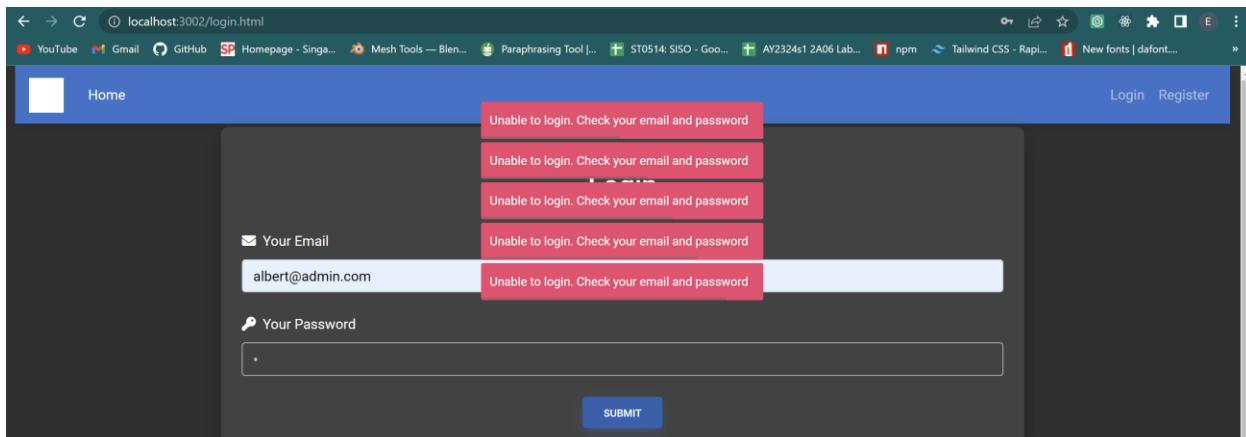
Insufficient Logging & Monitoring is a security weakness where an application or system lacks adequate mechanisms to generate, capture, retain, and analyze log events and security-related information. It occurs when an organization fails to implement robust logging and monitoring practices, which can hinder their ability to detect, investigate, and respond to security incidents effectively.

This can manifest in several ways like Lack of Event Logging, Inadequate Log Retention, Insensitive Log Data, and Lack of Real-Time Monitoring.

6.2 How to exploit vulnerability

Step 1:

Go into the vulnerable website and try to login in with albert's email but for the password let's try to input incorrect ones to imitate an actual brute forcing scenario.

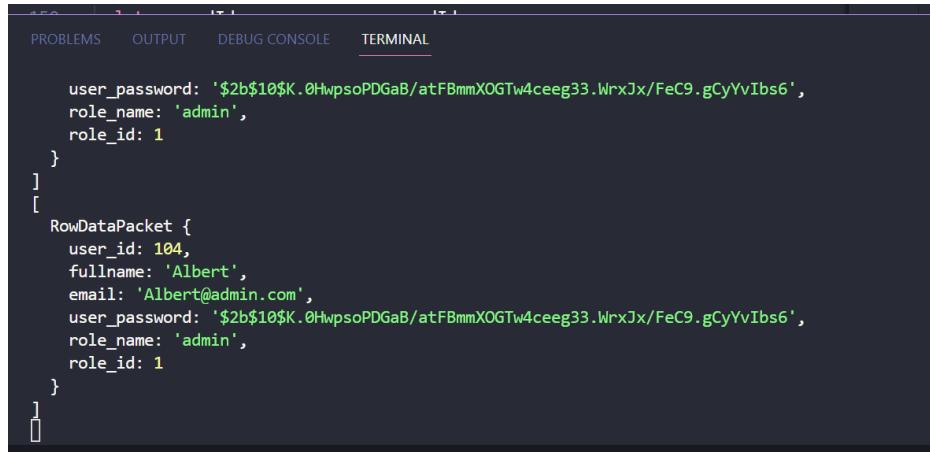


The screenshot shows a web browser window with the URL `localhost:3002/login.html`. The page has a dark theme with a blue header bar containing the word "Home". On the right side of the header are "Login" and "Register" buttons. The main content area contains a form with two fields: "Your Email" and "Your Password". The "Your Email" field is populated with "albert@admin.com". The "Your Password" field is empty. Below the form, there is a "SUBMIT" button. Above the form, there are five red error messages stacked vertically, all reading "Unable to login. Check your email and password".

- Public can register as a user
- The database has been seeded with user test data.
- user:kelly@designer.com password:password
normal user role
- user:albert@admin.com password:password
admin user role
- Inspect the user table for more information on all the seeded user test records
- Use the SQL script `prepare_database_ca1.sql` in the MySQL workbench to prepare a new database and necessary tables.

Step 2:

You will see that the only place that records the login process is only through the terminal as shown below. It is also not stating whether the login is successful or not and it also isn't being logged anywhere.



A screenshot of a terminal window from a code editor. The tab bar at the top shows 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', and 'TERMINAL', with 'TERMINAL' being the active tab. The terminal content displays a JSON array of two objects. The first object is a 'RowDataPacket' with fields: user_id: 103, fullname: 'Albert', email: 'Albert@admin.com', user_password: '\$2b\$10\$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg33.WrxJx/FeC9.gCyYvIbs6', role_name: 'admin', and role_id: 1. The second object is another 'RowDataPacket' with identical values. Both objects have a closing brace '}' at the end of their respective lines.

```
user_password: '$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg33.WrxJx/FeC9.gCyYvIbs6',
role_name: 'admin',
role_id: 1
]
[
  RowDataPacket {
    user_id: 104,
    fullname: 'Albert',
    email: 'Albert@admin.com',
    user_password: '$2b$10$K.0HwpsoPDGaB/atFBmmXOGTw4ceeg33.WrxJx/FeC9.gCyYvIbs6',
    role_name: 'admin',
    role_id: 1
  }
]
```

6.3 How to rectify vulnerability

Step 1:

To rectify the logging of the error etc. We will first import Winston and express-useragent as we will need to use this libraries to log the requests and errors.

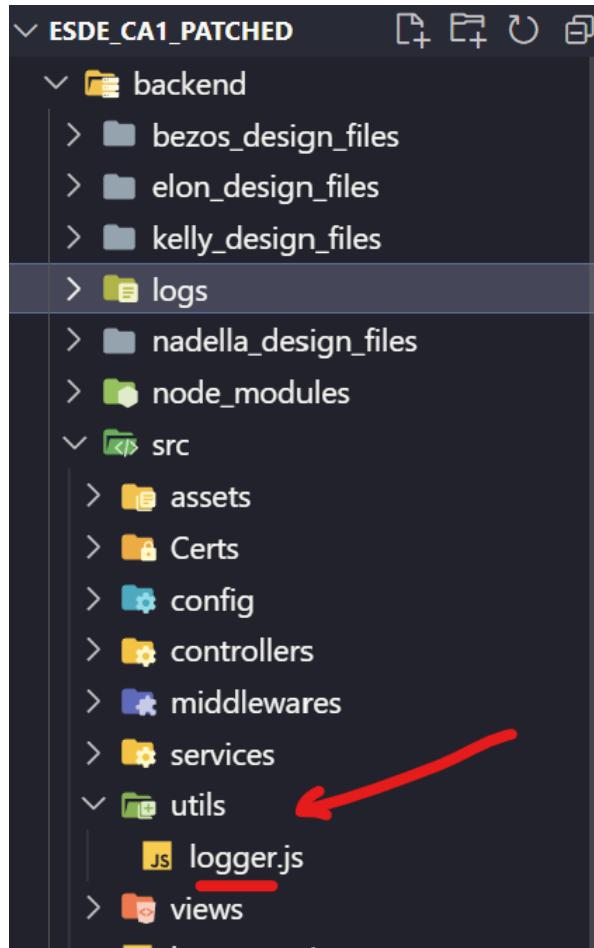
Copy the command and paste it into your backend terminal directory.

```
npm install winston express-useragent
```

Step 2:

Let's create a utils folder and create a logger.js file to set up our winston logger.

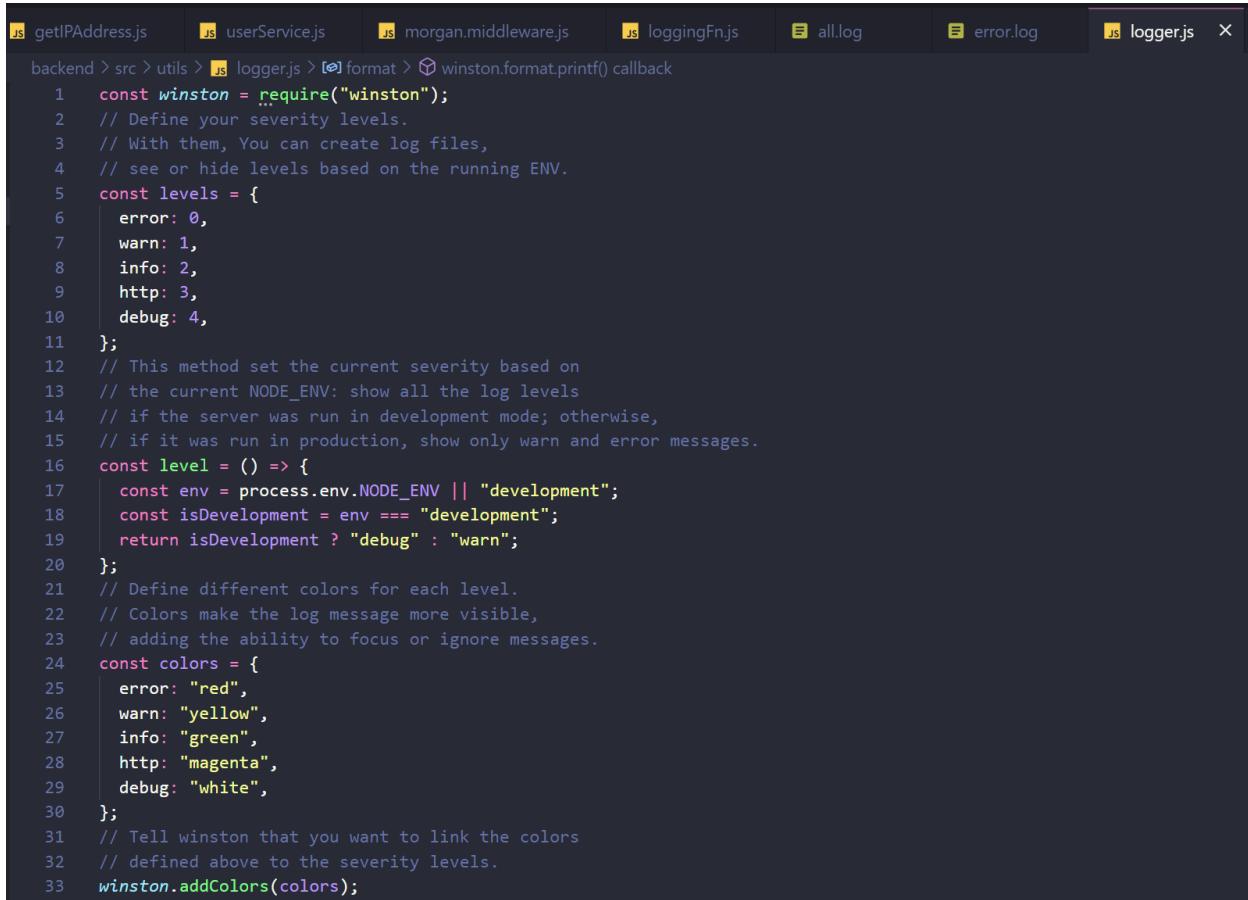
Under /backend/src/



Step 3:

Add the following codes to set up the winston logger.

Part 1 of logger.js:



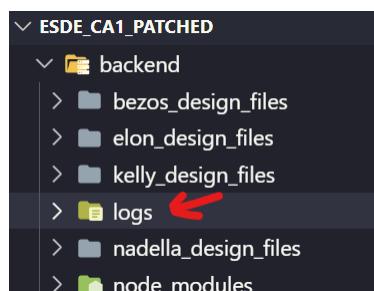
```
backend > src > utils > logger.js > [e] format > ⌂ winston.format.printf() callback
1  const winston = require("winston");
2  // Define your severity levels.
3  // With them, You can create log files,
4  // see or hide levels based on the running ENV.
5  const levels = {
6    error: 0,
7    warn: 1,
8    info: 2,
9    http: 3,
10   debug: 4,
11 };
12  // This method set the current severity based on
13  // the current NODE_ENV: show all the log levels
14  // if the server was run in development mode; otherwise,
15  // if it was run in production, show only warn and error messages.
16  const level = () => {
17    const env = process.env.NODE_ENV || "development";
18    const isDevelopment = env === "development";
19    return isDevelopment ? "debug" : "warn";
20  };
21  // Define different colors for each level.
22  // Colors make the log message more visible,
23  // adding the ability to focus or ignore messages.
24  const colors = {
25    error: "red",
26    warn: "yellow",
27    info: "green",
28    http: "magenta",
29    debug: "white",
30  };
31  // Tell winston that you want to link the colors
32  // defined above to the severity levels.
33  winston.addColors(colors);
```

Part 2 of logger.js:

```
getIPAddress.js    userService.js    morgan.middleware.js    loggingFn.js    all.log    error.log    logger.js
backend > src > utils > logger.js > [o] format > ⚡ winston.format.printf() callback
32 // defined above to the severity levels.
33 winston.addColors(colors);
34 // Choose the aspect of your log customizing the log format.
35 const format = winston.format.combine(
36     // Add the message timestamp with the preferred format
37     winston.format.timestamp({ format: "DD-MM-YYYY HH:mm:ss:ms" }),
38     // Tell Winston that the logs must be colored
39     winston.format.colorize({ all: true }),
40     // Define the format of the message showing the timestamp, the level and the message
41     winston.format.printf(
42         (info) => `${info.timestamp} ${info.level}: ${info.message}`
43     )
44 );
45 // Define which transports the logger must use to print out messages.
46 // In this example, we are using two different transports
47 const transports = [
48     // Allow the use the console to print the messages
49     // new winston.transports.Console(),
50     // Allow to print all the error level messages inside the error.log file
51     new winston.transports.File({
52         filename: "logs/error.log",
53         level: "error",
54     }),
55     // Allow to print all the error message inside the all.log file
56     // (also the error log that are also printed inside the error.log)
57     new winston.transports.File({ filename: "logs/all.log" }),
58 ];
59 // Create the logger instance that has to be exported
60 // and used to log messages.
61 const logger = winston.createLogger({
62     level: level(),
63     levels,
64     format,
65     transports,
66 });
67 module.exports = logger;
```

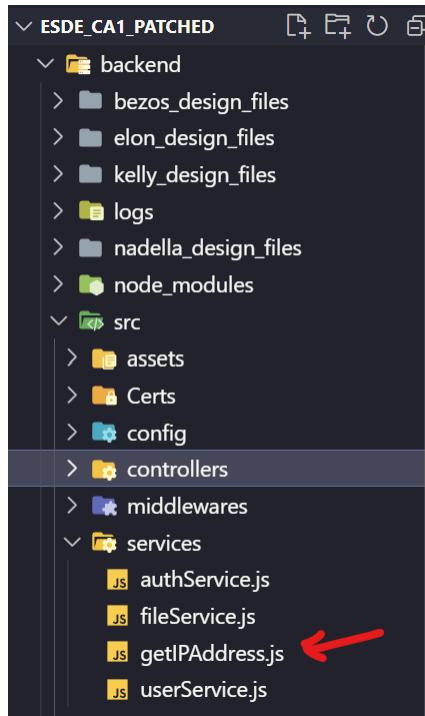
Step 4:

Create a logs folder under /backend/ we will use this to store our logs.



Step 5:

Next, create a file called getIPAddress.js the services folder under /backend/src/services.



