A dark blue vertical bar runs down the left side of the page. A blue arrow points to the right from this bar, containing the date.

10.05.2023

Dokumentation

Modularbeit M183

Several thin, curved lines in dark blue and light grey originate from the bottom left and sweep upwards and to the right.

Alperen Yilmaz

INFORMATIKMITTELSCHULE BASEL-STADT

Inhaltsverzeichnis

1. Einleitung	2
1.1 Ziel der Dokumentation	2
1.2 Überblick über die Applikation	2
1.3 Umfeld	2
2. Risikoanalyse	3
2.1 Broken Access Control	3
2.2 Cryptographic Failures	3
2.3 Injection	3
2.4 Security Misconfiguration	4
2.5 Verwendung von unsicheren Komponenten	4
2.6 Identifikations- und Authentifizierungsprobleme	4
2.7 Cross-Site Request Forgery	5
3. Risikobewertung	5
3.1 Injection	5
3.2 Broken Access Control	5
3.3 Identifikations- und Authentifizierungsprobleme	5
3.4 Cryptographic Failures	5
3.5 Security Misconfiguration	6
3.6 Verwendung von unsicheren Komponenten	6
3.7 Cross-Site Request Forgery	6
4. Risikomanagement	6
4.1 Verhinderung von Injection	6
4.1.1 Prepared Statements	6
4.1.2 Filterung und Validierung	7
4.2 Verhinderung von Broken Access Control	10
4.2.1 Zugriffskontrolle	10
4.3 Verhinderung von Identifizierungs- und Authentifizierungsproblemen	13
4.3.1 E-Mail-Verifizierung	13
4.3.2 Session-Hijacking	15
4.4 Verhinderung von "Cryptographic failures"	16
4.4.1 Password-Hashing	16
4.4.2 Tokenisierung	17
4.5 Verhinderung der "Verwendung von unsicheren Komponenten"	17
4.6 Verhinderung von Cross-Site Request Forgery	18
5. Weiteres	19
5.1 Impressum	19
6. Fazit	19

1. Einleitung

1.1 Ziel der Dokumentation

Diese Dokumentation dient einzig und allein für die Modularbeit M183. Es sollte den Lesern und Leserinnen ein Überblick über die Web-Applikation geben, hierbei fokussiert auf jegliche Risiken meiner Applikation und generell, wie mein "Risikomanagement" aussieht. Die Leser sollten schlussendlich wissen und verstehen, welche Risiken es gibt, welche in meinem Fall wie bewertet und auch wie verwaltet sind, warum bzw. warum nicht.

1.2 Überblick über die Applikation

Diese Applikation gibt den Benutzern die Möglichkeit, sich mit ihren Daten (wie etwa Benutzername, E-Mail und Passwort) zu registrieren. Um erfolgreich registriert zu werden, muss sich der Benutzer davor noch verifizieren. Dieses Verifizieren folgt direkt nach dem Eingeben der Daten. Via SMTP (dank der Library "PHPMailer") wird per E-Mail, welches der Benutzer davor eingegeben hat, ein 32-bit Token versendet, welchen er oder sie dann eingeben muss. Der Benutzer muss somit lediglich beweisen, dass er/sie auch Zugriff auf diese E-Mail hat. Somit wäre der Registrierungsprozess beendet und der Benutzer kann sich mit seinem Benutzernamen und mit seinem Passwort anmelden.

