

Secure Programming For Web

(When Security Meets Paranoia)

M.Sc. Cybersecurity
School of Computing
National College of Ireland
Dublin, Ireland

Mentor - Irina Tal

Saptarshi Laha - x18170081



1. Executive Summary

À La Codé is a platform for sharing and getting code peer-reviewed. It was designed with the National College of Ireland's Secure Programming for Web curriculum in mind. The website enables users to create projects, review code, provide feedback, and much more. The privilege levels of the site are divided into four categories:

1. **Owner + Admin** – Person creating a project has the highest privileges in that project.
2. **Admin** – Has almost all the privileges as the owner except certain exclusive ones.
3. **Reviewer** – The primary responsibility of this role is to review code and provide constructive feedback.
4. **Developer** – The lowest privilege level in the project, responsible for writing code.

The functionalities added to the project take into consideration and completely prevent not only the OWASP top 10 vulnerabilities but also additional faults such as logic flaws, unauthenticated sessions, etc. while having a reliable performance as discussed in the later sections in detail. The idea behind this project was to not only allow future students of National College of Ireland to have an easy to use system to get their code review classwork and homework done by their peers and professors, but also provide a classic example of a secure website which is fail-safe against penetration testing as it employs the best secure programming practices.

2. Background

2.1 Project Description

This project enables multiple users to create projects, share them with others, invite others to help them in their projects, get code peer-reviewed in a 4 tier architecture model. It consists of multiple internal PHP files to handle the processes. CodeMirror has been used to extend syntax highlighting for over 120 different languages. Exclusive locking is used while saving files as concurrent writes to the same data would otherwise result in broken text. There is no separate admin panel as such, but the logic of the program dynamically assigns different users the role of Owner and Admin when they create a project. They then further have the capability to invite others to the project and manage them according to the needs of the project. PHPMailer in SMTP mode is used for the forgot password form. Further details regarding the security mechanisms and additional functionalities in place are mentioned in the appropriate sections.

2.2 Choice of Programming Languages

This project uses PHP as a server-side programming language and MySQL as its database. This choice was made based on the popularity of the Apache Server in today's date and the ever-increasing functionality of the open-source PHP language, such as supporting the latest cryptographic algorithms from the libsodium library to allow us to perform Argon2ID password hashing, etc. Apart from this, my initial idea was to also host it on an actual web server for use by future National College of Ireland students pursuing the module. Since Apache Server hosting is the most common form of web hosting available, I decided to stick to PHP as my backend language.

The front end uses HTML5, CSS, Javascript, Bootstrap to provide functionality. This choice was made as HTML5 and CSS are the de-facto standards for Hypertext Markup. Javascript was used as it was a language natural to learn for someone who has previous experience in Java programming. Bootstrap was used to provide additional, visually pleasing aspects to the website.

MySQL was chosen as the database as it is a structured database and better than an unstructured database like MongoDB. Other options were not chosen as I was not experienced with them. Furthermore, it was preinstalled as a part of the XAMPP application which I used to run my project

after making changes, which allowed for easy debugging of the application or queries if something went wrong.

2.3 Hardware Requirements

The requirements to run the website are close to nil. Any basic system, even a Chromebook or a low powered mobile device, can run it (although the visuals are not optimized for mobile devices). In terms of the server needed to run the backend code, the requirement for memory is low and a maximum of 1 gigabyte is enough to run the application, however, as the data increases, the storage space needed will increase as well and thus it can act as a limiting factor in case the hosting provider is unable to provide enough disk space in terms of database storage and file storage. The model of this application allows an individual to opt for the shared hosting initially and scale up in terms of disk space upon the need for the same. Hence, the business model used for the implementation of the project was well thought out before starting to develop the same.

3. Requirements

3.1 Security Requirements

Since I tried to follow Microsoft's SSDLC model, I took into consideration the potential attacks that could probably occur on the website even before simulating them or trying them out. This methodology helped me plan a robust internal system to fend off any potential attacker. Common vulnerabilities such as SQL Injection, Clickjacking, XSS, Remote Code Execution, Cross-Site Request Forgery were completely prevented. Additionally, few more security implementations were used to combat additional vulnerabilities and a robust internal logging system is in place by Apache to log every access and error for diagnosis at a later stage in case of a potential breach. Defense-in-depth and principle of least privilege have been taken very seriously throughout the entirety of the project. Multiple layers of checks have been performed before allowing the user to achieve the same while trying to be on par or as close to competitive webpage load and execution speeds as possible. The fail-safe principle is also taken seriously as it is essential for the webpage not to fail once in the production stage.

3.2 Business Requirement

The business requirement of the web-application was to allow multiple users to create accounts, type code, save the code, get it reviewed, all while being present in a secure environment. The business logic describes the person who creates the project is the owner, as well as the admin of the project. The owner can create, delete, edit and review everyone's files apart from managing members by inviting them to the project, removing them from the project, promoting or demoting them. Additionally, the owner can delete the entire project, thereby eliminating everyone from the project and deleting all files within the same. An admin has equal rights apart from the fact that an admin cannot promote another member to become an admin, only the owner has the permission to do that. An admin also can't delete the owner's files or remove the owner from the project or delete the project. Furthermore, an admin cannot remove another admin from the group. A reviewer is an exceptional role who is assigned to review code and accept it or reject it based on defined standards in place. The reviewer can further comment on as to why they feel something was accepted or rejected. They can also create their files and provide example code to their fellow peers from which they can learn and improve. The last role within the project is the developer. This user can create their files and get them reviewed by members higher up in the hierarchy. An additional feature is that every member can delete, read or edit their files. However, they cannot review their files unless they are the owner. Some additional functionalities such as messaging, providing alerts on invitations or removal from projects, forgot password mechanism, full-screen code editing with syntax highlighting,

etc. have also been provided for ease of use and to increase the overall usability and functionality of the application.

3.3 Functional Requirements

The functional requirements of the application highlight the primary elements needed to be fulfilled by it. This includes but, is not limited to logging in only on entering the correct username and password. Linking the projects with the right people. Giving the right people privileges in a project. Sending messages to the right person. Sending alerts to the right people and much more. The security requirements often overlap with functional needs such as in this case. An unauthenticated user should not be allowed to access anything. This is both a security and a functional requirement.

The basic functional requirements for this project can be summarized as follow:

1. Have a login/signup and forgot-password page for the respective activities.
2. Have an option to create projects and add files and members to the project.
3. Have an option to communicate with members and personally.
4. Have a logout and change-password mechanism.
5. Have a mechanism to delete projects.
6. Have a mechanism to change user details.