Step 6:

In the file add the following code so that we will be able to get the IP Address of the user.

```
admin_manage_user.js      JS admin_update_user.js      JS authController.js      JS authService.js      JS fileService.js      JS getIPAddress.js X
backend > src > services > JS getIPAddress.js > getIPAddress > getIPAddress > catch() callback
1  const axios = require("axios");
2
3  exports.getIPAddress = () => {
4    return axios
5      .get("https://api.ipify.org")
6      .then((response) => {
7        const ip = response.data;
8        return ip;
9      })
10     .catch((error) => {
11       console.log(error);
12       return null;
13     });
14   };
15 }
```

Step 7:

Next, head over to validateFn.js and import the following or anywhere where you would want to log any data.



```
backend > src > middlewares > validateFn.js > validateFn > sanitizeResult
1 var validator = require("validator");
2 var path = require("path");
3 var fs = require("fs");
4 var logger = require("../utils/logger"); ←
5 const { getIPAddress } = require("../services/getIPAddress"); ←
6
```

Step 8:

We will create a validateEmail and add a log to it to capture the error in the login page if any hacker tries to brute force.



```
52 validateEmail: function (req, res, next) {
53   var email = req.body.email;
54   if (validator.isEmail(email)) {
55     next();
56   } else {
57     let message = "Unable to Login, Please enter a valid Email";
58     getIPAddress()
59       .then((ip) => {
60         logger.error({
61           userAgent: req.useragent.source,
62           method: req.method,
63           route: req.originalUrl,
64           email: req.body.email,
65           ip: ip,
66           error: message,
67           status: 400,
68         });
69       })
70       .catch((error) => {
71         console.log(error);
72       });
73     res.status(400).send({ message: message });
74   }
75 },
```

Step 9:

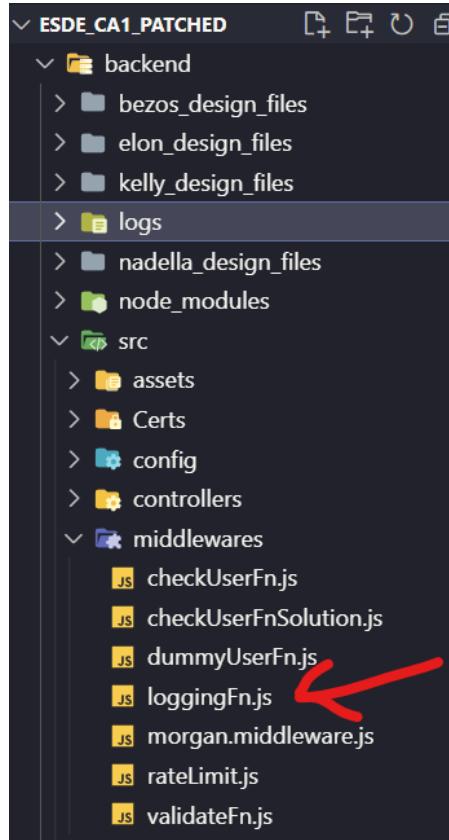
We will also add 1 logger to the bcrypt password at processLogin in the authController.js under /backend/src/controllers/authController.js.

We will add the logger to the else statement after checking if the bcrypt is not successful.

```
82         }
83     if (bcrypt.compareSync(password, results[0].user_password) == true) {
84         let data = {
85             user_id: results[0].user_id,
86             role_name: results[0].role_name,
87             token: jwt.sign(
88                 { id: results[0].user_id, role: results[0].role_name },
89                 config.JWTKey,
90                 {
91                     expiresIn: 86400, //Expires in 24 hrs
92                 }
93             ),
94         }; //End of data variable setup
95
96         return res.status(200).json(data);
97     } else {
98         getIPAddress()
99             .then((ip) => {
100                 logger.error({
101                     userAgent: req.useragent.source,
102                     method: req.method,
103                     route: req.originalUrl,
104                     email: req.body.email,
105                     ip: ip,
106                     error: "Password does not match",
107                     status: 401,
108                 });
109             })
110             .catch((error) => {
111                 console.log(error);
112             });
113         return res.status(500).json({ message: error });
114     } //End of password comparison with the retrieved decoded password.
```

Step 10:

We will now create 1 more file called logginFn.js under /backend/src/middlewares where we will use it to log all requests in the routes.js.



Step 11:

Add the following code into the loggingFn.js file under /backend/src/middlewares.

```
ler.js      JS admin_check_user_submission.js    JS admin_manage_user.js    JS admin_update_user.js    JS au
backend > src > middlewares > JS loggingFn.js > logDataFn > then() callback
  1  const logger = require("../utils/logger");
  2  const { getIPAddress } = require("../services/getIPAddress");
  3
  4  const logDataFn = {
  5    logLoginData: (req, res, next) => {
  6      getIPAddress()
  7        .then((ip) => {
  8          logger.info({
  9            userAgent: req.useragent.source,
10            method: req.method,
11            route: req.originalUrl,
12            email: req.body.email,
13            ip: ip,
14            });
15        })
16        .catch((error) => {
17          console.log(error);
18        });
19      return next();
20    },
21    logData: (req, res, next) => {
22      getIPAddress()
23        .then((ip) => {
24          logger.info({
25            userAgent: req.useragent.source,
26            method: req.method,
27            route: req.originalUrl,
28            ip: ip,
29            userId: req.body.userId,
30            });
31        })
32        .catch((error) => {
33          console.log(error);
34        });
35      return next();
36    },
37  };
38
39  module.exports = logDataFn;
40
```

Step 12:

Go to routes.js and we will import the loggingFn.js inside and use it on all the endpoints with a red arrow pointing to it.



```
// Import controllers
const authController = require("./controllers/authController");
const userController = require("./controllers/userController");
const checkUserFn = require("./middlewares/checkUserFn");
const checkUserFnSolution = require("./middlewares/checkUserFnSolution");
const loggingFn = require("./middlewares/loggingFn"); // Red arrow points here
const rateLimitFn = require("./middlewares/rateLimit");
const validateFn = require("./middlewares/validateFn");

// Match URL's with controllers
exports.appRoute = (router) => {
  // LOGIN
  router.post(
    "/api/user/login",
    loggingFn.logLoginData, // Red arrow points here
    rateLimitFn.loginCountLimit,
    validateFn.validateEmail,
    authController.processLogin
  );
  // REGISTER
  router.post(
    "/api/user/register",
    loggingFn.logData, // Red arrow points here
    rateLimitFn.registerCountLimit,
    validateFn.validateRegister,
    authController.processRegister
  );
  // USER SEARCH SUBMISSION
  router.get(
    "/api/user/process-search-design/:pagenumber/:search?",
    loggingFn.logData, // Red arrow points here
    checkUserFnSolution.checkForValidUser,
    validateFn.validateUserSearchOwnSubmission,
    userController.processGetSubmissionData
  );
  // USER SUBMIT DESIGN
  router.post(
    "/api/user/process-submission",
    loggingFn.logData, // Red arrow points here
    checkUserFnSolution.checkForValidUser,
    validateFn.validateDesignTitleAndDesc,
    validateFn.validateFileType,
    rateLimitFn.designSubmissionCountLimit,
    userController.processDesignSubmission
  );
  // USER UPDATE DESIGN (PERSISTENT PHASE)
}
```

```
routes.js  X  checkUserFnSolution.js  validateFn.js  index.js frontend  JS

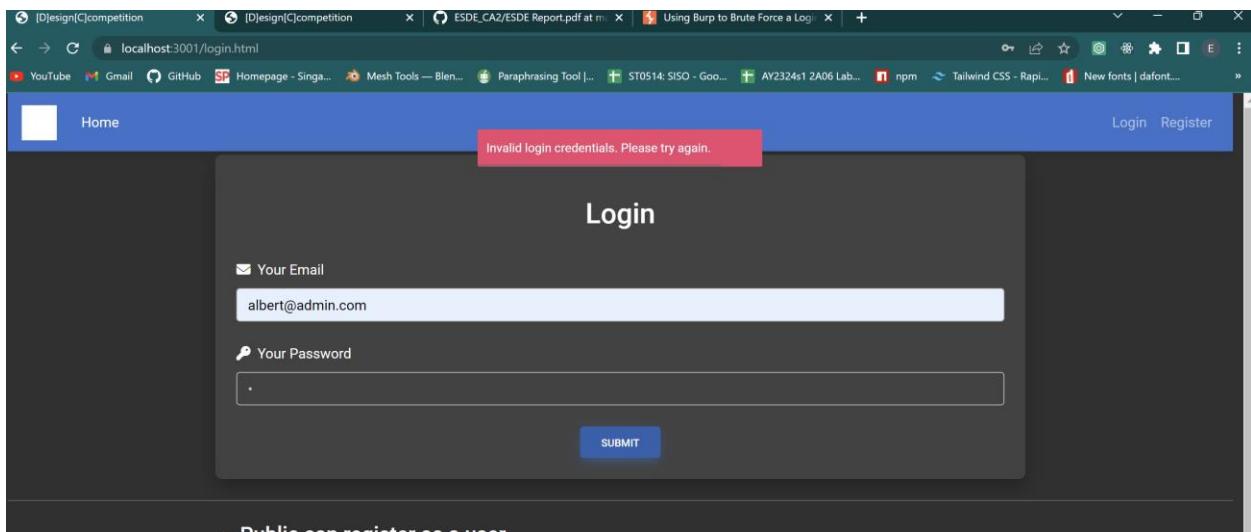
backend > src > routes.js > ...
52  );
53  // USER UPDATE DESIGN (UPDATE PHASE)
54  router.put(
55    "/api/user/design/",
56    loggingFn.logData, ↗
57    checkUserFnSolution.checkForValidUser,
58    validateFn.validateDesignTitleAndDesc,
59    userController.processUpdateOneDesign
60  );
61  // USER SEND INVITATION EMAIL
62  router.post(
63    "/api/user/processInvitation/",
64    loggingFn.logData, ↗
65    checkUserFnSolution.checkForValidUser,
66    validateFn.validateEmailRecipient,
67    userController.processSendInvitation
68  );
69  // USER VIEW PROFILE (RETRIEVE PHASE)
70  router.get(
71    "/api/user/:recordId",
72    checkUserFnSolution.checkForValidUser,
73    userController.processGetOneUserData
74  );
75
76  // ADMIN SEARCH USER
77  router.get(
78    "/api/user/process-search-user/:pagenumber/:search?",
79    loggingFn.logData, ↗
80    checkUserFnSolution.checkValidAdminRole,
81    userController.processGetUserData
82  );
83  // ADMIN SEARCH SUBMISSION BY EMAIL
84  router.get(
85    "/api/user/process-search-user-design/:pagenumber/:search?",
86    loggingFn.logData, ↗
87    checkUserFnSolution.checkValidAdminRole,
88    validateFn.validateSearchUserSubByEmail,
89    userController.processGetSubmissionsByEmail
90  );
91  // ADMIN CHANGE USER ROLE (UPDATE PHASE)
92  router.put(
93    "/api/user/",
94    loggingFn.logData, ↗
95    checkUserFnSolution.checkValidAdminRole,
96    userController.processUpdateOneUser
97  );
98
```

6.4 Evidence that the vulnerability has been rectified

Step 1:

Now, let's try if our logs are all working. We will go into the patched website now and go to the login page and fill in albert's email and we will input an incorrect password and submit it. For my case, I should see a message saying "Invalid login credentials. Please try again".

*Note: This depends on what message you return when the password does not match.



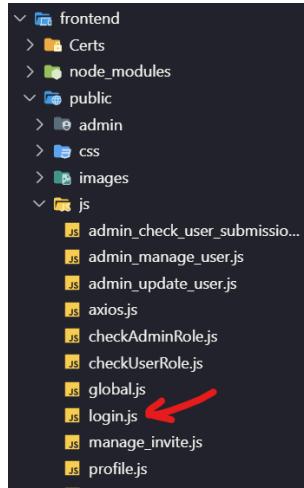
Continued Below...

For my case, I added the error message in my authController.js processLogin under the catch error and return it.

This is found under /backend/src/authController.

```
        }
        if (!bcrypt.compareSync(password, results[0].user_password) == true) {
            getIPAddress()
                .then((ip) => {
                    logger.error({
                        userAgent: req.useragent.source,
                        method: req.method,
                        route: req.originalUrl,
                        email: req.body.email,
                        ip: ip,
                        error: "Password does not match",
                        status: 401,
                    });
                })
                .catch((error) => {
                    console.log(error);
                });
            throw "Invalid login credentials. Please try again.";
        } else {
            let data = {
                user_id: results[0].user_id,
                role_name: results[0].role_name,
                token: jwt.sign(
                    { id: results[0].user_id, role: results[0].role_name },
                    config.JWTKey,
                    {
                        expiresIn: 86400, //Expires in 24 hrs
                    }
                ),
            }; //End of data variable setup
            return res.status(200).json(data);
        }
    }
} catch (error) {
    if (error == "Invalid login credentials. Please try again.") {
        let message = "Invalid login credentials. Please try again.";
        return res.status(500).json({ message: message });
    } else {
        console.log(error);
        let message = "Login has failed.";
        return res.status(500).json({ message: message });
    }
}
}; //End of processLogin
```

Head over to the frontend code and look for the file called login.js under /frontend/public/js/login.js.



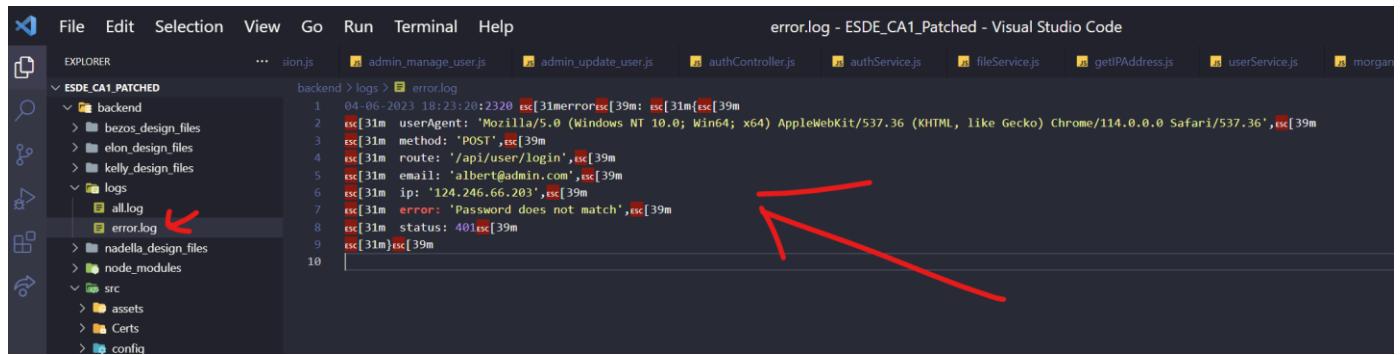
Change the static response "Unable to login. Check your email and password" to the following response message. This is so that we would be able to see our own custom error message that we return.

```
frontend > public > js > login.js > on("click") callback > catch() callback > text
11 const baseUrl = "https://localhost:3000",
12 let email = $("#emailInput").val();
13 let password = $("#passwordInput").val();
14 let webFormData = new FormData();
15 webFormData.append("email", email);
16 webFormData.append("password", password);
17 axios({
18   method: "post",
19   url: baseUrl + "/api/user/login",
20   data: webFormData,
21   headers: { "Content-Type": "multipart/form-data" },
22 })
23 .then(function (response) {
24   //Inspect the object structure of the response object.
25   //console.log('Inspecting the response object returned from the
26   //console.dir(response);
27   userData = response.data;
28   if (userData.role_name == "user") {
29     sessionStorage.setItem("token", userData.token);
30     sessionStorage.setItem("user_id", userData.user_id);
31     sessionStorage.setItem("role_name", userData.role_name);
32     window.location.replace("user/manage_submission.html");
33     return;
34   }
35   if (response.data.role_name == "admin") {
36     sessionStorage.setItem("token", userData.token);
37     sessionStorage.setItem("user_id", userData.user_id);
38     sessionStorage.setItem("role_name", userData.role_name);
39     window.location.replace("admin/manage_users.html");
40     return;
41   }
42 })
43 .catch(function (response) {
44   //Handle error
45   console.dir(response);
46   new Noty({
47     type: "error",
48     layout: "topCenter",
49     theme: "sunset",
50     timeout: "6000",
51     text: response.response.data.message, ↖
52   }).show();
53 });
54 );
55 } //End of checking for $loginFormContainer jQuery object
```

Step 2:

Head over to the logs folder under /backend and you should see 2 .log files, go to error.log and you should see the following error log as shown below.

You will be able to see the userAgent, method, route, email, IP address, error message and status. You will also be able to see the date and time where the following error occurred. This shows that we have successfully logged the error in the login page that would be sufficient for us.



```
error.log - ESDE_CA1_Patched - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
ESDE CA1 PATCHED
backend > logs > error.log
1 04-06-2023 18:23:20:2320 esc[31merroresc[39m: esc[31m{esc[39m
2 esc[31m userAgent: 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0 Safari/537.36',esc[39m
3 esc[31m method: 'POST',esc[39m
4 esc[31m route: '/api/user/login',esc[39m
5 esc[31m email: 'albert@admin.com',esc[39m
6 esc[31m ip: '124.246.66.203',esc[39m
7 esc[31m error: 'Password does not match',esc[39m
8 esc[31m status: 401esc[39m
9 esc[31m}esc[39m
10 |
```

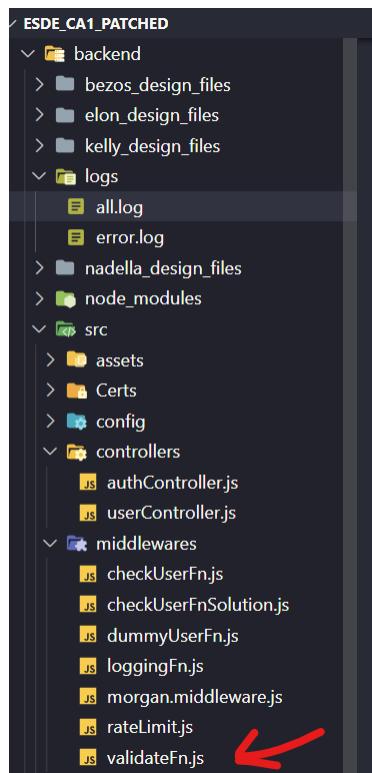
7. Bonus

7.0.1 Validating Invalid File Types

In the design submission page, there is a file submission available for users to submit their designs, however there are times where hackers may submit malicious files. So, let's add a validation middleware to the design submission endpoint to validate the file type and size before it is submitted to cloudinary.

Step 1:

Go into the validateFn.js file located under
/backend/src/middlewares/validateFn.js



Step 2:

Create a new function called validateFileTypeAndSize and add the following code as shown below.

```
110  validateFileTypeAndSize: function (req, res, next) {
111    var submittedFile = req.body.file;
112    console.log("Validation for file type: " + submittedFile);
113
114    var fileName = path.basename(submittedFile.path);
115    console.log("Submitted file path: " + submittedFile.path);
116    console.log("Path base name :" + path.basename(submittedFile.path));
117
118    var fileExtension = fileName.split(".");
119    fileExtension = fileExtension[fileExtension.length - 1];
120
121    var validFileSize = false;
122
123    //Get file size in bytes
124    var fileStat = fs.statSync(submittedFile.path);
125    var fileSizeInBytes = fileStat.size;
126    console.log(`File size in bytes: ${fileSizeInBytes / 1000000}MB`);
127
128    //Check file size
129    if (fileSizeInBytes <= 1000000) {
130      validFileSize = true;
131    } else {
132      console.log("File size exceeded 1MB");
133    }
134
135    var validFileType = false;
136
137    switch (fileExtension.toLowerCase()) {
138      case "jpg":
139        validFileType = true;
140        break;
141      case "jpeg":
142        validFileType = true;
143        break;
144      case "png":
145        validFileType = true;
146        break;
147      case "gif":
148        validFileType = true;
149        break;
150    }
151
152    if (validFileSize == false || validFileType == false) {
153      console.log("Invalid file type/size received");
154      fs.unlink(submittedFile.path, function (err) {
155        if (err) return console.log(err);
156        console.log("Deleted file successfully");
157      });
158  }
```

Continued Below...

```

152   if ([validFileSize == false || validFileType == false]) {
153     console.log("Invalid file type/size received");
154     fs.unlink(submittedFile.path, function (err) {
155       if (err) return console.log(err);
156       console.log("Deleted file successfully");
157     });
158
159     if (!validFileSize) {
160       let message = "Invalid file size. File size must be less than 1MB.";
161       getIPAddress()
162         .then((ip) => {
163           logger.error({
164             userAgent: req.userAgent.source,
165             method: req.method,
166             route: req.originalUrl,
167             ip: ip,
168             userId: req.body.userId,
169             error: message,
170             status: 400,
171           });
172         })
173         .catch((error) => {
174           console.log(error);
175         });
176
177       return res.status(400).send({ message: message });
178     } else if (!validFileType) {
179       let message = "Invalid file type. Only jpg/jpeg/png/gif are allowed.";
180       getIPAddress()
181         .then((ip) => {
182           logger.error({
183             userAgent: req.userAgent.source,
184             method: req.method,
185             route: req.originalUrl,
186             ip: ip,
187             userId: req.body.userId,
188             error: message,
189             status: 400,
190           });
191         })
192         .catch((error) => {
193           console.log(error);
194         });
195
196       return res.status(400).send({ message: message });
197     }
198   } else {
199     console.log("Validation Passed");
200     next();
201   }
202 },
203

```

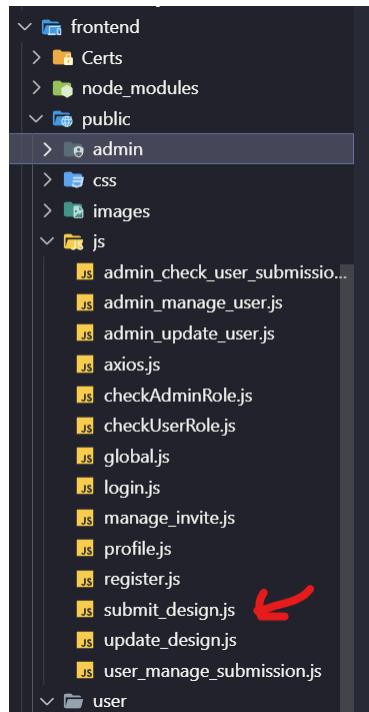
Step 3:

Let's add the function we created in routes.js under /backend/src/routes.js

```
// USER SUBMIT DESIGN
router.post(
  "/api/user/process-submission",
  loggingFn.logData,
  checkUserFnSolution.checkForValidUser,
  validateFn.validateDesignTitleAndDesc,
  validateFn.validateFileTypeAndSize, ↖
  rateLimitFn.designSubmissionCountLimit,
  userController.processDesignSubmission
);
// USER UPDATE DESIGN (RETROACTIVE PHASE)
```

Step 4:

Let's go to the submit_design.js file under /frontend/public/js/submit_design.js



Step 5:

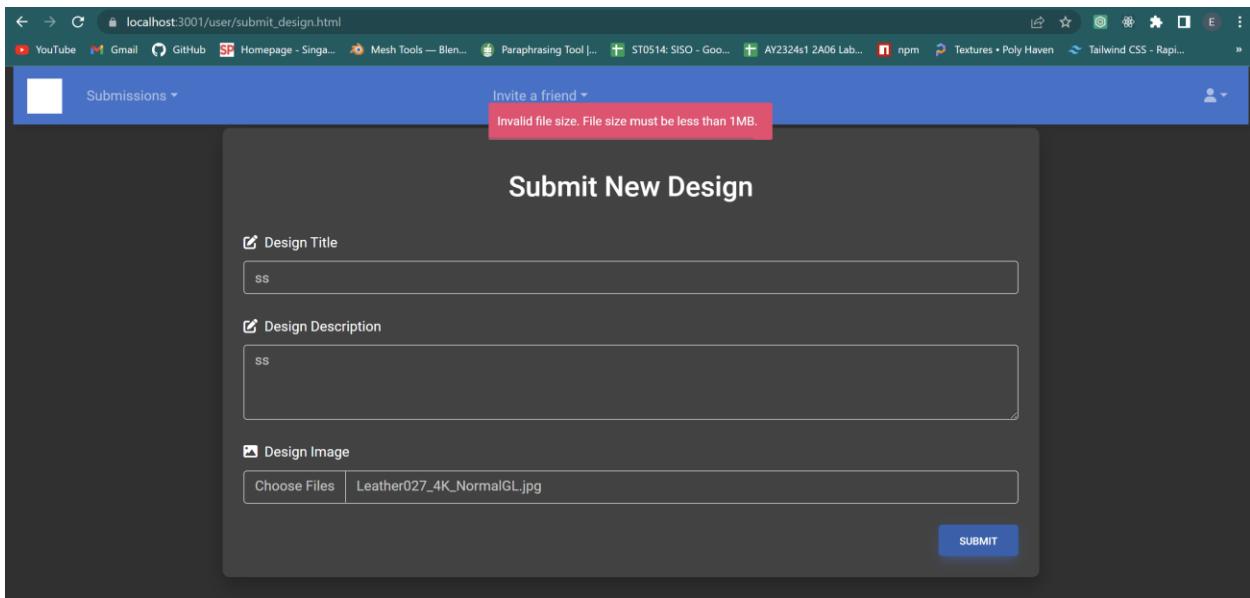
Change the following static error message to the following so that we will be able to display the respective error messages when the user submits a file type or size that is not valid.

```
JS checkAdminRole.js  JS checkUserRole.js  JS login.js  JS manage_invite.js  JS profile.js
frontend > public > js > submit_design.js > on("click") callback > then() callback
2/           user: userId,
28          authorization: `Bearer ${token}`,
29        },
30      })
31      .then(function (response) {
32        Noty.overrideDefaults({
33          callbacks: {
34            onTemplate: function () {
35              if (this.options.type === "systemresponse") {
36                this.barDom.innerHTML =
37                  '<div class="my-custom-template noty_body">';
38                this.barDom.innerHTML +=
39                  '<div class="text-black noty-header">Message:</div>';
40                this.barDom.innerHTML +=
41                  '<p class="text-black noty-message-body">' +
42                  this.options.text +
43                  "</p>";
44                this.barDom.innerHTML +=
45                  '<img src=' + this.options.imageURL + '">';
46                this.barDom.innerHTML += "<div>";
47              }
48            },
49          },
50        });
51
52        new Noty({
53          type: "systemresponse",
54          layout: "topCenter",
55          timeout: "5000",
56          text: response.data.message,
57          imageURL: response.data.imageURL,
58        }).show();
59      })
60      .catch(function (response) {
61        //Handle error
62        console.dir(response);
63        new Noty({
64          type: "error",
65          timeout: "6000",
66          layout: "topCenter",
67          theme: "sunset",
68          text: response.response.data.message, ←
69        }).show();
70      });
71    });
72  } //End of checking for $submitDesignFormContainer jQuery object
73
```

Step 6:

Let's test if we can submit file types and file sizes before trying to submit to clouddinary.

 Leather027_4K_Normal...	6/6/2023 2:38 pm	JPG File	25,427 KB
 maliciousCode	26/5/2023 3:52 pm	Text Document	1 KB



localhost:3001/user/submit_design.html

Submissions ▾

Invite a friend ▾

Invalid file size. File size must be less than 1MB.

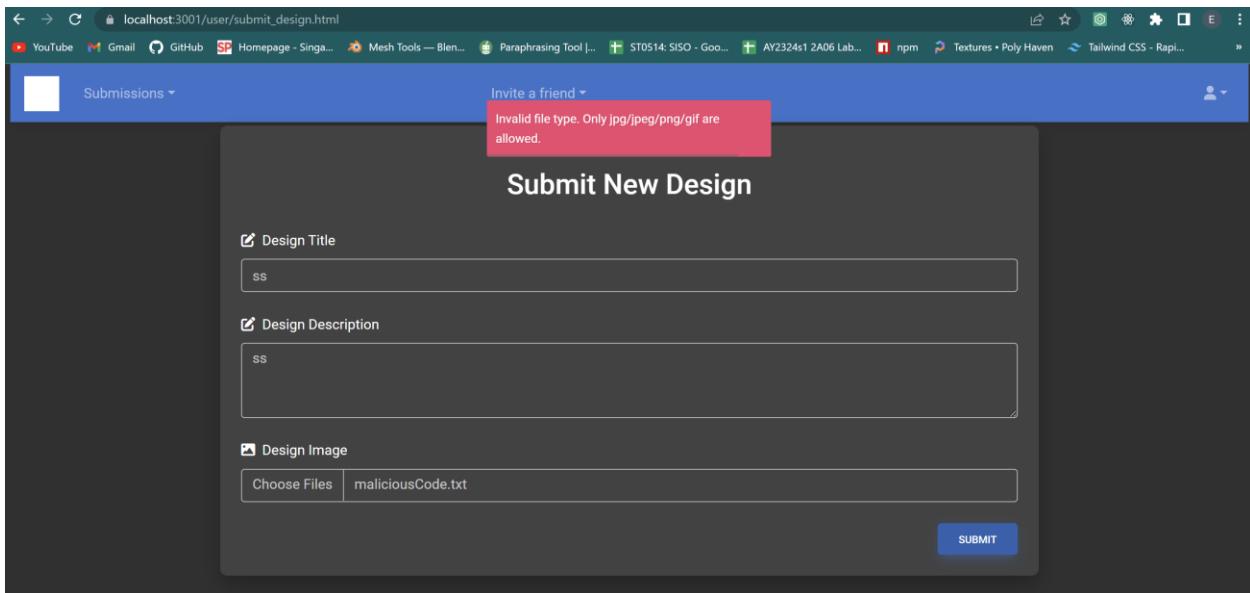
Submit New Design

Design Title
ss

Design Description
ss

Design Image
Choose Files | Leather027_4K_NormalGL.jpg

SUBMIT



localhost:3001/user/submit_design.html

Submissions ▾

Invite a friend ▾

Invalid file type. Only jpg/jpeg/png/gif are allowed.

Submit New Design

Design Title
ss

Design Description
ss

Design Image
Choose Files | maliciousCode.txt

SUBMIT

Step 7:

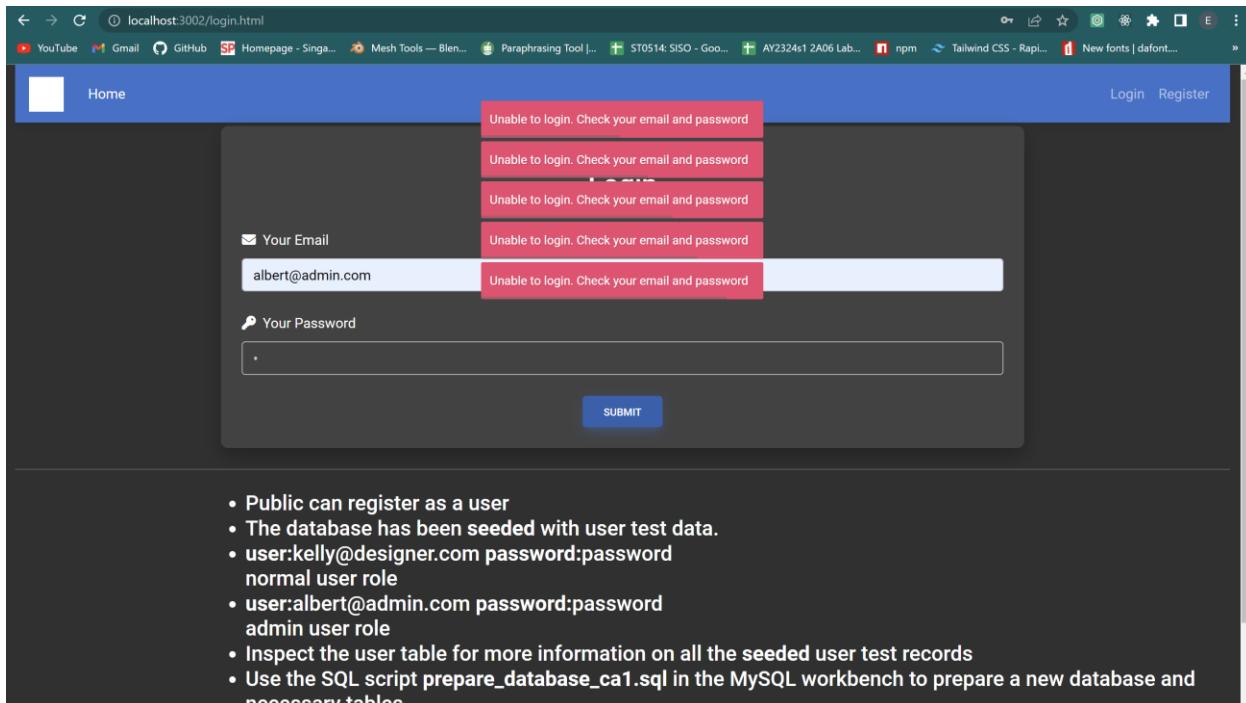
You will also see in your terminal that the errors have been successfully logged as well to have sufficient logging to track any suspicious activities. This shows that we have successfully validated the file type and size before sending it to clouddinary.

```
36
37 07-06-2023 18:26:39:2639 esc[31merroresc[39m: esc[31m{esc[39m
38 esc[31m userAgent: 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0 Safari/537.36',esc[39m
39 esc[31m method: 'POST',esc[39m
40 esc[31m route: '/api/user/process-submission',esc[39m
41 esc[31m ip: '124.246.66.203',esc[39m
42 esc[31m userId: 100,esc[39m
43 esc[31m error: 'Invalid file size. File size must be less than 1MB.',esc[39m
44 esc[31m status: 400esc[39m
45 esc[31m}esc[39m
46 07-06-2023 18:26:56:2656 esc[31merroresc[39m: esc[31m{esc[39m
47 esc[31m userAgent: 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0 Safari/537.36',esc[39m
48 esc[31m method: 'POST',esc[39m
49 esc[31m route: '/api/user/process-submission',esc[39m
50 esc[31m ip: '124.246.66.203',esc[39m
51 esc[31m userId: 100,esc[39m
52 esc[31m error: 'Invalid file type. Only jpg/jpeg/png/gif are allowed.',esc[39m
53 esc[31m status: 400esc[39m
54 esc[31m}esc[39m
55
```

7.1.1 How to exploit vulnerability – Limiting brute forcing attacks

Step 1:

Go into the vulnerable website and try to login in with albert's email but for the password let's try to input incorrect ones to imitate an actual brute forcing scenario.



The screenshot shows a web browser window with the URL `localhost:3002/login.html`. The page has a dark theme with a blue header bar containing a logo, the word "Home", and "Login" and "Register" links. Below the header is a form with two input fields: "Your Email" and "Your Password". The "Your Email" field contains the value "albert@admin.com". The "Your Password" field contains a single dot ("."). To the right of the password field is a red error message box that says "Unable to login. Check your email and password". This error message is repeated four times vertically, indicating that the password was incorrect for all four attempts. A blue "SUBMIT" button is located at the bottom of the form.

- Public can register as a user
- The database has been seeded with user test data.
- **user:kelly@designer.com password:password**
normal user role
- **user:albert@admin.com password:password**
admin user role
- Inspect the user table for more information on all the seeded user test records
- Use the SQL script `prepare_database_ca1.sql` in the MySQL workbench to prepare a new database and necessary tables.

Step 2:

On the other hand, there will be a chance where the server will crash due to overwhelming requests.

An example would be using burp suite to brute force the password of albert by sending a huge number of password combinations to try to guess albert's password.