3.4 Non-functional Requirements

Requirements that don't fall under the functional requirements category but are equally as important as the previously mentioned requirements are listed here. Usability, accessibility, and performance are major factors when it comes to serious website development. I have personally taken into consideration every factor needed to assure a mix of all three of these ingredients in their correct proportions while incorporating tank tier security. A few examples are listed below:

1. **Usability/Accessibility** – Using google reCAPTCHA v3, which uses score-based analysis for human verification rather than asking the user to type in details present in an image or click images to verify.

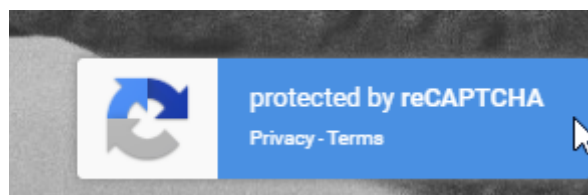


Fig. 1 - Google reCAPTCHA v3

2. **Performance** – Performance is a huge factor when it comes to website development. I have made sure that the pages load on an average within 4 seconds even after performing complexing operations on the background. This speed can further be increased by allocating more processing power and memory, but I wanted to set the default parameters for testing to not acquire unrealistic numbers. Most of the elements end up loading within 2.56 seconds on an average with the captcha API calls taking a while longer to respond, clocking the overall average of scripts at 3.89 seconds. This can further be increased by downloading the resource instead of loading them every time. While this is a good approach, the same was not implemented as the links tend to have the latest version of the scripts. This helps reduce vulnerabilities and from the older version and reduces the effort to replace the scripts on every release.

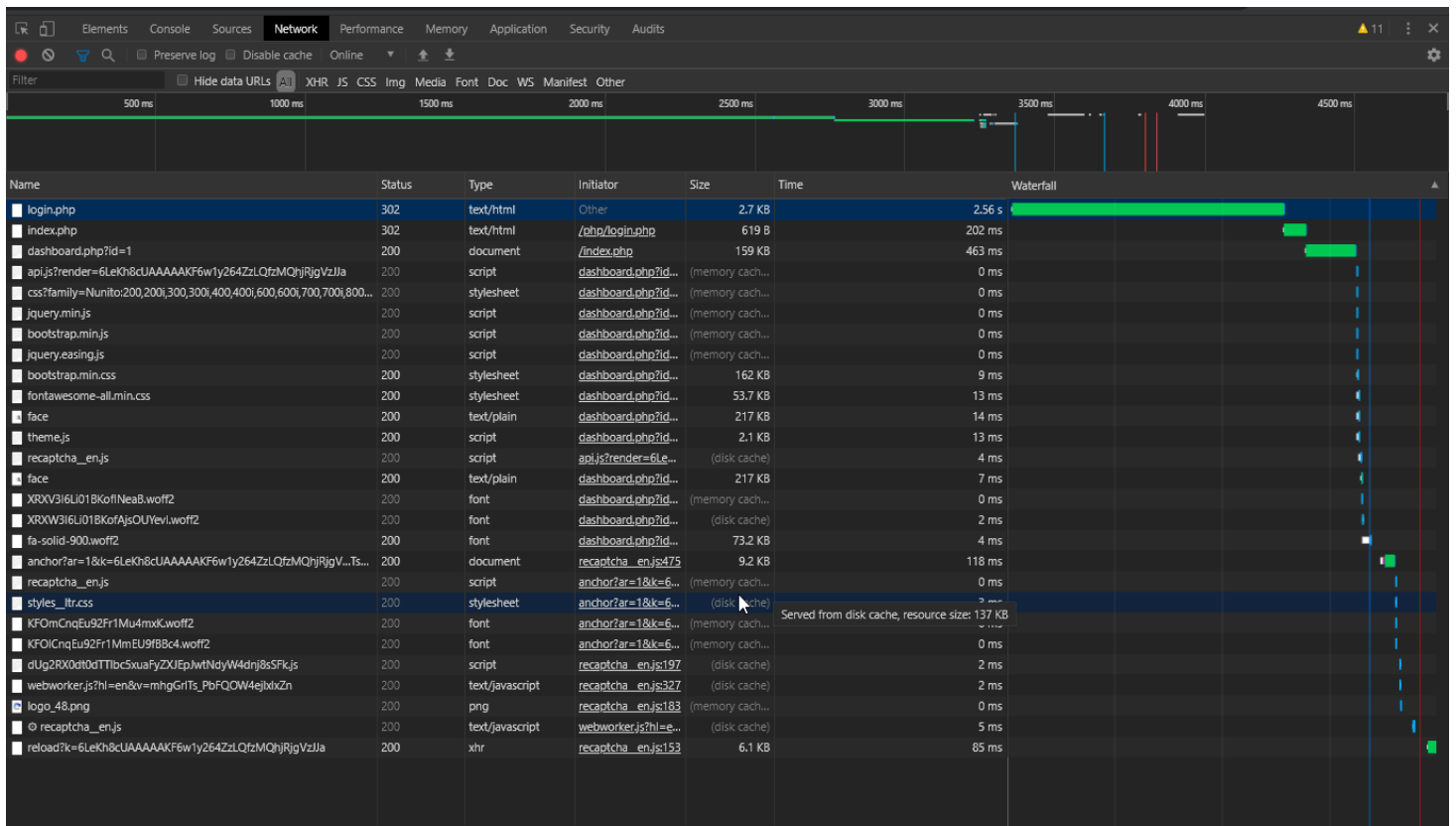


Fig. 2 – Page Load performance (Google Chrome)

4. Design Implementation

4.1 Overall Design Model

I followed an iterative design model that encompasses requirement analysis, architecture design, implementation in code, security testing, functional testing, and non-functional testing. The deployment phase was left out till the very end as I was incorporating a modular approach where files have their own set out individual actions to carry out. A pictorial description of the process is shown below.

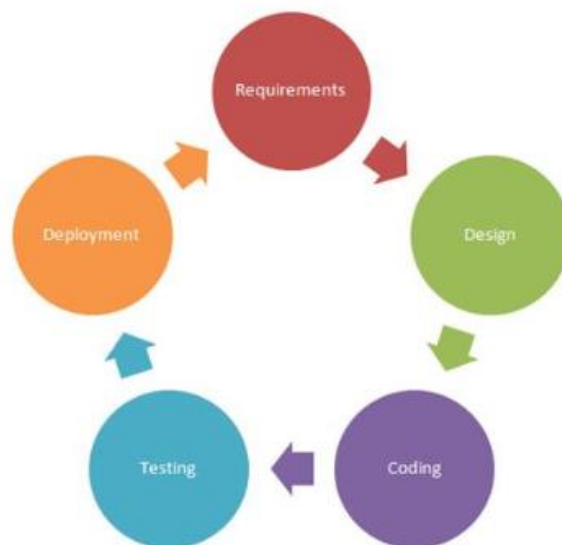


Fig. 3 – Software Development Life Cycle

The above methodology was extended to perform security analysis at every phase by performing modular security testing. Every module was tested for security flaws after their initial design and implementation. The security factors were kept in mind while analyzing the requirements and creating the layout as well.