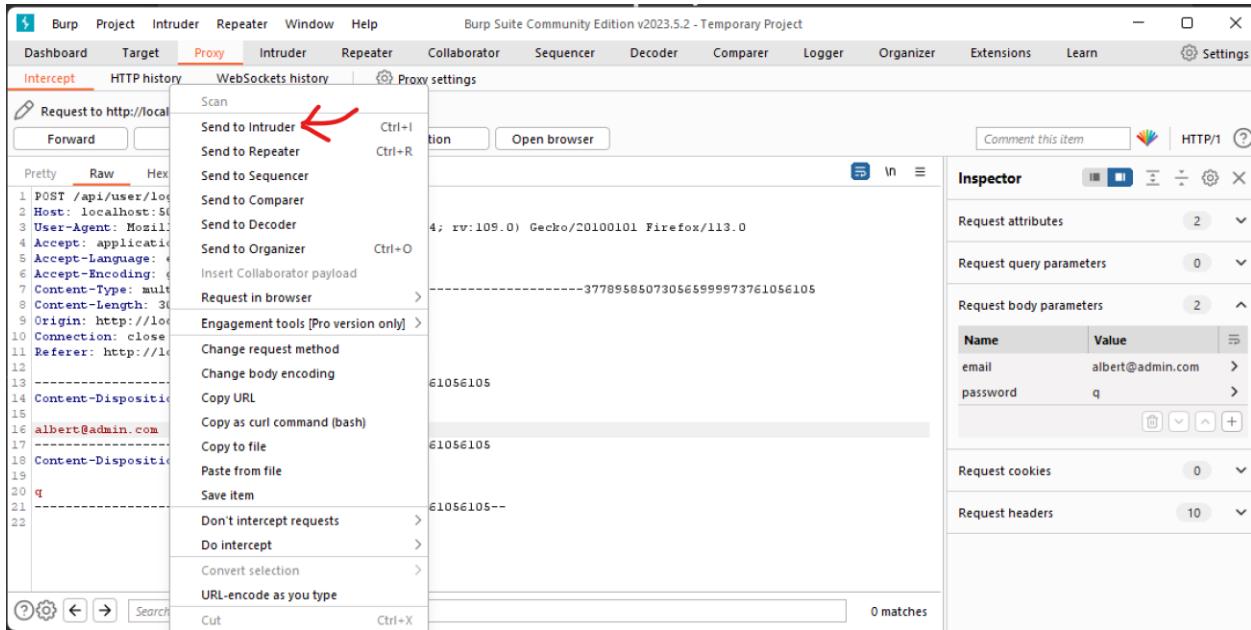
Open your Firefox browser and intercept it using burp suite and stop at the login intercept as shown.

The screenshot shows the Burp Suite interface with a captured POST request to `/api/user/login`. The request body is multipart/form-data with boundary `-----37785585073056599973761056105`. It contains two fields: `email` with value `albert@admin.com` and `password` with value `q`. The Inspector panel on the right displays the request attributes, query parameters, body parameters, cookies, and headers for this request.

```
1 POST /api/user/login HTTP/1.1
2 Host: localhost:5001
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/113.0
4 Accept: application/json, text/plain, */*
5 Accept-Language: en-US, en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: multipart/form-data; boundary=-----37785585073056599973761056105
8 Content-Length: 305
9 Origin: http://localhost:3002
10 Connection: close
11 Referer: http://localhost:3002/
12
13 -----37785585073056599973761056105
14 Content-Disposition: form-data; name="email"
15
16 albert@admin.com
17 -----37785585073056599973761056105
18 Content-Disposition: form-data; name="password"
19
20 q
21 -----37785585073056599973761056105--
```

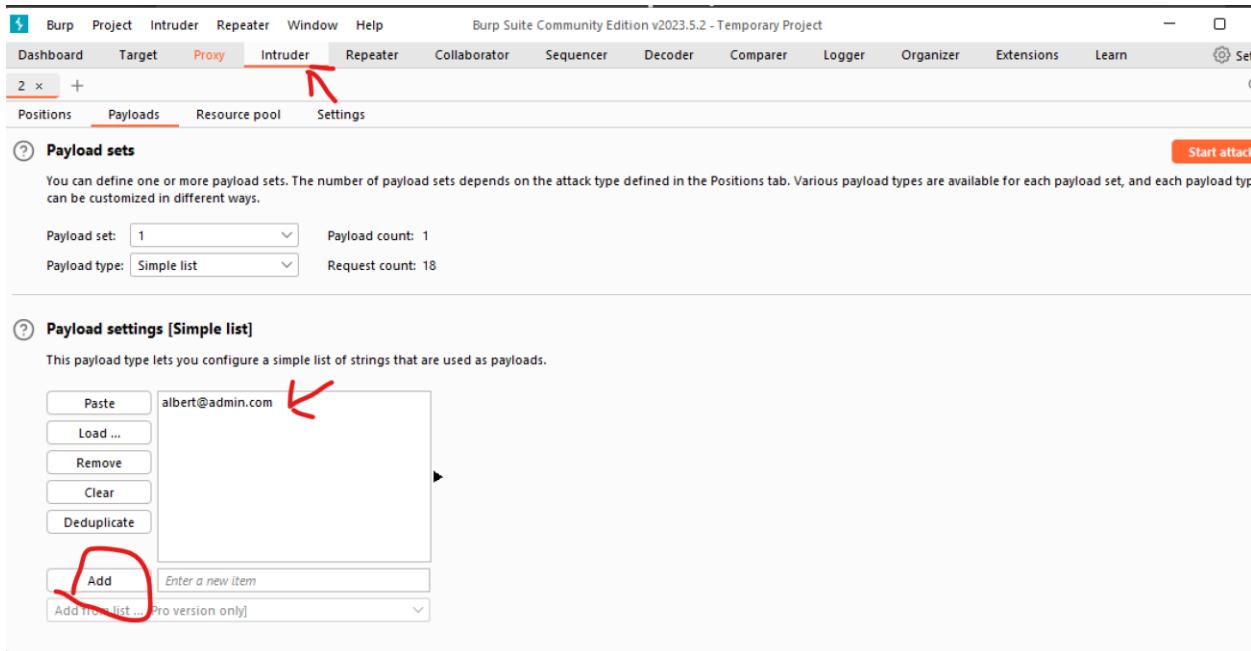
Step 3:

Right click and select Send to Intruder to set up the brute forcing.



Step 4:

Click on Intruder at the top and add albert@admin.com into eh payload settings.



Step 5:

Select 2 in the payload set and fill in the passwords that we will try to use for brute forcing.

Burp Suite Community Edition v2023.5.2 - Temporary Project

Dashboard Target Proxy **Intruder** Repeater Window Help

Positions Payloads Resource pool Settings

Payload sets

You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various payload types are available for each payload set, and each payload type can be customized in different ways.

Payload set: **2** (Red arrow) Payload count: 18

Payload type: Simple list Request count: 18

Start attack

Payload settings [Simple list]

This payload type lets you configure a simple list of strings that are used as payloads.

a
b
c (Red arrow)
d
d
dag
egs

Paste
Load ...
Remove
Clear
Deduplicate
Add
Add from list ... [Pro version only]

Step 6:

After adding the passwords click on start attack and you should see something as shown below where they will brute force the login page using the passwords that you provided.

Request	Payload 1	Payload 2	Status code	Error	Timeout	Length	Comment
0			500	<input type="checkbox"/>	<input type="checkbox"/>	274	
1	albert@admin.com	a	500	<input type="checkbox"/>	<input type="checkbox"/>	288	
2	albert@admin.com	b	500	<input type="checkbox"/>	<input type="checkbox"/>	288	
3	albert@admin.com	c	500	<input type="checkbox"/>	<input type="checkbox"/>	288	
4	albert@admin.com		500	<input type="checkbox"/>	<input type="checkbox"/>	288	
5	albert@admin.com		500	<input type="checkbox"/>	<input type="checkbox"/>	288	
6	albert@admin.com	d	500	<input type="checkbox"/>	<input type="checkbox"/>	288	
7	albert@admin.com	d	500	<input type="checkbox"/>	<input type="checkbox"/>	288	
8	albert@admin.com	dag	500	<input type="checkbox"/>	<input type="checkbox"/>	288	
9	albert@admin.com	egs	500	<input type="checkbox"/>	<input type="checkbox"/>	288	
10	albert@admin.com	hvg	500	<input type="checkbox"/>	<input type="checkbox"/>	288	
11	albert@admin.com	aa	500	<input type="checkbox"/>	<input type="checkbox"/>	288	
12	albert@admin.com	bb	500	<input type="checkbox"/>	<input type="checkbox"/>	288	
13	albert@admin.com	cc	500	<input type="checkbox"/>	<input type="checkbox"/>	288	
14	albert@admin.com	cc	500	<input type="checkbox"/>	<input type="checkbox"/>	288	

7.1.2 How to rectify vulnerability - Limiting brute forcing attacks

Step 1:

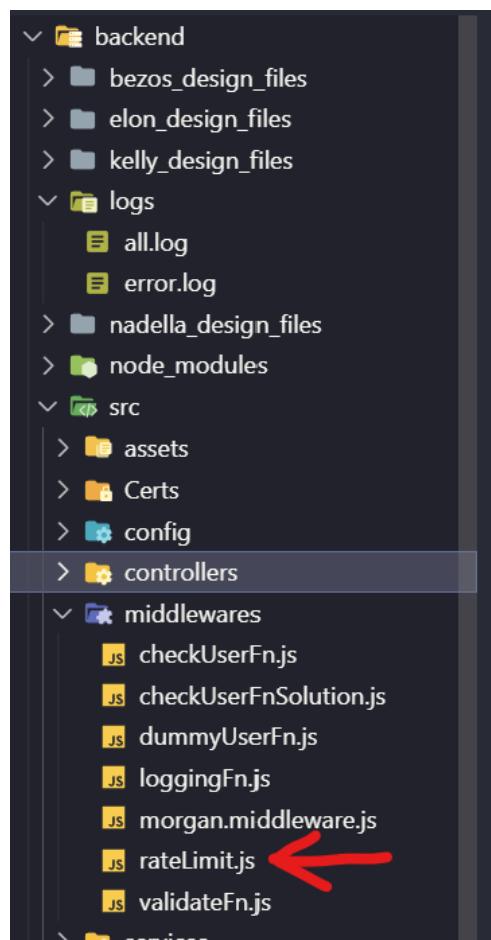
Let's install a library called express-rate-limit which will help us to limit the number of requests sent by a user at a given time to prevent brute forcing attacks.

Run the following command in your backend terminal:

```
npm install express-rate-limit
```

Step 2:

Next, we will create a new file called rateLimit.js under /backend/src/middlewares for us to limit the number of tries allowed for the login.

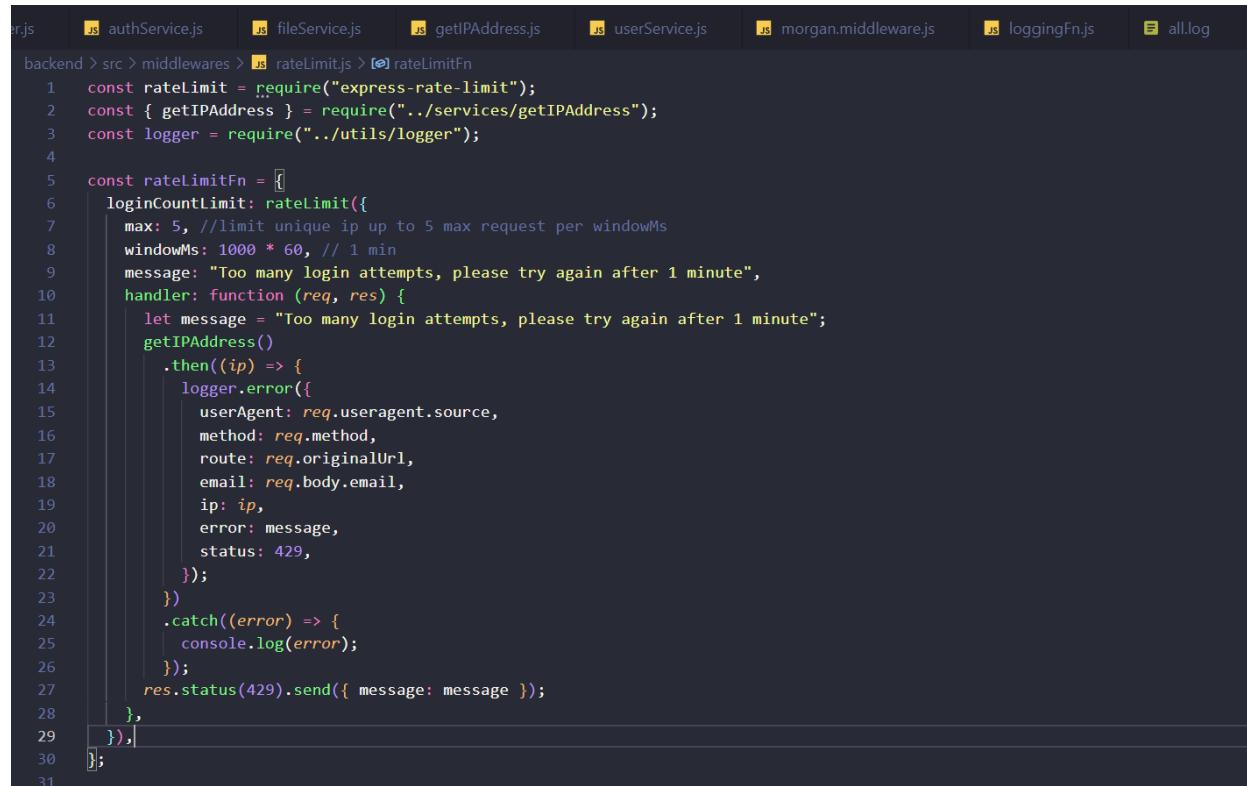


Step 3:

Add the following code into the rateLimit.js file under /backend/src/middlewares.

We will allow the user to login in a maximum of 5 times and if they are still not logged in they will need to wait for 1 minute before being able to attempt to login again. We will also add a log here to record if anyone has passed the 5 login tries.

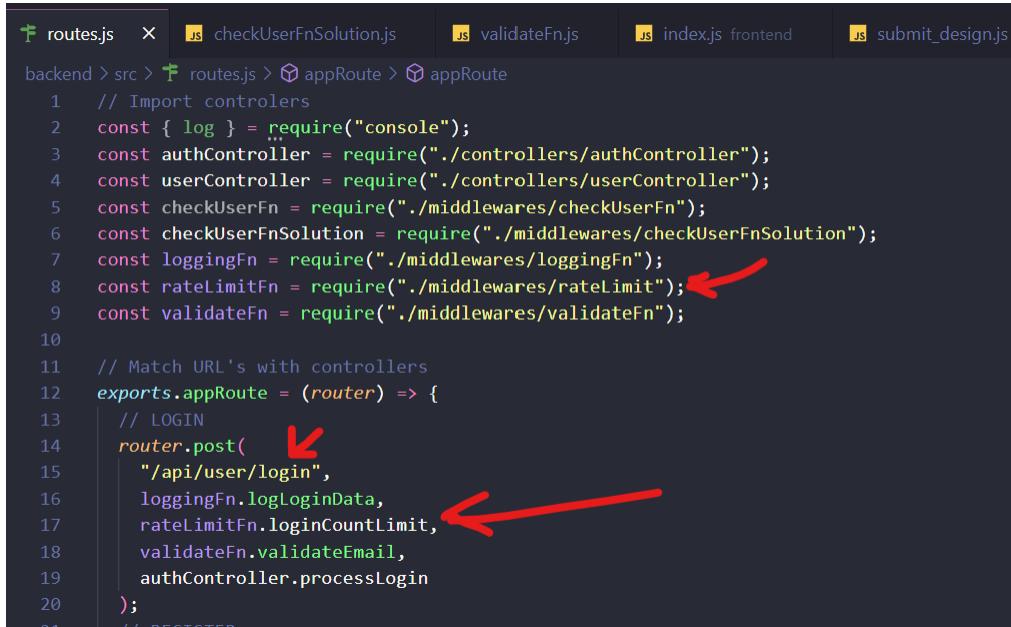
Reference: <https://blog.logrocket.com/rate-limiting-node-js/>



```
backend > src > middlewares > rateLimit.js > rateLimitFn
1 const rateLimit = require("express-rate-limit");
2 const { getIPAddress } = require("../services/getIPAddress");
3 const logger = require("../utils/logger");
4
5 const rateLimitFn = [
6   loginCountLimit: rateLimit({
7     max: 5, //limit unique ip up to 5 max request per windowMs
8     windowMs: 1000 * 60, // 1 min
9     message: "Too many login attempts, please try again after 1 minute",
10    handler: function (req, res) {
11      let message = "Too many login attempts, please try again after 1 minute";
12      getIPAddress()
13        .then((ip) => {
14          logger.error({
15            userAgent: req.useragent.source,
16            method: req.method,
17            route: req.originalUrl,
18            email: req.body.email,
19            ip: ip,
20            error: message,
21            status: 429,
22          });
23        })
24        .catch((error) => {
25          console.log(error);
26        });
27      res.status(429).send({ message: message });
28    },
29  }],
30];
31
```

Step 4:

We will now add the rateLimit.js into routes.js under /backend/src/routes.js. Make sure to import the rateLimitFn into routes.js as shown below.

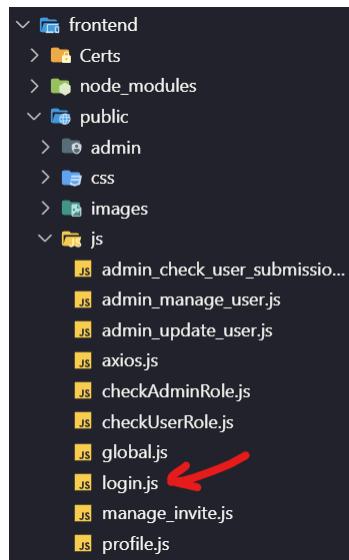


```
routes.js  X  JS checkUserFnSolution.js  JS validateFn.js  JS index.js frontend  JS submit_design.js

backend > src > routes.js > appRoute > appRoute
1 // Import controllers
2 const { log } = require("console");
3 const authController = require("./controllers/authController");
4 const userController = require("./controllers/userController");
5 const checkUserFn = require("./middlewares/checkUserFn");
6 const checkUserFnSolution = require("./middlewares/checkUserFnSolution");
7 const loggingFn = require("./middlewares/loggingFn");
8 const rateLimitFn = require("../middlewares/rateLimit"); ←
9 const validateFn = require("./middlewares/validateFn");
10
11 // Match URL's with controllers
12 exports.appRoute = (router) => {
13   // LOGIN
14   router.post(←
15     "/api/user/login",
16     loggingFn.logLoginData,
17     rateLimitFn.loginCountLimit, ←
18     validateFn.validateEmail,
19     authController.processLogin
20   );
21   // REGISTERED
22 }
```

Step 5:

Head over to the frontend code and look for the file called login.js under /frontend/public/js/login.js.



Step 6:

Change the static response "Unable to login. Check your email and password" to the following response message. This is so that we would be able to see the message when we get pass the 5 tries.



```
frontend > public > js > loginjs > on("click") callback > catch() callback > text
  const baseUrl = "https://localhost:5000",
12  let email = $("#emailInput").val();
13  let password = $("#passwordInput").val();
14  let webFormData = new FormData();
15  webFormData.append("email", email);
16  webFormData.append("password", password);
17  axios({
18    method: "post",
19    url: baseUrl + "/api/user/login",
20    data: webFormData,
21    headers: { "Content-Type": "multipart/form-data" },
22  })
23    .then(function (response) {
24      //Inspect the object structure of the response object.
25      //console.log('Inspecting the respone object returned from the
26      //console.dir(response);
27      userData = response.data;
28
29      if (userData.role_name == "user") {
30        sessionStorage.setItem("token", userData.token);
31        sessionStorage.setItem("user_id", userData.user_id);
32        sessionStorage.setItem("role_name", userData.role_name);
33        window.location.replace("user/manage_submission.html");
34        return;
35      }
36      if (response.data.role_name == "admin") {
37        sessionStorage.setItem("token", userData.token);
38        sessionStorage.setItem("user_id", userData.user_id);
39        sessionStorage.setItem("role_name", userData.role_name);
40        window.location.replace("admin/manage_users.html");
41        return;
42      }
43    })
44    .catch(function (response) {
45      //Handle error
46      console.dir(response);
47      new Noty({
48        type: "error",
49        layout: "topCenter",
50        theme: "sunset",
51        timeout: "6000",
52        text: response.response.data.message, ↖
53      }).show();
54    });
55  });
56 } //End of checking for $loginFormContainer jQuery object
```

7.1.3 Evidence that the vulnerability has been rectified - Limiting brute forcing attacks

Step 1:

Now we will see if our rateLimit is working let's clear the error.log and try to login by clicking it 6 times at one go using an incorrect password. You should see a different alert message after submitting more than 6 times in a row stating that you have done too many requests at once.

A screenshot of a web browser window showing a login form. The URL is 'localhost:3001/login.html'. The form has fields for 'Your Email' (containing 'albert@admin.com') and 'Your Password' (containing a masked password). There are six red error messages stacked vertically above the password field, each reading 'Invalid login credentials. Please try again.' A blue 'SUBMIT' button is at the bottom. The background is dark grey, and the error messages are in red boxes.

A screenshot of a web browser window showing a login form. The URL is 'localhost:3001/login.html'. The form has fields for 'Your Email' and 'Your Password'. Above the password field is a red error message box containing the text 'Too many login attempts, please try again after 1 minute'. A red arrow points from the left towards this message. A blue 'SUBMIT' button is at the bottom. The background is dark grey, and the error message is in red.

Step 2:

Head over to the error.log file. This time you should see that there are 5 login error attempts and the 6th being the rateLimit Function that we created, stopping anymore further requests including logging it as shown below after reaching the maximum of 5 tries for the number of allowed requests per window per user.

This shows that we have successfully limited the login count to prevent any brute forcing.

```
lions.js admin_manage_user.js admin_update_user.js authController.js authService.js fileService.js getAddress.js userService.js morgan.middleware.js loggingfn.js
backend > logs > error.log
1 04-06-2023 18:34:12:3412 esc[31merroresc[39m: esc[31m{esc[39m
2 esc[31m userAgent: 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0 Safari/537.36',esc[39m
3 esc[31m method: 'POST',esc[39m
4 esc[31m route: '/api/user/login',esc[39m
5 esc[31m email: 'albert@admin.com',esc[39m
6 esc[31m ip: '124.246.66.203',esc[39m
7 esc[31m error: 'Password does not match',esc[39m
8 esc[31m status: 401esc[39m
9 esc[31mesc[39m
10 04-06-2023 18:34:13:3413 esc[31merroresc[39m: esc[31m{esc[39m
11 esc[31m userAgent: 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0 Safari/537.36',esc[39m
12 esc[31m method: 'POST',esc[39m
13 esc[31m route: '/api/user/login',esc[39m
14 esc[31m email: 'albert@admin.com',esc[39m
15 esc[31m ip: '124.246.66.203',esc[39m
16 esc[31m error: 'Password does not match',esc[39m
17 esc[31m status: 401esc[39m
18 esc[31mesc[39m
19 04-06-2023 18:34:13:3413 esc[31merroresc[39m: esc[31m{esc[39m
20 esc[31m userAgent: 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0 Safari/537.36',esc[39m
21 esc[31m method: 'POST',esc[39m
22 esc[31m route: '/api/user/login',esc[39m
23 esc[31m email: 'albert@admin.com',esc[39m
24 esc[31m ip: '124.246.66.203',esc[39m
25 esc[31m error: 'Password does not match',esc[39m
26 esc[31m status: 401esc[39m
27 esc[31mesc[39m
28 04-06-2023 18:34:15:3415 esc[31merroresc[39m: esc[31m{esc[39m
29 esc[31m userAgent: 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0 Safari/537.36',esc[39m
30 esc[31m method: 'POST',esc[39m
31 esc[31m route: '/api/user/login',esc[39m
32 esc[31m email: 'albert@admin.com',esc[39m
33 esc[31m ip: '124.246.66.203',esc[39m
34 esc[31m error: 'Password does not match',esc[39m
35 esc[31m status: 401esc[39m
36 esc[31mesc[39m
37 04-06-2023 18:34:16:3416 esc[31merroresc[39m: esc[31m{esc[39m
38 esc[31m userAgent: 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0 Safari/537.36',esc[39m
39 esc[31m method: 'POST',esc[39m
40 esc[31m route: '/api/user/login',esc[39m
41 esc[31m email: 'albert@admin.com',esc[39m
42 esc[31m ip: '124.246.66.203',esc[39m
43 esc[31m error: 'Password does not match',esc[39m
44 esc[31m status: 401esc[39m
45 esc[31mesc[39m
46 04-06-2023 18:34:16:3416 esc[31merroresc[39m: esc[31m{esc[39m
47 esc[31m userAgent: 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0 Safari/537.36',esc[39m
48 esc[31m method: 'POST',esc[39m
49 esc[31m route: '/api/user/login',esc[39m
50 esc[31m email: 'albert@admin.com',esc[39m
51 esc[31m ip: '124.246.66.203',esc[39m
52 esc[31m error: 'Too many login attempts, please try again after 1 minute',esc[39m
53 esc[31m status: 429esc[39m
54 esc[31mesc[39m
```

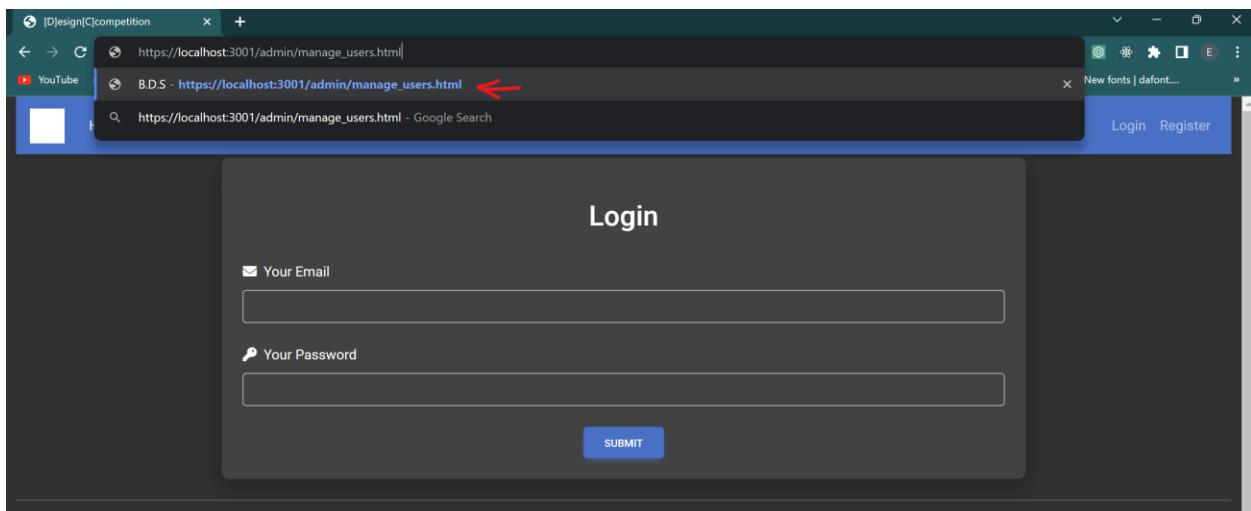


7.2.1 How to exploit vulnerability - About URL Link access control (Broken Access Control)

Step 1:

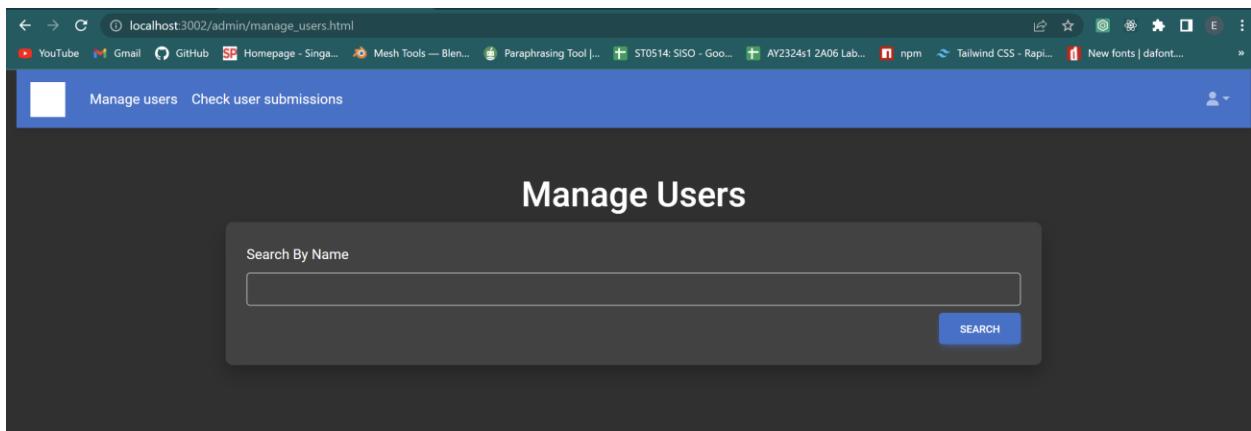
Paste the following URL link to see if we can access the admin page without logging in.

Link: yourLocalhostPort/admin/manage_users.html



Step 2:

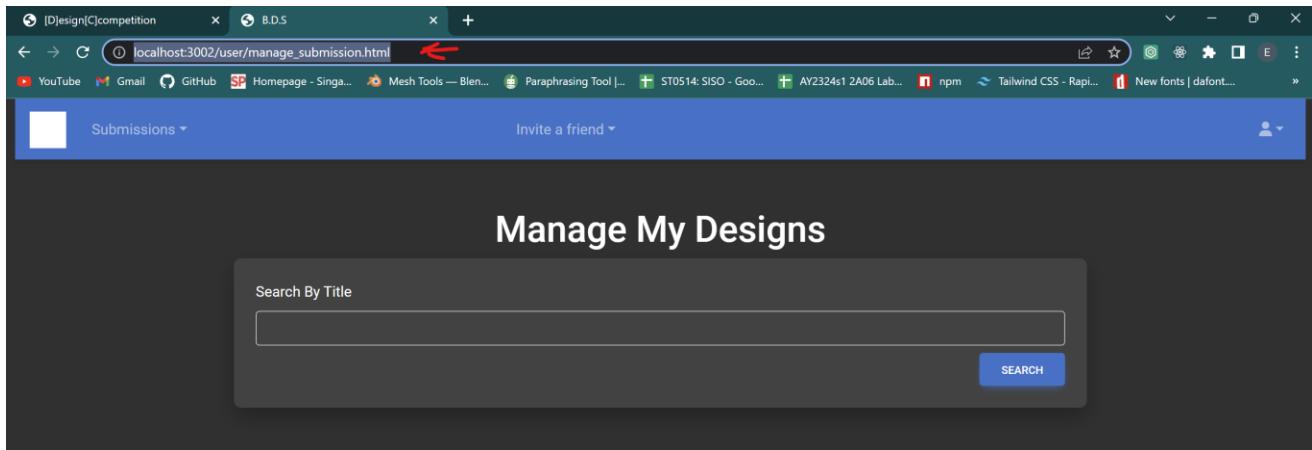
You will see that you would be able to access the admin page without logging in as an admin.



Step 3:

Similarly, we can also access the user page as well when we type the following link.

Link: yourLocalhostPort/user/manage_submission.html

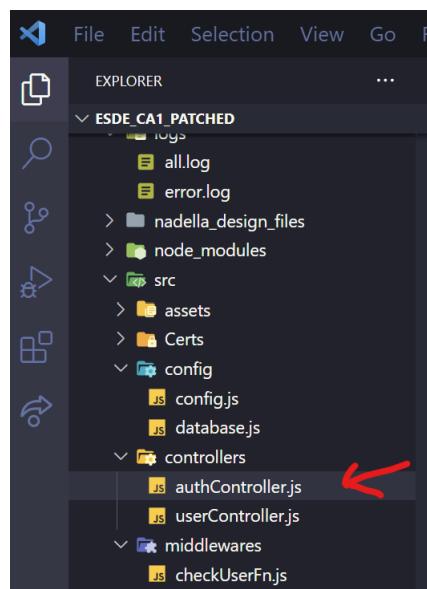


7.2.2 How to rectify vulnerability - About URL Link access

Step 1:

To rectify this vulnerability, let us first go to authController.js to do some changes to the token.