4.2 Use Case Analysis

Use case analysis is a mandatory requirement to develop any complete software package. It highlights the significant aspects of the application that need to be implemented and helps distinguish between permitted and restricted activities. The use case diagrams for my project are listed below.

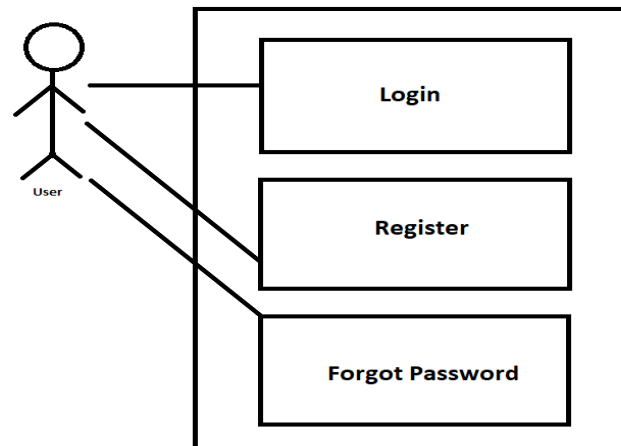


Fig. 4 – Base cases on visiting the website

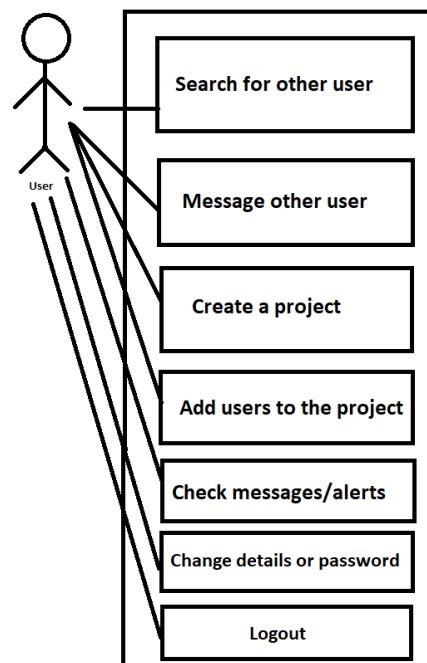


Fig. 5 – Base cases on logging in

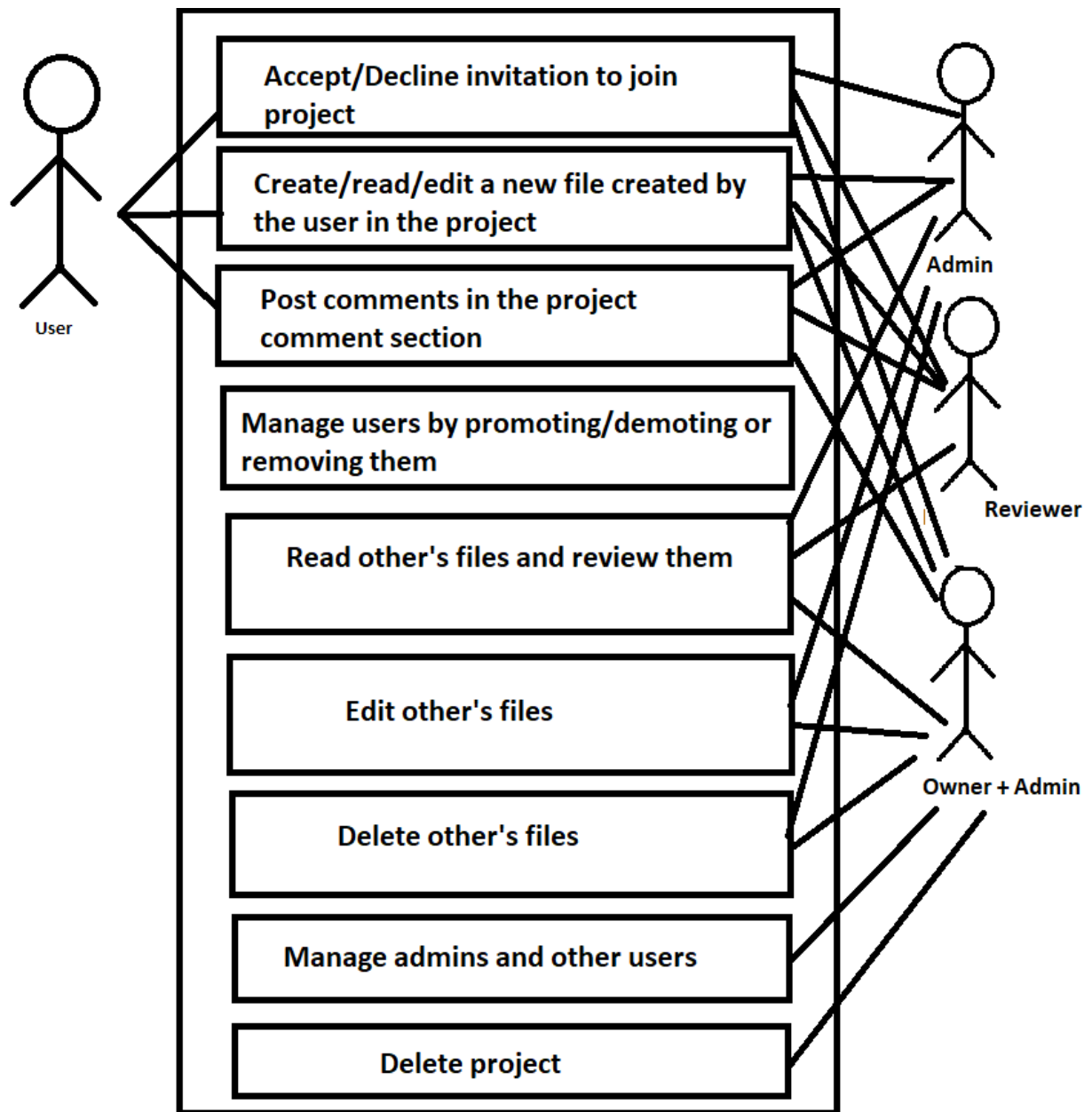


Fig. 6 – Project Cases

In the project cases, the creator of the project is the owner, as well as the admin. He can further make other users admin or reviewer. By default, every new person joining the project is a user or a developer. Hence the roles are dynamically allocated based on the project. A person can be an admin in one of the projects while being just a developer in another. The privilege levels are not set in stone and hence there is complex logic on the backend to manage the same and prevent unauthorized activity.

4.3 Abuse Case Analysis

A mandatory aspect of secure software development is analyzing the attack surface and preparing strategies to counteract any attacks. When starting this journey, I analyzed every possible attack vector that could potentially maliciously interact with my project and perform unintended operations. After this, I devised a robust defensive strategy. The abuse cases mentioned on the left are a small part of the actions that I was anticipating my website to be facing. The corresponding defense mechanisms implemented to fend off the same are also mentioned in the diagram below. It is not only essential to perform such an analysis from the security perspective but also vital to have the same for the general testing phase as well to try out corner cases and apply mandatory patches necessary if the program breaks or performs unintended operations.

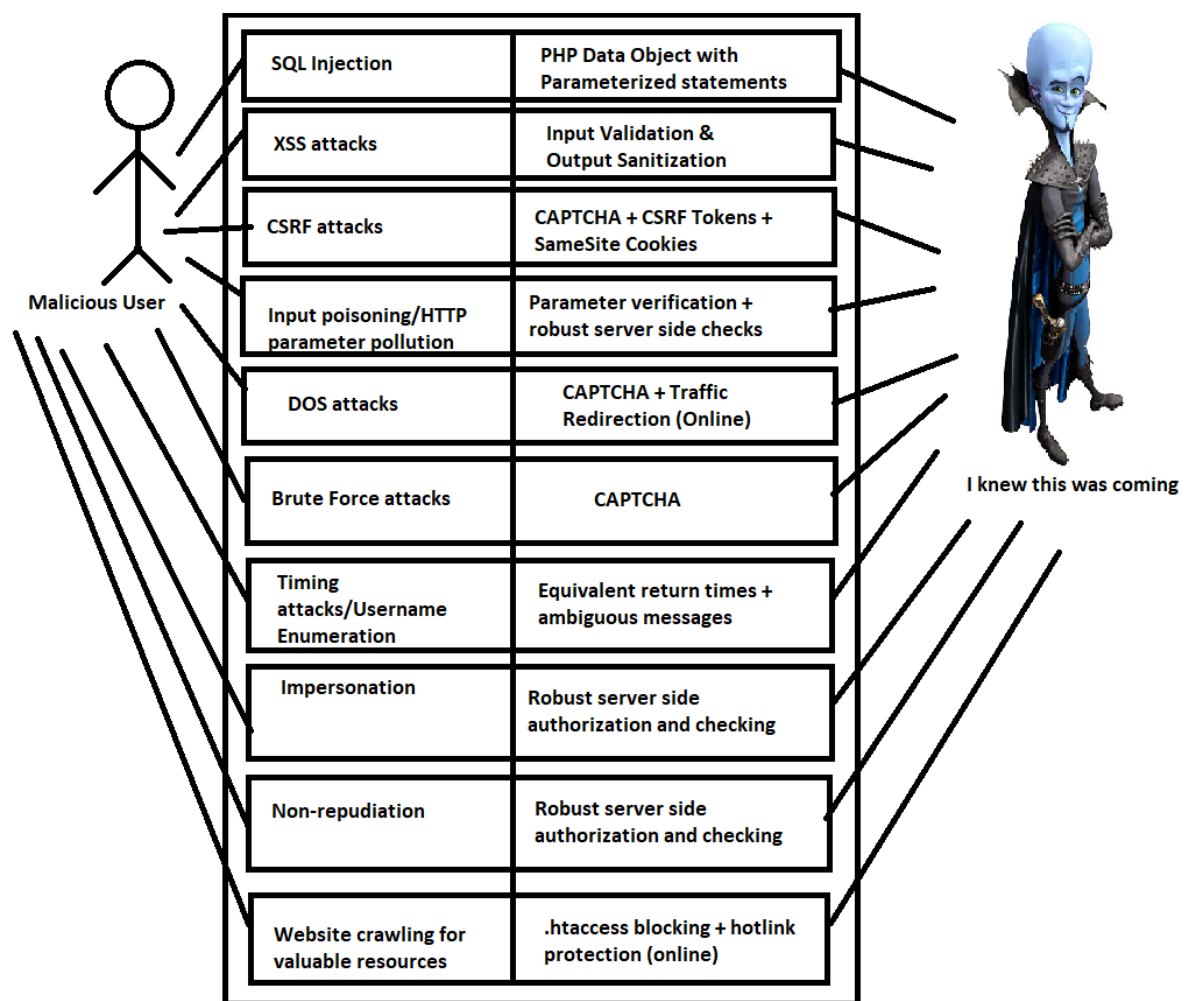


Fig. 7 – Abuse Case Analysis

5. Implementation of Code

It is close to impossible to post all the code that's present in the project due to the sheer size of the project I created and the number of dependencies it needed for performing multiple operations.

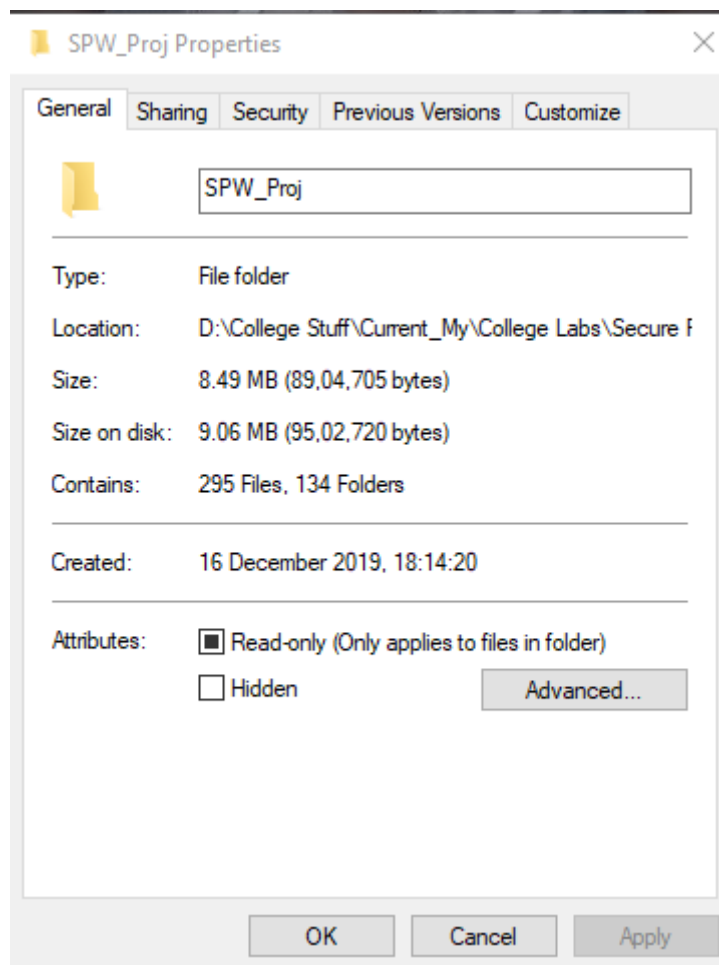


Fig. 8 – Project Size

However, the basic layout of the project is as follows.