Under /backend/src/controllers/authController.js



Step 2:

Apply the following changes to the exports.processLogin. This will allow us to check the role of the current logged in user through the token.

```
Run Terminal Help authController.js - ESDE_CA1_Patched - Visual Studio Code
routes.js checkUserFnSolution.js submit_design.js update_design.js user_manage_submission.js userController.js authController.js

backend > src > controllers > authController.js > processLogin > processLogin > auth.authenticate() callback > info

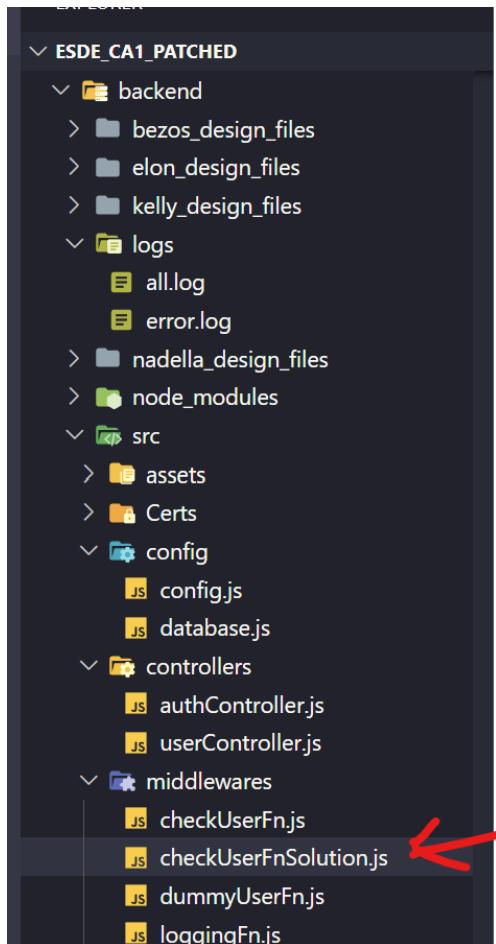
12     auth.authenticate(email, function (error, results) {
13       if (error) {
14         let message = "Credentials are not valid.";
15         //return res.status(500).json({ message: message });
16         //If the following statement replaces the above statement
17         //to return a JSON response to the client, the SQLMap or
18         //any attacker (who relies on the error) will be very happy
19         //because they relies a lot on SQL error for designing how to do
20         //attack and anticipate how much "rewards" after the effort.
21         //Rewards such as sabotage (seriously damage the data in database),
22         //data theft (grab and sell).
23         logger.error({
24           method: req.method,
25           route: req.originalUrl,
26           email: req.body.email,
27           // ip: requestIp.getClientIp(req),
28           ip: req.headers["x-forwarded-for"] || req.connection.remoteAddress,
29           info: error,
30         });
31         return res.status(500).json({ message: error });
32       } else {
33         if (results.length == 1) {
34           if (password == null || results[0] == null) {
35             return res.status(500).json({ message: "login failed" });
36           }
37           if (bcrypt.compareSync(password, results[0].user_password) == true) {
38             let data = {
39               user_id: results[0].user_id,
40               role_name: results[0].role_name,
41               token: jwt.sign(
42                 { id: results[0].user_id, role: results[0].role_name },
43                 config.JWTKey,
44                 {
45                   expiresIn: 86400, //Expires in 24 hrs
46                 }
47               ),
48             }; //End of data variable setup
49             return res.status(200).json(data);
50           } else {
51         }
52       }
53     }
54   }
55 }

56   return res.status(500).json({ message: "Email or password is incorrect" });
57 }

58 module.exports = auth;
```

Step 3:

Next, let us go into the checkUserFnSolution.js file located at
/backend/src/middlewares/checkUserFnSolution.js



Step 4:

Add the following code as shown below located at /backend/src/middlewares/checkUserFnSolution.js for us to retrieve the role from the token and verify if the token has a user or admin role or no token at all in the front end.

*Note: You can ignore lines 131-138 for now as this will be used only for the 6th vulnerability

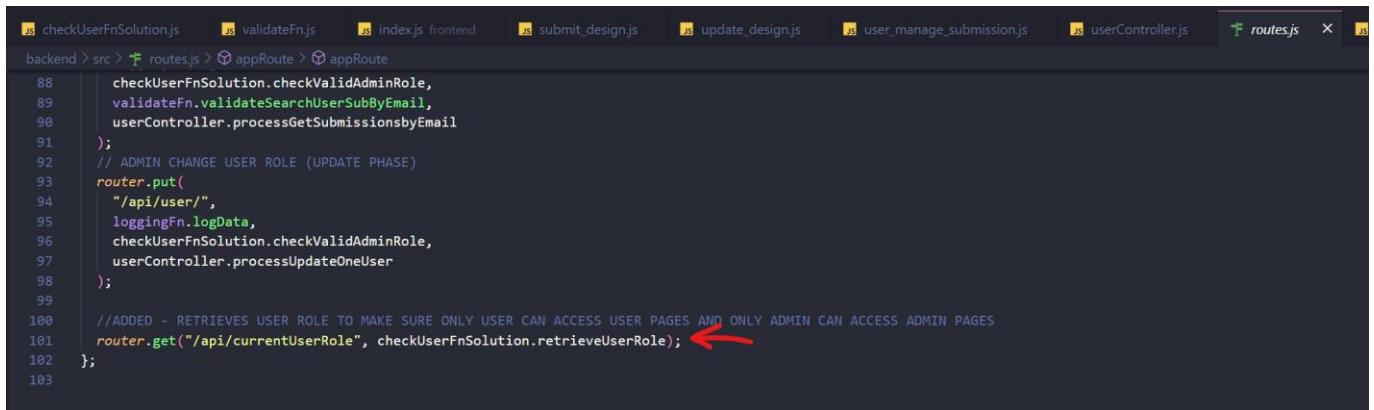


The screenshot shows a code editor with the file 'checkUserFnSolution.js' open. The code is written in JavaScript and handles an authorization header. It checks if the header is null or undefined, or if it does not start with 'Bearer'. If so, it returns a 403 status with an unauthorized message. Otherwise, it splits the header to get the token and logs the received token. It then uses the jwt library to verify the token against a config key. If verification fails, it logs an error and returns a 403 status. If successful, it returns a 200 status with the role information. The code includes imports for module.exports, req, res, next, jwt, and config. It also uses promises for the getIPAddress function and handles errors by logging them.

```
js checkUserFnSolution.js X js validateFn.js js index.js frontend js submit_design.js js up
backend > src > middlewares > js checkUserFnSolution.js > retrieveUserRole > retrieveUserRole > jwt.verify
78 module.exports.retrieveUserRole = (req, res, next) => {
79   const authHeader = req.headers["authorization"];
80
81   if (
82     authHeader === null ||
83     authHeader === undefined ||
84     !authHeader.startsWith("Bearer "))
85   {
86     console.log(
87       "authHeader is null or undefined or does not start with Bearer"
88     );
89     return res.status(403).send({ message: "Unauthorized access" });
90   } else {
91     console.log("Retrieving authorization header");
92     let token = authHeader.split(" ")[1];
93     // console.log("Check for received token from frontend : \n");
94     // console.log(token);
95
96     jwt.verify(token, config.JWTKey, (err, data) => {
97       if (err) {
98         getIPAddress()
99           .then((ip) => {
100             logger.error({
101               userAgent: req.useragent.source,
102               method: req.method,
103               route: req.originalUrl,
104               ip: ip,
105               info: "Unauthorized Access Triggered",
106               status: 403,
107             });
108           })
109           .catch((error) => {
110             console.log(error);
111           });
112             return res.status(403).send({ message: "Unauthorized access" });
113           } else {
114             return res.status(200).send({ role: data.role });
115           }
116         });
117       }
118     };
119   };
120 }
```

Step 5:

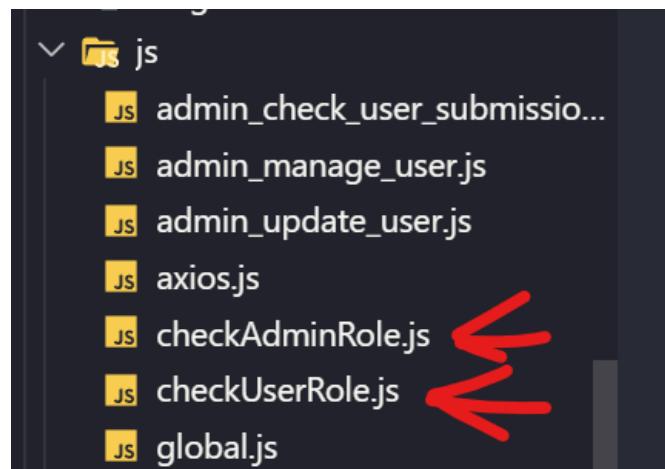
Next, head over to the routes.js file under /backend/src/routes.js to add the code that we previously added in Step 4 into a new endpoint created called currentUserRole.



```
checkUserFnSolution.js validateFn.js index.js frontend submit_design.js update_design.js user_manage_submission.js UserController.js routes.js
backend > src > routes.js > appRoute > appRoute
88     checkUserFnSolution.checkValidAdminRole,
89     validateFn.validateSearchUserSubByEmail,
90     UserController.processGetSubmissionsByEmail
91   );
92   // ADMIN CHANGE USER ROLE (UPDATE PHASE)
93   router.put(
94     "/api/user/",
95     loggingFn.logData,
96     checkUserFnSolution.checkValidAdminRole,
97     UserController.processUpdateOneUser
98   );
99
100  //ADDED - RETRIEVES USER ROLE TO MAKE SURE ONLY USER CAN ACCESS USER PAGES AND ONLY ADMIN CAN ACCESS ADMIN PAGES
101  router.get("/api/currentUserRole", checkUserFnSolution.retrieveUserRole); ↗
102
103
```

Step 6:

After adding the endpoint in routes.js we will now go to the front-end code to add some files that will help to check the authentication of the user. Go to /frontend/public/js folder and add two js files as shown below.



Step 7:

Lets' go into the checkAdminRole.js file first located under /frontend/public/js/checkAdminRole.js and add the following code.

We will use the endpoint that we created earlier to check if there is a token to begin with if there isn't, we will direct them back to the login page. Also, they will be able to go to the admin page if they have a valid token but will be redirected back to the user page if a non-admin user tries to access the admin page.



The screenshot shows a code editor with multiple tabs open. The active tab is 'checkAdminRole.js'. The code in the editor is as follows:

```
routes.js      js checkAdminRole.js X  js checkUserRole.js  js validateFn.js  js che
frontend > public > js > js checkAdminRole.js > ...
1  function checkUserRole() {
2    const baseUrl = "https://localhost:5000";
3
4    let token = sessionStorage.getItem("token");
5    let userId = sessionStorage.getItem("user_id");
6    axios({
7      headers: {
8        user: userId,
9        authorization: `Bearer ${token}`,
10     },
11     method: "get",
12     url: baseUrl + "/api/currentUserRole/",
13   })
14   .then(function (response) {
15     //retrieve the role of the user
16     if (response.data.role == "user") {
17       window.location.assign(
18         "https://localhost:3001/user/manage_submission.html"
19       );
20     } else if (response.data.role != "admin") {
21       sessionStorage.removeItem("token");
22       sessionStorage.removeItem("user_id");
23       sessionStorage.removeItem("role_name");
24       window.location.assign("https://localhost:3001/login.html");
25     }
26   })
27   .catch(function (response) {
28     //Handle error
29     console.dir(response);
30     sessionStorage.removeItem("token");
31     sessionStorage.removeItem("user_id");
32     sessionStorage.removeItem("role_name");
33     window.location.assign("https://localhost:3001/login.html");
34   });
35 }
36 checkUserRole();
```

Step 8:

Let us go to the checkUserRole.js file located under /frontend/public/js/checkUserRole.js and add the following code.

We will use the endpoint that we created earlier to check if there is a token to begin with if there is not we will direct them back to the login page. Also, they will be able to go to the user page if they have a valid token but will be redirected back to the admin page if an admin user tries to access the user page.



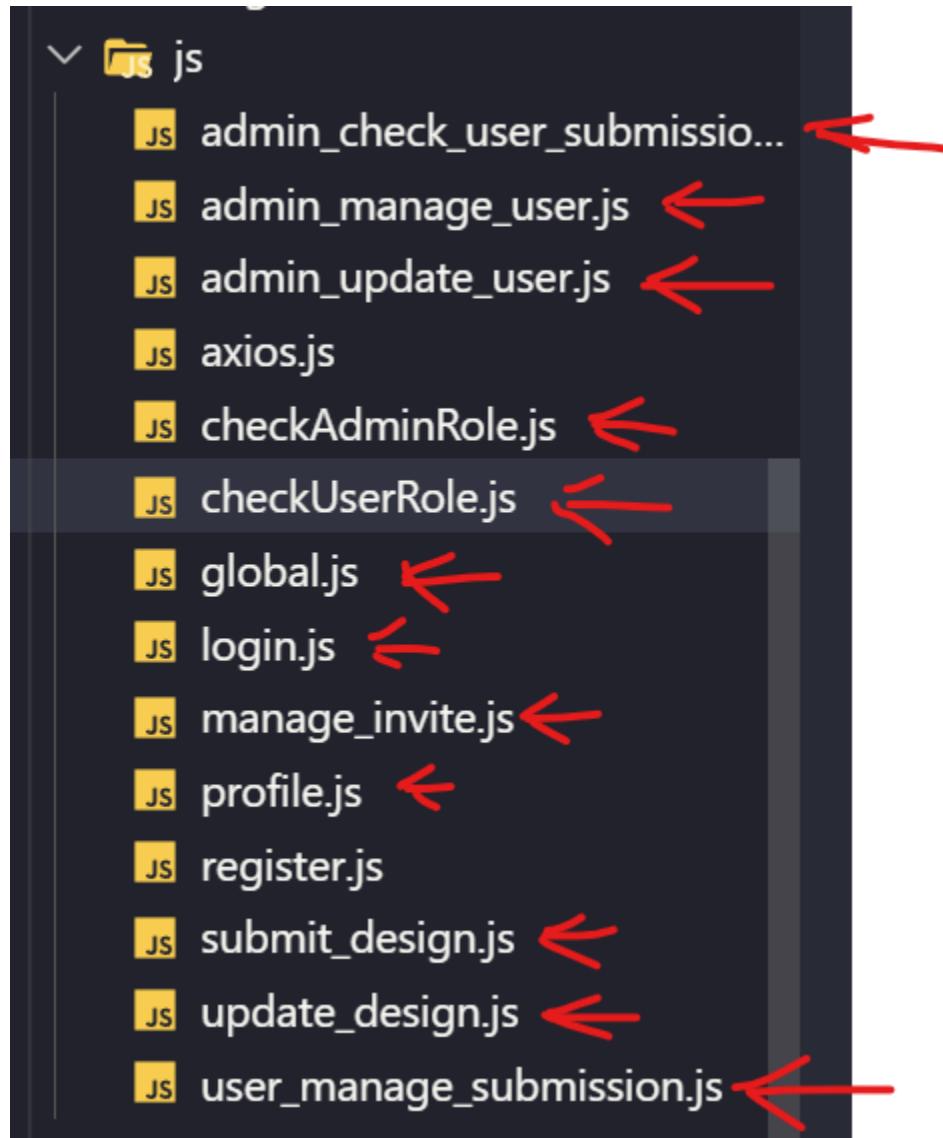
The screenshot shows a code editor with multiple tabs at the top: routes.js, checkAdminRole.js, checkUserRole.js (which is the active tab), validateFn.js, and checkUs. The code in checkUserRole.js is as follows:

```
routes.js          JS checkAdminRole.js    JS checkUserRole.js X    JS validateFn.js    JS checkUs
frontend > public > js > checkUserRole.js > ...
1  function checkUserRole() {
2      const baseUrl = "https://localhost:5000";
3
4      let token = sessionStorage.getItem("token");
5      let userId = sessionStorage.getItem("user_id");
6      axios({
7          headers: {
8              user: userId,
9              authorization: `Bearer ${token}`,
10         },
11         method: "get",
12         url: baseUrl + "/api/currentUserRole/",
13     })
14     .then(function (response) {
15         //retrieve the role of the user
16         if (response.data.role == "admin") {
17             window.location.assign(
18                 "https://localhost:3001/admin/manage_users.html"
19             );
20         } else if (response.data.role != "user") {
21             sessionStorage.removeItem("token");
22             sessionStorage.removeItem("user_id");
23             sessionStorage.removeItem("role_name");
24             window.location.assign("https://localhost:3001/login.html");
25         }
26     })
27     .catch(function (response) {
28         //Handle error
29         console.dir(response);
30         sessionStorage.removeItem("token");
31         sessionStorage.removeItem("user_id");
32         sessionStorage.removeItem("role_name");
33         window.location.assign("https://localhost:3001/login.html");
34     });
35 }
36 checkUserRole();
```

Step 9:

You would realize that the code above uses **session storage** instead of local storage. This is so that when the user exits the page without logging out, the token will automatically be deleted to prevent any unwanted use of the token.

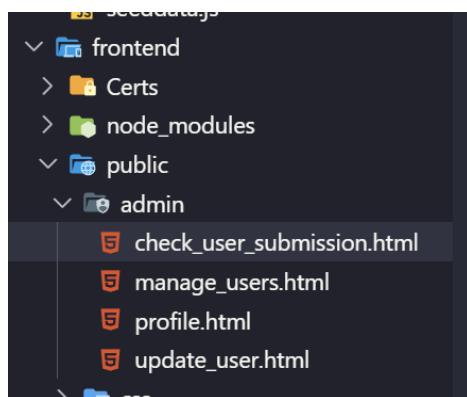
So, please do change all local storage to session storage for all the JavaScript files as shown in the bottom that has a red arrow pointing to it.



Step 10:

Next, let us go to the admin html pages and add a code to call the checkAdminRole.js file that we created just now for each of the respective admin role pages.

Under /frontend/public/admin/



Step 11:

Let us add the following code shown below to the above 4 admin html pages in the head tag to prevent anyone with no access from accessing the admin pages.