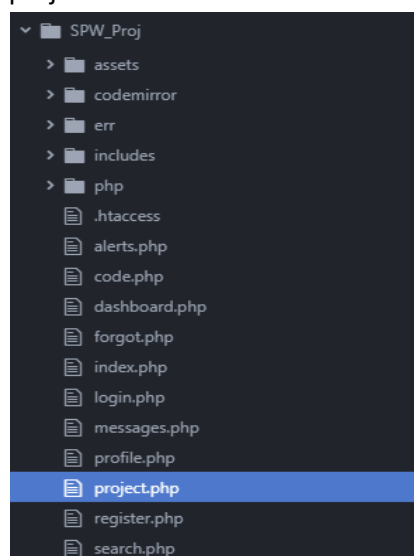


Fig. 9 – Project Structure

The critical features incorporated for security are discussed in the security implementations section along with the respective code. Apart from that, the PHP folder contains PHP scripts that execute when they receive a request from the outer PHP pages. Some of them also include class definitions or perform other specialized functions. The err folder contains all the error documents that are mentioned in the .htaccess files such as error 404 for file not found or error 403 for access forbidden. The codemirror folder has the javascript implementation of the codemirror text editor that I use to allow syntax highlighting of 120+ different languages along with full-screen view. The includes folder contains common HTML elements that are present in all pages to reduce code duplication and making the project more secure by reducing the attack surface. The assets folder consists of bootstrap and CSS files apart from a few javascript files which are used to improve the responsiveness of the website.

6. Database Implementation

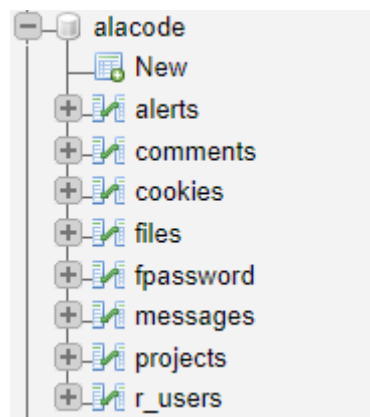


Fig. 10 – Database Structure

The alacode database is divided into eight tables. The alerts table is responsible for storing alerts that are generated by a sender and received by a receiver. Here the Sender_ID and Receiver_ID fields are foreign keys that link to the ID column of the r_users table.

ID	Sender_ID	Receiver_ID	New_Alert	Alert
32	2	1	0	Your project dadadad is being actively contributed...
33	1	1	0	The project dadadad has been deleted by its creato...
34	1	2	0	The project dadadad has been deleted by its creato...
36	1	1	0	The project dadad has been deleted by its creator ...
38	1	1	0	The project dadada has been deleted by its creator...
39	1	1	0	The project adada has been deleted by its creator ...
40	1	1	0	The project djiaiodjoadi has been deleted by its c...

Fig. 11 – Alerts Table

The New_Alert signifies a new alert and Alert column contains the message present inside the alert.

ID	Sender_ID	Creator_ID	Comment	Project_Name
----	-----------	------------	---------	--------------

Fig. 12 – Comments Table

This table is responsible for holding all the comments made by different users in different projects. Here the Sender_ID and Creator_ID fields are foreign keys that link to the r_users ID column. The Project_Name refers to the project in which they have commented. The Comment field is used to store the comment of the user. Here the Creator_ID refers to the creator of the project.

ID	Cookie	User_ID
39	4399017ba2e83de3a2806327abaaa34034b38458f1be64b528...	1
40	0f824a3ef2335034a65cfa6880d6b06150570fc9467be548fd...	1

Fig. 13 – Cookie Table

This table is responsible for storing the hashed values of the randomly generated user cookies when they login to the website. The User_ID is a foreign key field that link to the ID column of r_users table. The cookies are stored in a hashed format as even if someone manages to breach the database, the cookies will still be secure.

ID	Member_ID	Creator_ID	Filename	Project_Name	Reviewed
----	-----------	------------	----------	--------------	----------

Fig. 14 – Files Table

This table is responsible for storing all the files of all the users in all the projects. It distinguishes files based on the Member_ID, Creator_ID and Project_Name fields. Here Member_ID and Creator_ID are the two foreign keys that link to the r_users table in the ID column. The Filename is the name of the file created by the user. Reviewed is a flag that is set if a reviewer or an admin has reviewed and accepted the file.

ID	Reset_ID	Email	Token	Valid
13	1	saptarshi.laha@gmail.com	82b395528d32b71f55eb23f7ee3f4dc94b5799cbf374b87c1e...	1576563592
15	1	saptarshi.laha@gmail.com	d145b6b21e0f87fdaa9b8c25775f59b25186409e8a006e6f9c...	1576564045

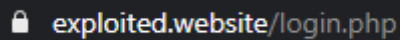
Fig. 15 – fpassword Table

This table holds the forgot password token in hashed format apart from the Email of the user and validity period. By default, all tokens are valid for 60 minutes. Reset_ID is a foreign key that links the r_users ID column to the fpassword table. If a link with an expired token is clicked, the password reset mechanism isn't executed.

8. Security Measures Implemented

Most of the security measures implemented are mentioned below:

1. **HTTPS** – Secure HTTP. Data is not transferred in plaintext and it provides confidentiality and integrity of data. A self-signed SSL certificate needed to be installed.



```
RewriteEngine On
RewriteCond %{HTTPS} !=on
RewriteRule ^(.*)$ https://%{HTTP_HOST}%{REQUEST_URI} [L,R=301,NE]
```

Fig. 19 – HTTPS Implementation

2. **HSTS** – HTTP Strict Transport Security which enforces HTTPS and uses a preload list to never establish a HTTP connection with the domains on the list.

```
Header set Strict-Transport-Security "max-age=31536000; includeSubDomains; preload"
```

Fig. 20 – HSTS Implementation

3. **Security Headers.**

```
<IfModule mod_headers.c>
Header set Cache-Control "no-cache, no-store, must-revalidate"
Header set Pragma "no-cache"
Header set Expires 0
Header set X-XSS-Protection "1; mode=block"
Header always append X-Frame-Options "deny"
Header set X-Content-Type-Options "nosniff"
Header set Strict-Transport-Security "max-age=31536000; includeSubDomains; preload"
Header set Expect-CT "max-age=7776000, enforce"
Header set Referrer-Policy "origin-when-cross-origin"
</IfModule>
```

Fig. 21 – Security Headers

These headers prevent caching, protect against rendering **XSS attacks** (google chrome browser), **clickjacking and frame embedding**, **MIME type sniffing**, **referrer policy when originating cross origin request** and **certificate transparency**. Additionally, **HTTP Public Key Pinning** header is also used in the online version.