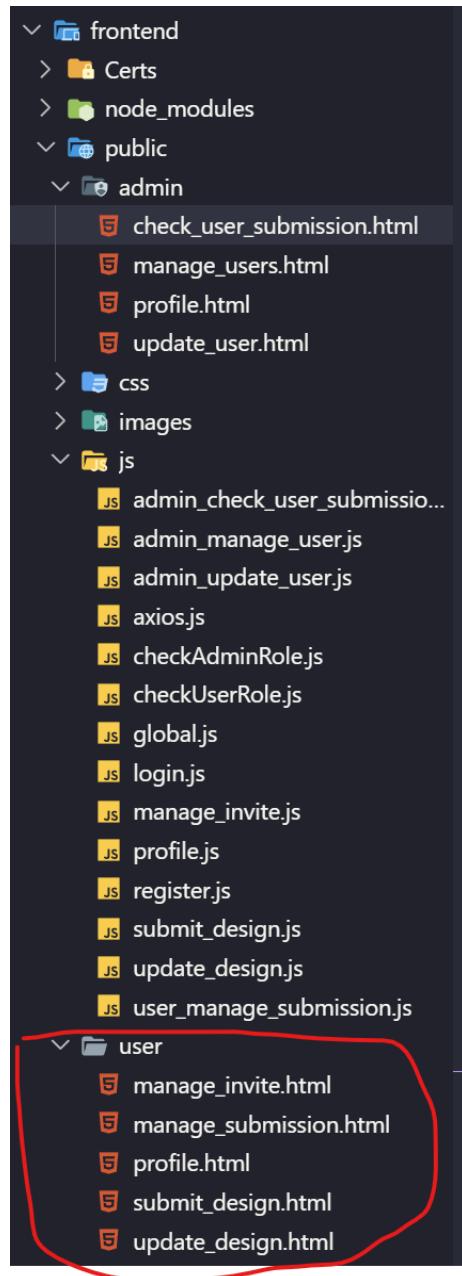
*Note: Do remember to add a defer to allow the page to fully load before calling the function.

```
routes.js      js checkAdminRole.js      js checkUserRole.js      js checkUserFnSolution.js      js submit_design.js      check_user_submission.html      js
frontend > public > admin > check_user_submission.html > head
1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5      <title>B.D.S</title>
6      <!-- Font Awesome-->
7      <link rel="stylesheet" href="https://use.fontawesome.com/releases/v6.2.0/css/all.css">
8      <!-- Google Fonts-->
9      <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Roboto:300,400,500,700&display=swap">
10     <!-- Bootstrap core CSS-->
11     <link href="https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/4.5.0/css/bootstrap.min.css" rel="stylesheet">
12     <!-- Material Design Bootstrap-->
13     <link href="https://cdnjs.cloudflare.com/ajax/libs/mdb-ui-kit/5.0.0/mdb.dark.min.css" rel="stylesheet"/>
14     <!-- Noty CSS-->
15     <link href="https://cdnjs.cloudflare.com/ajax/libs/noty/3.1.4/noty.css" rel="stylesheet">
16     <!-- Noty CSS Map-->
17     <link href="https://cdnjs.cloudflare.com/ajax/libs/noty/3.1.4/noty.css.map" rel="stylesheet">
18     <!-- Noty CSS Map-->
19     <link href="/css/site.css" rel="stylesheet">
20     <!-- Check if role is valid to prevent broken authentication -->
21     <script src="/js/checkAdminRole.js" defer></script> ↙
22
23 </head>
```

Step 12:

Next, let us go to the user html pages and add a code to call the checkUserRole.js file that we created just now for each of the respective user role pages.

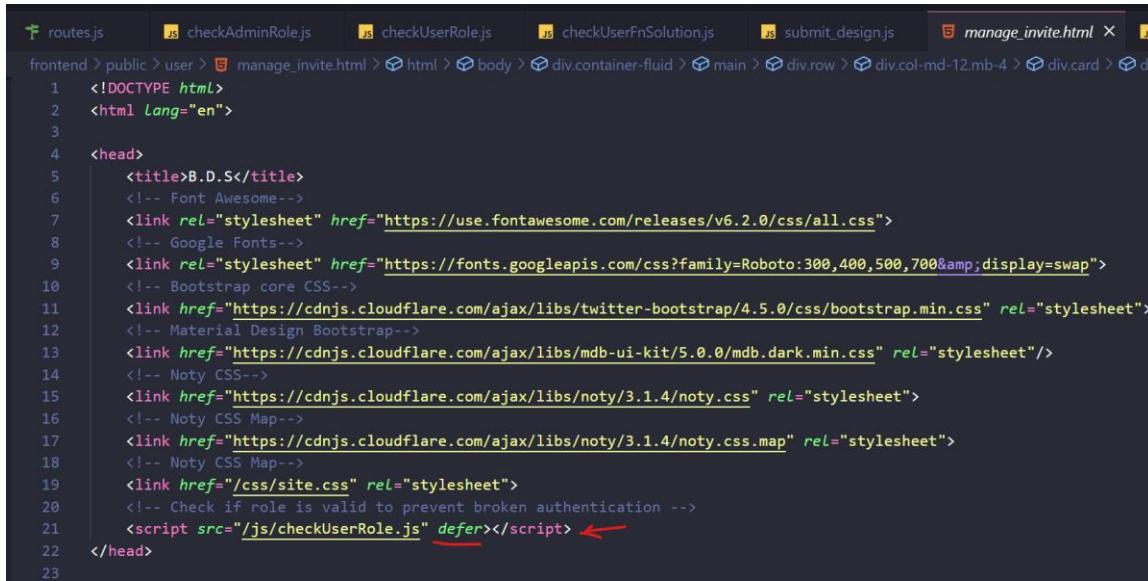
Under /frontend/public/user/



Step 13:

Let us **add the following code shown below to the above 5 user html pages in the head tag** to prevent anyone unauthenticated to access the user pages.

*Note: Do remember to add a defer to allow the page to fully load before calling the function.

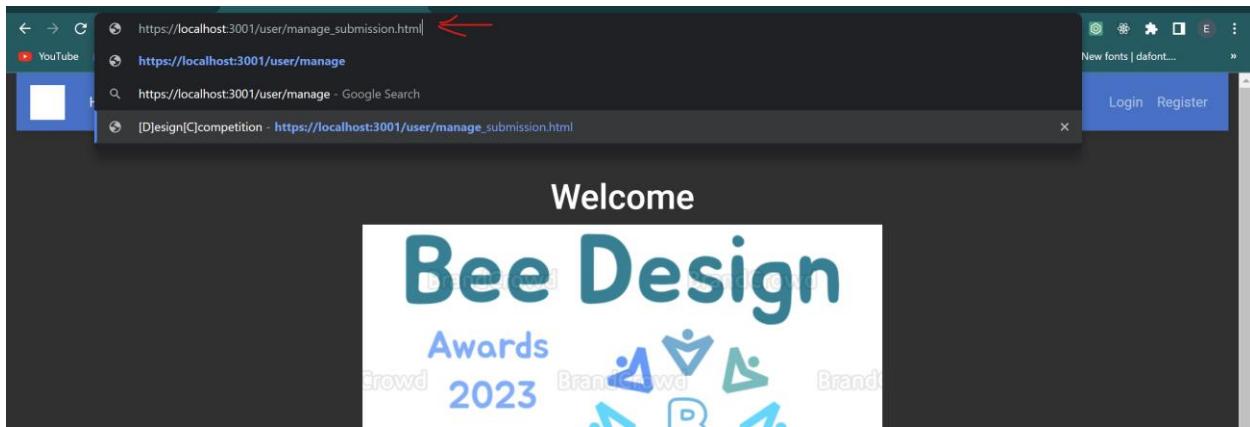


```
routes.js      JS checkAdminRole.js      JS checkUserRole.js      JS checkUserFnSolution.js      JS submit_design.js      manage_invite.html X JS
frontend > public > user > manage_invite.html > html > body > div.container-fluid > main > div.row > div.col-md-12.mb-4 > div.card > div
1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5      <title>B.D.S</title>
6      <!-- Font Awesome-->
7      <link rel="stylesheet" href="https://use.fontawesome.com/releases/v6.2.0/css/all.css">
8      <!-- Google Fonts-->
9      <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Roboto:300,400,500,700&amp;display=swap">
10     <!-- Bootstrap core CSS-->
11     <link href="https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/4.5.0/css/bootstrap.min.css" rel="stylesheet">
12     <!-- Material Design Bootstrap-->
13     <link href="https://cdnjs.cloudflare.com/ajax/libs/mdb-ui-kit/5.0.0/mdb.dark.min.css" rel="stylesheet"/>
14     <!-- Noty CSS-->
15     <link href="https://cdnjs.cloudflare.com/ajax/libs/noty/3.1.4/noty.css" rel="stylesheet">
16     <!-- Noty CSS Map-->
17     <link href="https://cdnjs.cloudflare.com/ajax/libs/noty/3.1.4/noty.css.map" rel="stylesheet">
18     <!-- Noty CSS Map-->
19     <link href="/css/site.css" rel="stylesheet">
20     <!-- Check if role is valid to prevent broken authentication -->
21     <script src="/js/checkUserRole.js" defer></script> ←
22  </head>
23
```

7.2.3 Evidence that the vulnerability has been rectified - About URL Link access

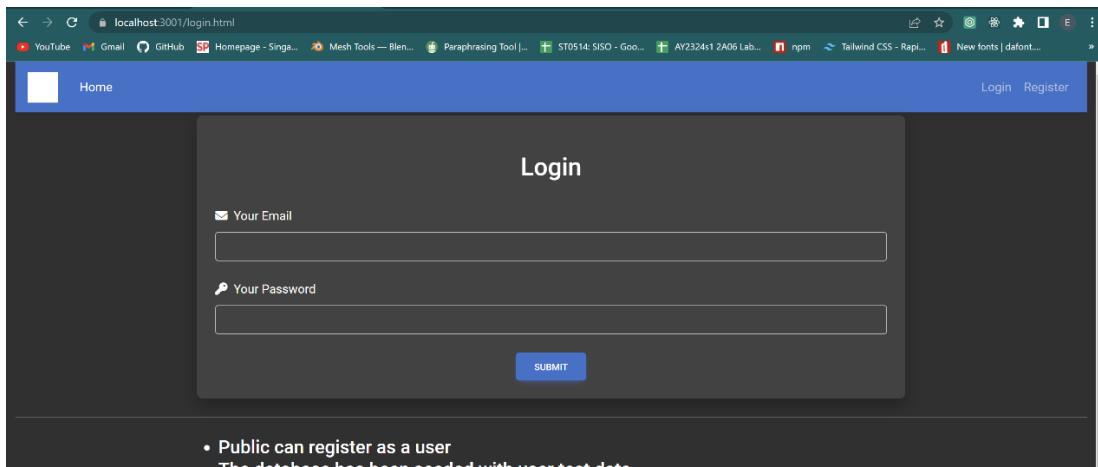
Step 1:

Now, let us check if the above changes have fixed the vulnerability. First, let us go into the webpage and paste the following into the URL link in the Welcome Page.



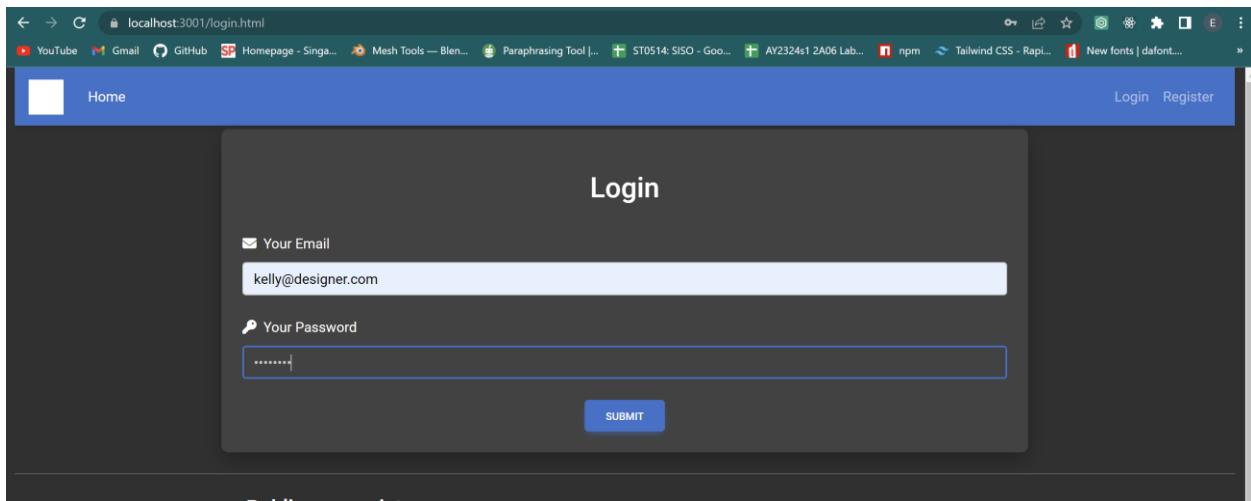
Step 2:

You should be able to see that the site will redirect you to the login page if we try to access the user page without logging in.



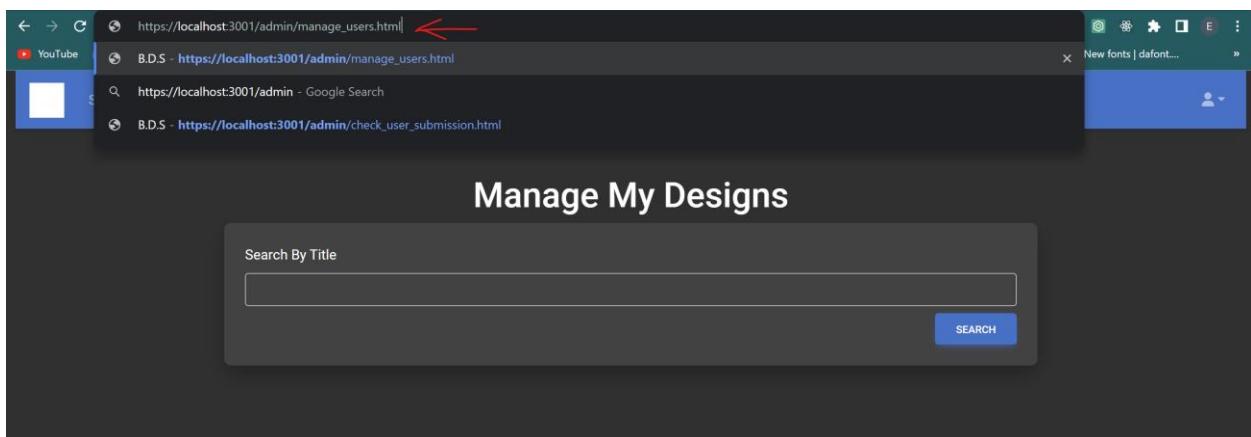
Step 3:

Let us now try to login using Kelly's Email and Password



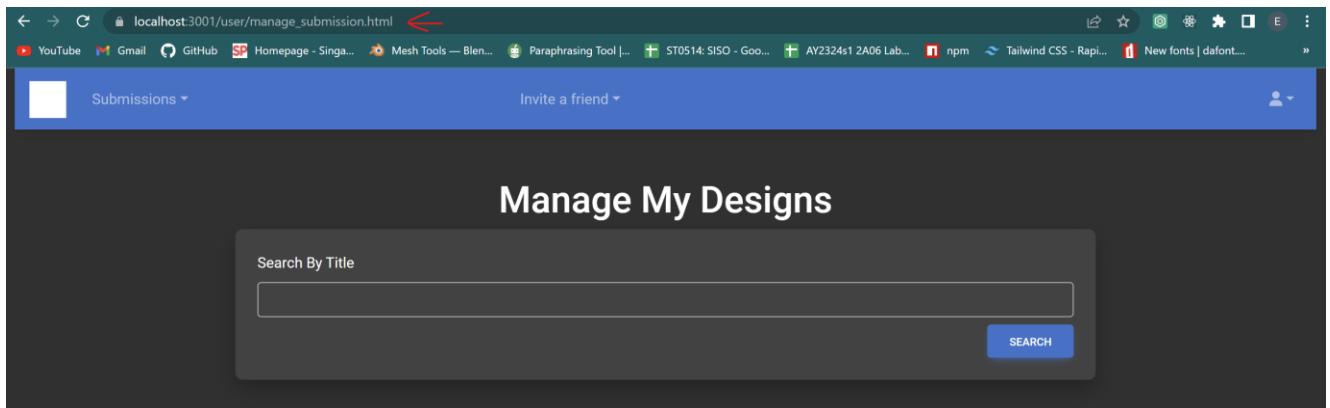
Step 4:

After logging in, let us enter the admin page URL to see if we can access the admin page even though our role is user.



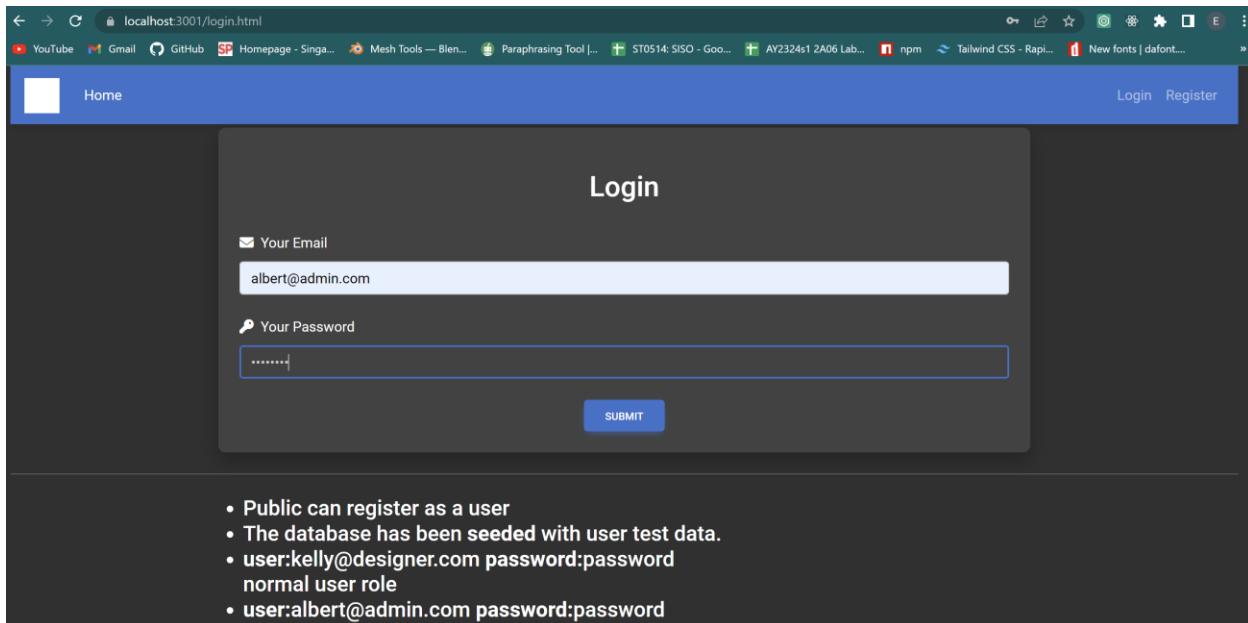
Step 5:

You should see that you are redirected back to the user page as Kelly is not an admin.



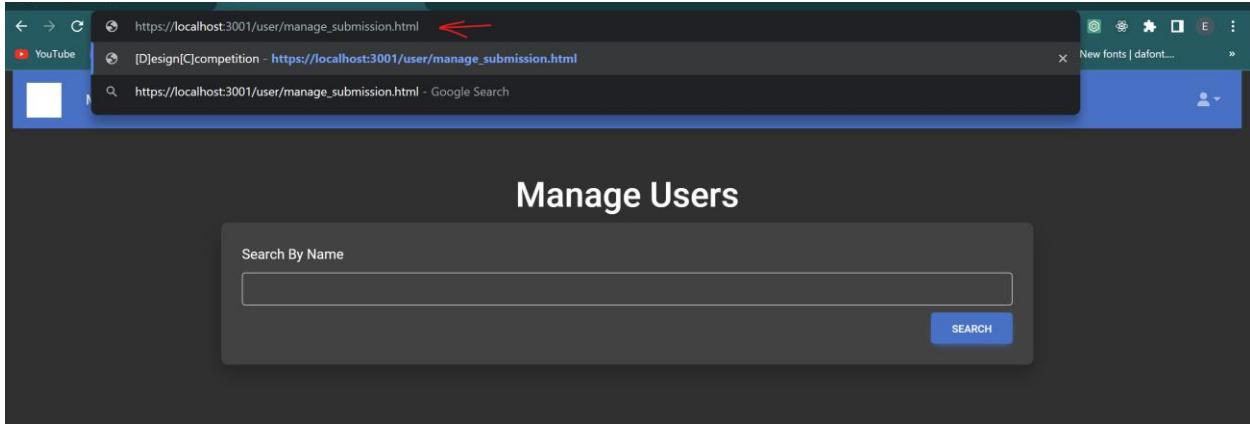
Step 6:

Next, let us try the same testing with an admin account this time. First, log out from Kelly's Account and Login using Albert's Account.



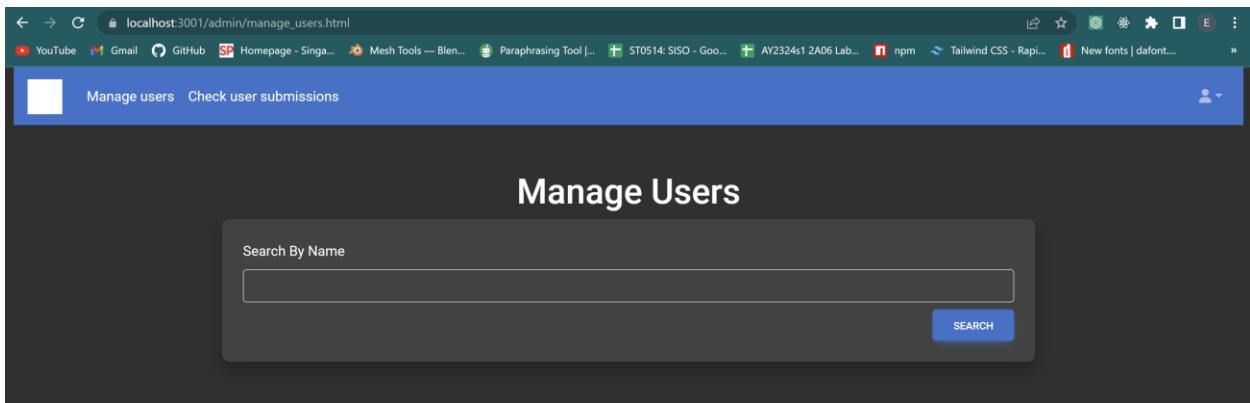
Step 7:

We will now change the above link to a user page URL and see if we can access the user page.



Step 8:

You should also be able to see that you have been redirected back to the admin page as albert is an admin role and not a user role.



7.3.0 Code Review – Preventing Future Vulnerabilities

Error handling and Nested Callbacks

The application contains many different way to resolve the errors, where some endpoints handle the errors immediately.

A few examples are shown below:

Figure A:

```
55
56     exports.processRegister = (req, res, next) => {
57       console.log('processRegister running.');
58       let fullName = req.body.fullName;
59       let email = req.body.email;
60       let password = req.body.password;
61
62
63       bcrypt.hash(password, 10, async(err, hash) => {
64         if (err) {
65           console.log('Error on hashing password');
66           return res.status(500).json({ statusMessage: 'Unable to complete registration' });
67         } else {
68
69           results = user.createUser(fullName, email, hash, function(results, error){
70             if (results!=null){
71               console.log(results);
72               return res.status(200).json({ statusMessage: 'Completed registration.' });
73             }
74             if (error) {
75               console.log('processRegister method : callback error block section is running.');
76               console.log(error, '=====');
77               return res.status(500).json({ statusMessage: 'Unable to complete registration' });
78             }
79           });
80         } //End of anonymous callback function
81       });
82     });
83   };
84
85
86
87   });
88 });
89
90
91 }; // End of processRegister
```

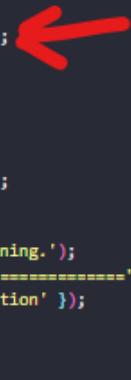


Figure B:

```
9  exports.processLogin = (req, res, next) => {
10
11    let email = req.body.email;
12    let password = req.body.password;
13    try [
14      auth.authenticate(email, function(error, results) {
15        if (error) {
16          let message = 'Credentials are not valid.';
17          //return res.status(500).json({ message: message });
18          //If the following statement replaces the above statement
19          //to return a JSON response to the client, the SQLMap or
20          //any attacker (who relies on the error) will be very happy
21          //because they relies a lot on SQL error for designing how to do
22          //attack and anticipate how much "rewards" after the effort.
23          //Rewards such as sabotage (seriously damage the data in database),
24          //data theft (grab and sell).
25          return res.status(500).json({ message: error });
26
27      } else {
28        if (results.length == 1) {
29          if ((password == null) || (results[0] == null)) {
30            return res.status(500).json({ message: 'login failed' });
31          }
32          if (bcrypt.compareSync(password, results[0].user_password) == true) {
33
34            let data = {
35              user_id: results[0].user_id,
36              role_name: results[0].role_name,
37              token: jwt.sign({ id: results[0].user_id }, config.JWTKey, {
38                expiresIn: 86400 //Expires in 24 hrs
39              })
40            }; //End of data variable setup
41
42            return res.status(200).json(data);
43          } else {
44            // return res.status(500).json({ message: 'Login has failed.' });
45            return res.status(500).json({ message: error });
46          } //End of password comparison with the retrieved decoded password.
47        } //End of checking if there are returned SQL results
48
49      }
50    });
51  });
52
53  ] catch (error) {
54    return res.status(500).json({ message: error });
55  } //end of try
56
57
58
59 ];
```

The examples above are what we usually call "callback hell" where multiple callbacks are nested on top of each other. This can get even more unreadable when more features are added.

So, the bottom 2 images that rectifies processLogin and processRegister would be the way to go to avoid any confusion and callback hell by using try and catch for all authController and userController.js.

processLogin:

```
9  exports.processLogin = async (req, res, next) => {
10    let email = req.body.email;
11    let password = req.body.password;
12    try {
13      var results = await auth.authenticate(email);
14      if (results.length == 1) {
15        if (password == null || results[0] == null) {
16          throw "Login has failed.";
17        }
18        if (!bcrypt.compareSync(password, results[0].user_password) == true)
19          getIPAddress()
20            .then((ip) => {
21              logger.error({
22                userAgent: req.userAgent.source,
23                method: req.method,
24                route: req.originalUrl,
25                email: req.body.email,
26                ip: ip,
27                error: "Password does not match",
28                status: 401,
29              });
30            })
31            .catch((error) => {
32              console.log(error);
33            });
34          throw "Invalid login credentials. Please try again.";
35        } else {
36          let data = {
37            user_id: results[0].user_id,
38            role_name: results[0].role_name,
39            token: jwt.sign(
40              { id: results[0].user_id, role: results[0].role_name },
41              config.JWTKey,
42              {
43                expiresIn: 86400, //Expires in 24 hrs
44              }
45            ),
46          }; //End of data variable setup
47          return res.status(200).json(data);
48        }
49      }
50    } catch (error) {
51      if (error == "Invalid login credentials. Please try again.") {
52        let message = "Invalid login credentials. Please try again.";
53        return res.status(500).json({ message: message });
54      } else {
55        console.log(error);
56        let message = "Login has failed.";
57        return res.status(500).json({ message: message });
58      }
59    }
60  }; //End of processLogin
61
```

processRegister:

```
136  exports.processRegister = (req, res, next) => {
137    console.log("processRegister running.");
138    let fullName = req.body.fullName;
139    let email = req.body.email;
140    let password = req.body.password;
141
142    bcrypt.hash(password, 10, async (error, hash) => {
143      try {
144        if (error) {
145          console.log("Error on hashing password");
146          throw "Error on hashing password.";
147        } else {
148          var results = await user.createUser(fullName, email, hash);
149          if (results != null) {
150            console.log(results);
151            let message = "Completed registration.";
152            return res.status(200).json({ message: message });
153          }
154        }
155      } catch (error) {
156        console.log(error);
157        if (error.errno == 1062) {
158          let message = "Email already exists.";
159          return res.status(500).json({ message: message });
160        } else {
161          let message = "Unable to complete registration.";
162          return res.status(500).json({ message: message });
163        }
164      }
165    });
166  }; //End of processRegister
167
```

Standardizing the JSON Response Data

In the application, there are different types of ways used to send back a JSON Response.

In the example shown below, you can see that there is a declared variable message, but it was not used rather the message sends the error instead.

Figure C:

```
59 exports.processGetSubmissionData = async (req, res, next) => {
60   let pageNumber = req.params.pagenumber;
61   let search = req.params.search;
62   let userId = req.body.userId;
63   try {
64     let results = await fileDataManager.getFileData(userId, pageNumber, search);
65     console.log(
66       "Inspect result variable inside processGetSubmissionData code\n",
67       results
68     );
69     if (results) {
70       var jsonResult = {
71         number_of_records: results[0].length,
72         page_number: pageNumber,
73         filedatal: results[0],
74         total_number_of_records: results[2][0].total_records,
75       };
76       return res.status(200).json(jsonResult);
77     }
78   } catch (error) {
79     let message = "Server is unable to process your request."; ←
80     return res.status(500).json({
81       message: error,
82     });
83   }
84 }; //End of processGetSubmissionData
```

On the other hand, there is also another JSON Response where it uses a string in the message instead of using a variable shown in Figure D.

Figure D:

```
172 exports.processUpdateOneUser = async (req, res, next) => {
173   console.log("processUpdateOneUser running");
174   //Collect data from the request body
175   let recordId = req.body.recordId;
176   let newRoleId = req.body.roleId;
177   try {
178     results = await userManager.updateUser(recordId, newRoleId);
179     console.log(results);
180     return res.status(200).json({ message: "Completed update" });
181   } catch (error) {
182     console.log(
183       "processUpdateOneUser method : catch block section code is running"
184     );
185     console.log(
186       error,
187       "=====";
188     );
189     return res
190       .status(500)
191       .json({ message: "Unable to complete update operation" });
192   }
193 }; //End of processUpdateOneUser
194
```

On the other hand, in Figure E the JSON Response uses statusMessage while in Figure F it uses message instead of statusMessage which can cause some confusion for future developers as shown below.

Figure E:

```
51 exports.processRegister = (req, res, next) => {
52   console.log('processRegister running.');
53   let fullName = req.body.fullName;
54   let email = req.body.email;
55   let password = req.body.password;
56
57
58   bcrypt.hash(password, 10, async(err, hash) => {
59     if (err) {
60       console.log('Error on hashing password');
61       return res.status(500).json({ statusMessage: 'Unable to complete registration' });
62     } else {
63
64       results = user.createUser(fullName, email, hash, function(results, error){
65         if (results!=null){
66           console.log(results);
67           return res.status(200).json({ statusMessage: 'Completed registration.' });
68         }
69         if (error) {
70           console.log('processRegister method : callback error block section is running.');
71           console.log(error, "=====");
72           return res.status(500).json({ statusMessage: 'Unable to complete registration' });
73         }
74       });
75     }
76   });
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91 }; // End of processRegister
92
```

Figure F:

```
9  exports.processLogin = (req, res, next) => {
10
11    let email = req.body.email;
12    let password = req.body.password;
13    try {
14      auth.authenticate(email, function(error, results) {
15        if (error) {
16          let message = 'Credentials are not valid.';
17          //return res.status(500).json({ message: message });
18          //If the following statement replaces the above statement
19          //to return a JSON response to the client, the SQLMap or
20          //any attacker (who relies on the error) will be very happy
21          //because they relies a lot on SQL error for designing how to do
22          //attack and anticipate how much "rewards" after the effort.
23          //Rewards such as sabotage (seriously damage the data in database),
24          //data theft (grab and sell).
25          return res.status(500).json({ message: error });
26
27      } else {
28        if (results.length == 1) {
29          if ((password == null) || (results[0] == null)) {
30            return res.status(500).json({ message: 'login failed' });
31          }
32          if (bcrypt.compareSync(password, results[0].user_password) == true) {
33
34            let data = {
35              user_id: results[0].user_id,
36              role_name: results[0].role_name,
37              token: jwt.sign({ id: results[0].user_id }, config.JWTKey, {
38                expiresIn: 86400 //Expires in 24 hrs
39              })
40            }; //End of data variable setup
41
42            return res.status(200).json(data);
43          } else {
44            // return res.status(500).json({ message: 'Login has failed.' });
45            return res.status(500).json({ message: error });
46          } //End of password comparison with the retrieved decoded password.
47        } //End of checking if there are returned SQL results
48
49      }
50
51    })
52
53  } catch (error) {
54    return res.status(500).json({ message: error });
55  } //end of try
56
57
58
59};
```

So, to standardize the JSON Response, we will be using the following format for all the JSON Response in userController.js file as shown below.

```
        }
    } catch (error) {
        let message = "Server is unable to process your request.";
        return res.status(500).json({
            error: error,
            message: message,
        });
}
```

Summary of bonus parts including those not shown above and rectifications that are not shown in the above 6 vulnerabilities

1. Done Input Validation for all input fields like register new user, search inputs, email invitation, design file type submission etc.
2. Done Output Sanitization for User Data E.g., Full name, Email and Role name displayed in Admin Page and User Page.
3. To prevent easy password guessing against broken authentication I did a weak password check where the password must Contain at least 1 upper case, 1 lower case, 1 number, 1 special character with at least 7 characters.
4. Changed the cloudinary files in database to https instead of http by changing the *seeddata.js* file and the cloudinary upload image (*uploadFile* Function) in *fileService.js* by changing *result.url* to *result.secure_url* to remove the https warnings for future and current submissions.
5. Used Bearer Token for all and not using user Id and role for user access control and authentication.
6. Used Session-Storage instead of Local-Storage to delete token after user has closed the tab/browser instead of continuing to store it.
7. Used express-rate-limit to limit the number of attempts for register page, login page, email invitation page and design submission page to prevent brute forcing.

8. Created a check role from token to make sure that only user and admin can access their own sites without being able to access a web page that they don't have access to through the URL Link.
9. Used Try and Catch for all functions in authController.js and userController.js to handle all errors at the end to prevent callback hell.
10. Standardized fileService.js, authService.js and userService.js to use Promise and declare SQL statement before the Promise.
11. Used Standardized JSON Response for authController.js and userController.js.

References

- <https://www.liquidweb.com/kb/how-to-install-chocolatey-on-windows/>
- <https://technixleo.com/create-locally-trusted-ssl-certificates-with-mkcert-on-windows/>
- <https://blog.logrocket.com/rate-limiting-node-js/>