4. **CSRF protection** – It is prevented in multiple ways in the project, such as by making all cookies samesite:strict, using a captcha and also a randomly generated CSRF token.

```
Header edit Set-Cookie ^(.*)$ $1;HttpOnly;Secure;SameSite=strict
```

```
<input type='hidden' name='__csrf_magic' value='sid:c6d71c6175d35b6f8e040be716fd9f1e2c71402c,1576623637' />
```

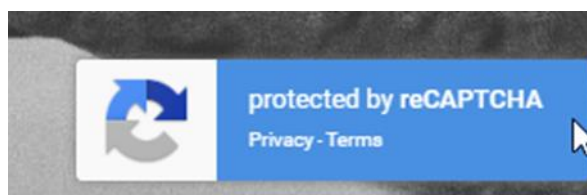


Fig. 22 – CSRF Protections

- SQL Injection Prevention** – It was prevented by using the PHP data object version of parameterized statements along with binding of variables. I defined a class for it with a public static function. Every database query used this function.

```
private static function connect(){
    include('creds.php');
    $pdo = new PDO('mysql:host=127.0.0.1;dbname=alacode;charset=utf8', $unroot, $pwroot);
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    return $pdo;
}

public static function query($query, $parameters = array()){
    self::setupDBTable();
    $statement = self::connect()->prepare($query);
    $statement->execute($parameters);

    if(explode(' ', $query)[0] == 'SELECT'){
        $data = $statement->fetchAll();
        return $data;
    }
}
```

Fig. 23 – SQL Injection Prevention

- 6. XSS Prevention** – I used input validation and output sanitization to achieve this.

```
if(Database::query('SELECT Username from r_users WHERE Username=:username', array(':username'=>$username)) && preg_match('/^(?=.{5,32}$)(?![_.])(?!.*[_.]2)[a-zA-Z0-9_]+(?<[_.]$)/', $username) && strlen($username) >= 5 && strlen($username) <= 32){
    if(preg_match('~(?:[p{L}]p{Mn})p{Pd}\x{2019}+s2)+$~u', $fname) && preg_match('~(?:[p{L}]p{Mn})p{Pd}\x{2019}+s2)+$~u', $lname) && strlen($fname) >= 2 && strlen($lname) >= 2 && strlen($fname) <= 30 && strlen($lname) <= 30){
        if(filter_var($email, FILTER_VALIDATE_EMAIL) && strlen($email) >= 5 && strlen($email) <= 50 && !Database::query('SELECT Email from r_users WHERE Email=:email', array(':email'=>$email))){
            if(strlen($contact) >= 8 && strlen($contact) <= 10 && preg_match('/^[0-9]*$/ ', $contact) && !Database::query('SELECT Contact_Number from r_users WHERE Contact_Number=:contactnum', array(':contactnum'=>$contact))){
                if(strlen($address) >= 10 && strlen($address) <= 500 && preg_match('/^[#0-9a-zA-Z\s,-]*$/ ', $address)){
                    if(strlen($password1) >= 12 && $password1==password2 && strlen($password1) <= 500 && preg_match("(?!^(?=.*[a-Z])(?=.*[a-z]){1})(?=(.*[\\d]){1})(?=(.*[\\W]){1})(?!.*s).{12,500}$/", $password1)){
                        $htmlspecialchars("<div>./users/U_</div>".$username_current."<div>/face</div>").<div>".$username_current.</div>";
                    }
                }
            }
        }
    }
}
```

Fig. 24 – Example input validation logic for registration and input sanitization for output.

- 7. Multi-layered Authentication Logic** – Used this to validate the user before letting him perform any sensitive operation.

```

if (checkLoggedIn.isLoggedin()){
    $userId = checkLoggedIn.isLoggedin();

    if(isset($_POST['captcha'])){
        $captcha = Captcha::createRequest($_POST['captcha']);
        if($captcha[0] == true && $captcha[1] >= 0.7){
            if(isset($_POST['P']) && isset($_POST['DE']) && isset($_POST['RE']) && isset($_POST['REA']) && isset($_POST['RER']) && isset($_POST['R']) && isset($_POST['E']) && isset($_POST['D']) && isset($_POST['projid'])){
                $cid = Database::query('SELECT Creator_ID FROM projects WHERE ID = :pid', array(':pid'=>$_POST['projid']));
                if(count($cid) == 1){
                    $cid = Database::query('SELECT Creator_ID FROM projects WHERE ID = :pid', array(':pid'=>$_POST['projid']))[0]['Creator_ID'];
                    $pname = Database::query('SELECT Project_Name FROM projects WHERE ID = :pid', array(':pid'=>$_POST['projid']))[0]['Project_Name'];
                    $result = Database::query('SELECT * FROM projects WHERE Creator_ID = :pname1', array(':cid1'=>$cid, ':pname1'=>$pname));
                    $role1 = Database::query('SELECT * FROM projects WHERE Creator_ID = :cid1 AND Project_Name = :pname1 AND Member_ID = :mid', array(':cid1'=>$cid, ':pname1'=>$pname, ':mid'=>$userId));
                    $role2 = Database::query('SELECT * FROM projects WHERE Creator_ID = :cid1 AND Project_Name = :pname1 AND ID = :mid', array(':cid1'=>$cid, ':pname1'=>$pname, ':mid'=>$_POST['P']));
                    if(count($role2) == 1 && $role2[0]['Member_ID'] != $userId && $_POST['P'] != $cid && $role1[0]['role'] == -1 && $role2[0]['role'] == 1 && $userId != $cid){
                        Database::query('UPDATE projects SET Role = :x WHERE Creator_ID = :cid1 AND Project_Name = :pname1 AND ID = :mid', array(':x'=>($role2[0]['role'] - 1), ':cid1'=>$cid, ':pname1'=>$pname, ':mid'=>$_POST['P']));
                        header("Location: ../project.php?id=".$_POST['projid']);
                    }
                } else if(count($role2) == 1 && $role2[0]['Member_ID'] != $userId && $userId == $cid && $role2[0]['role'] == 0 || $role2[0]['role'] == 1){
                    Database::query('UPDATE projects SET Role = :x WHERE Creator_ID = :cid1 AND Project_Name = :pname1 AND ID = :mid', array(':x'=>($role2[0]['role'] - 1), ':cid1'=>$cid, ':pname1'=>$pname, ':mid'=>$_POST['P']));
                    header("Location: ../project.php?id=".$_POST['projid']);
                }
            } else{
                header("Location: ../project.php?id=".$_POST['projid']);
            }
        }
    }
} else{
    header("Location: ../index.php");
}

```

Fig. 25 – Multi-layered Authentication Logic

8. **Logging** – I have made use of Apache’s inbuilt logging interface to acquire access logs as well as error logs. I have also tried using monolog and database logging, but both are extremely resource heavy and reduces the performance of the application.

9. Testing

It is an important aspect of any project. It is used to analyze the reliability of an application when under stress or even otherwise. In this case, I did perform both security and general testing each time I build a new module. The results of static testing in terms of security are reported by SonarQube and VisualCodeGrepper as follows.

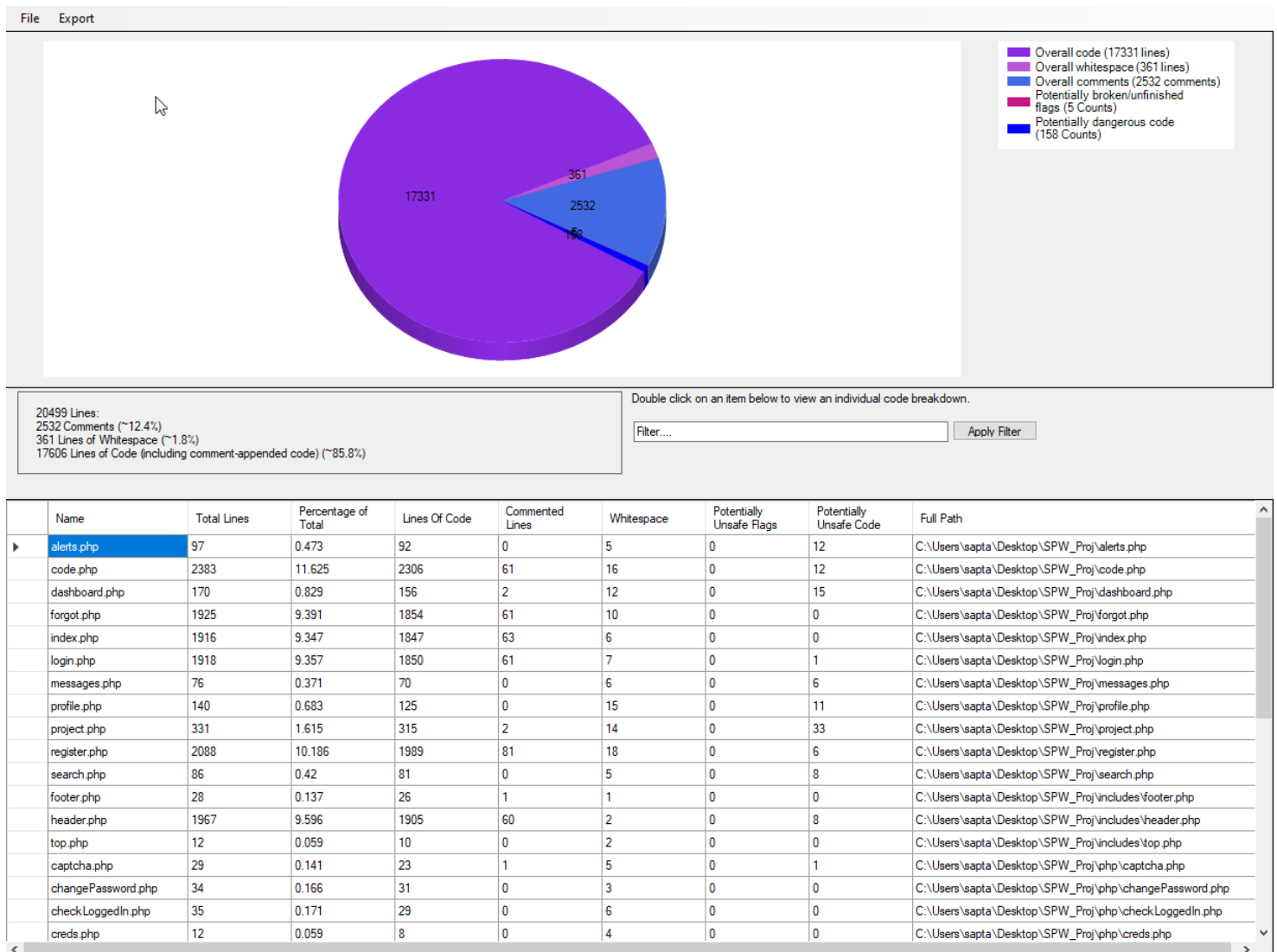


Fig. 26 – Visual Code Grepper Output

It reported a lot of issues but helped me fix a true XSS vulnerability. All the rest are false positives such as shown below.

MEDIUM: Potentially Unsafe Code - Potential XSS
 Line: 61 - C:\Users\sapta\Desktop\SPW_Proj\messages.php
 The application appears to reflect data to the screen with no apparent validation or sanitisation. It was not clear if this variable is controlled by the user.

```
echo '<td style="text-align:center;">'.htmlspecialchars($results2[$i]['Message']).</td>';
```

Fig. 27 – False positives as it’s an old tool and doesn’t recognize htmlspecialchars function.

SonarQube gave a much better output in terms of vulnerability analysis.

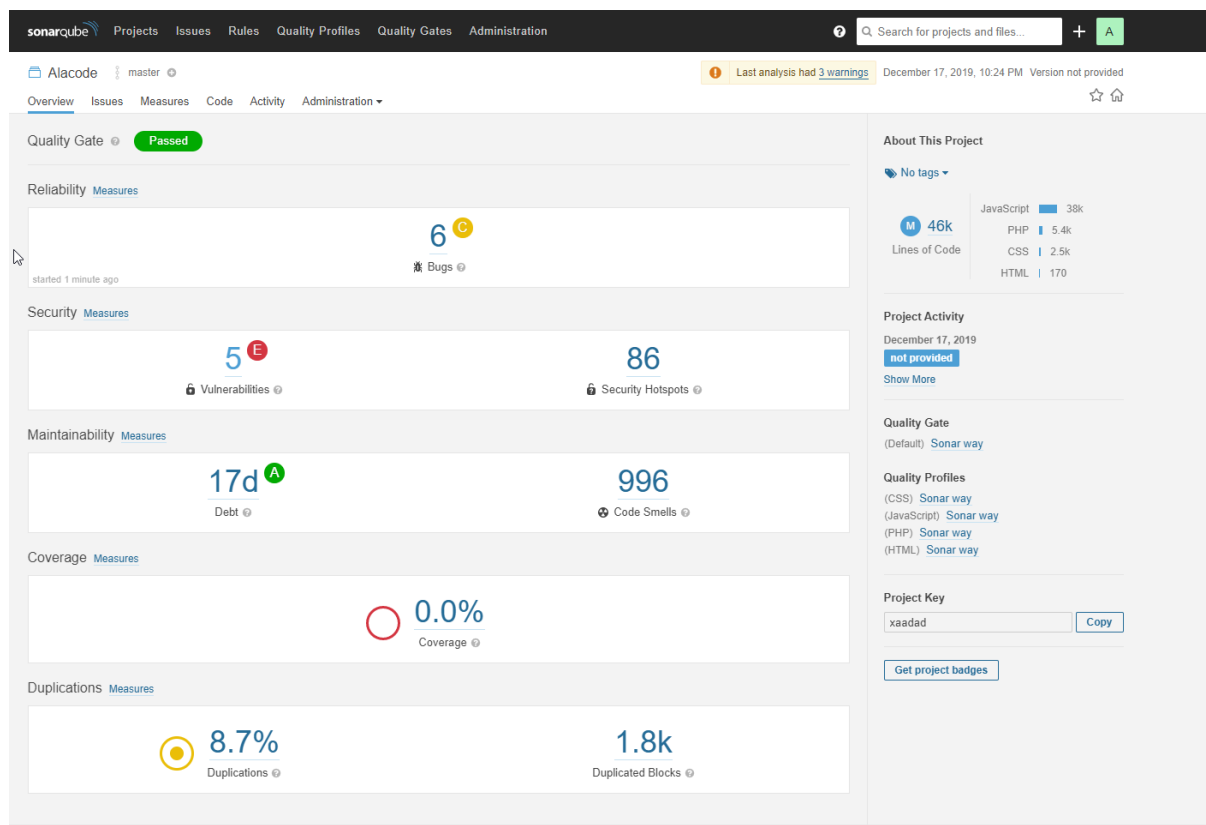


Fig. 28 – SonarQube Output

However, the bugs and vulnerabilities caught by SonarQube refer to hardcoded credentials which do not exist in either of the 2:

1. Limited reach of user.
2. Does not exist at all.

```
Database::query('UPDATE r_users SET Password=:password WHERE ID=:userid', array(':password'=>password_hash($_POST['newpass'], PASSWORD_ARGON2ID, ['memory_cost' => 2048, 'time_cost' => 50, 'threads' => 50]), 'userid'=>$id);
```

Fig. 28 – SonarQube False Positives

Active analysis was performed by BURP suite which could not come up with any potential vulnerabilities as shown below.

#	Task	Time	Action	Issue type	Host	Path
22	3	23:47:34 17 Dec 2019	Issue found	Path-relative style sheet import	https://exploited.we...	/php/login.php
21	3	23:47:32 17 Dec 2019	Issue found	Path-relative style sheet import	https://exploited.we...	/php/register.php
20	3	23:47:15 17 Dec 2019	Issue found	Path-relative style sheet import	https://exploited.we...	/php/forgot.php
19	3	23:47:03 17 Dec 2019	Issue found	Path-relative style sheet import	https://exploited.we...	/register.php
18	3	23:46:47 17 Dec 2019	Issue found	Path-relative style sheet import	https://exploited.we...	/login.php
17	3	23:46:46 17 Dec 2019	Issue found	Path-relative style sheet import	https://exploited.we...	/index.php
16	3	23:46:40 17 Dec 2019	Issue found	Path-relative style sheet import	https://exploited.we...	/forgot.php
15	3	23:46:09 17 Dec 2019	Issue found	HTTP TRACE method is enabled	http://exploited.web...	/
14	3	23:46:08 17 Dec 2019	Issue found	SSL certificate	https://exploited.we...	/
13	3	23:46:08 17 Dec 2019	Issue found	HTTP TRACE method is enabled	https://exploited.we...	/
12	3	23:45:25 17 Dec 2019	Issue found	Cross-domain script include	https://exploited.we...	/php/forgot.php
11	3	23:45:25 17 Dec 2019	Issue found	Cross-domain script include	https://exploited.we...	/php/login.php
10	3	23:45:25 17 Dec 2019	Issue found	Cross-domain script include	https://exploited.we...	/php/register.php
9	3	23:45:25 17 Dec 2019	Issue found	Password field with autocomplete enabled	https://exploited.we...	/php/register.php
8	3	23:45:24 17 Dec 2019	Issue found	Cross-domain script include	https://exploited.we...	/robots.txt
7	3	23:45:24 17 Dec 2019	Issue found	Password field with autocomplete enabled	https://exploited.we...	/php/login.php
6	3	23:45:24 17 Dec 2019	Issue found	Cross-domain script include	https://exploited.we...	/login.php
5	3	23:45:24 17 Dec 2019	Issue found	Password field with autocomplete enabled	https://exploited.we...	/php/forgot.php
4	3	23:45:24 17 Dec 2019	Issue found	Cross-domain script include	https://exploited.we...	/forgot.php
3	3	23:45:24 17 Dec 2019	Issue found	Cross-domain script include	https://exploited.we...	/index.php
2	3	23:45:24 17 Dec 2019	Issue found	Cross-domain script include	https://exploited.we...	/register.php
1	3	23:45:24 17 Dec 2019	Issue found	Password field with autocomplete enabled	https://exploited.we...	/register.php

Fig. 29 – Burp shows no errors in active authenticated and unauthenticated scans.

10. Conclusion

Overall, after working hard on improving my programming practices and reducing code duplication, I reached about 1.8K lines of code which is a little more than 8% code duplication. Considering the time limit on the project, it was quite challenging while keeping every feature intact. The security considerations taken into account are the best standards that are being used in today's world. Some of them may be considered overkill, but it is mandatory to reduce the attack surface from the start as portrayed by this project. In conclusion, even though the time limit was short and the task at hand was huge, the feeling of accomplishment after completing the assignment in the best possible manner gives me ultimate pleasure. Given the chance to do it again, I would reconstruct the entire application in a modular fashion to further reduce code duplication and increase performance since security standards implemented are already the best currently available.