1.3 Umfeld

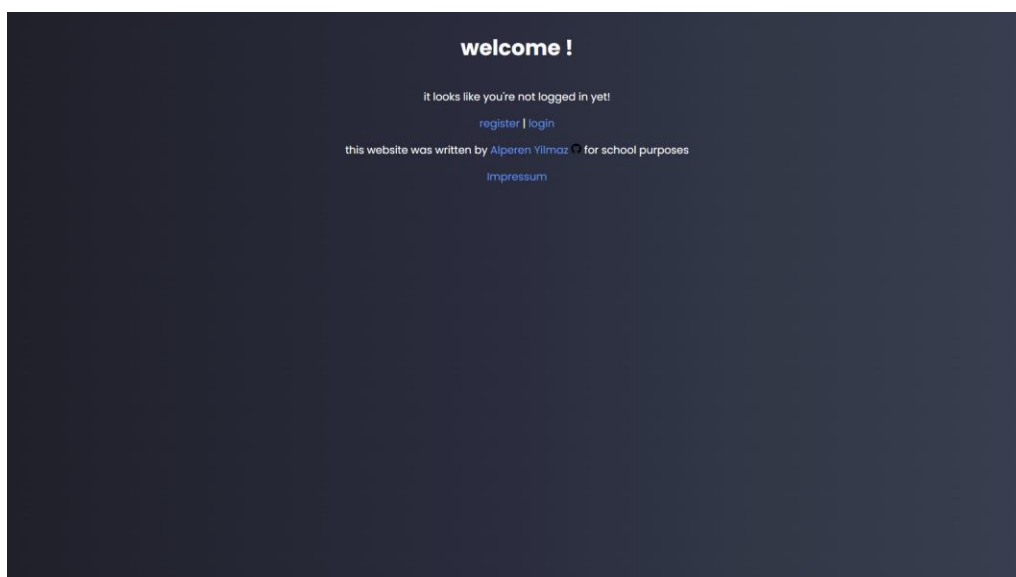
Programmiersprache: PHP 8.2

Datenbank: MySQL

Library: PHPMailer

IDE: PhpStorm

Anderes: "CodeSnap" wurde für die Screenshots von den Code-Snippets verwendet.



2. Risikoanalyse

In diesem Kapitel werden (potenzielle) Risiken angesprochen, welche Allgemein in Webapplikationen eine bestimmte Rolle spielen.

2.1 Broken Access Control

Wenn Authentifizierung und Zugriffsbeschränkung nicht ordnungsgemäss implementiert sind, ist es für Angreifer einfach, auf alles zuzugreifen. Bei Schwachstellen in der Zugriffskontrolle können nicht authentifizierte oder unbefugte Benutzer Zugang zu sensiblen Dateien und Systemen oder sogar zu Einstellungen für Benutzerrechte erhalten.

Ein Beispiel für eine Schwachstelle beim Access Control wäre eine Applikation, die den Zugriff auf bestimmte Funktionen nicht ordnungsgemäss auf der Grundlage der Rolle eines Benutzers einschränkt. Ein Administratorkonto könnte beispielsweise die Berechtigung haben, dem System neue Benutzer hinzuzufügen, ein normales Benutzerkonto jedoch nicht.

Schwache Zugriffskontrollen und Probleme bei der Verwaltung von Anmeldeinformationen lassen sich durch sicheres Best-Practice-Code sowie durch Präventivmassnahmen wie das Sperren von administrativen Konten und Kontrollen und die Verwendung von Multi-Faktor-Authentifizierung vermeiden.

2.2 Cryptographic Failures

APIs, die es Entwicklern ermöglichen, ihre Anwendungen mit Diensten von Drittanbietern wie Google Maps zu verbinden, sind eine grosse Zeitersparnis. Einige APIs basieren jedoch auf unsicheren Datenübertragungsmethoden, die Angreifer ausnutzen können, um Zugang zu Benutzernamen, Passwörtern und anderen vertraulichen Informationen zu erhalten.

Datenverschlüsselung/Hashing, Tokenisierung, ordnungsgemässe Tokenverwaltung und die Deaktivierung der Zwischenspeicherung von Responses können dazu beitragen, das Risiko der Publizierung sensibler Daten zu verringern.

2.3 Injection

Injection liegt vor, wenn ein Angreifer unsicheren Code ausnutzt, um seinen eigenen Code in ein Programm einzufügen (oder zu «injecten»). Da das Programm nicht in der Lage ist, den auf diese Weise eingefügten Code von seinem eigenen Code zu unterscheiden, können Angreifer Injektionsangriffe nutzen, um auf sichere Bereiche und vertrauliche Informationen zuzugreifen, als wären sie vertrauenswürdige Benutzer. Beispiele für Injections sind SQL-Injections, command injections, CRLF-Injections und LDAP-Injections.

Ebenfalls spielt Cross-Site Scripting (XSS) hierbei eine grosse Rolle. Beim Cross-Site-Scripting (XSS) machen sich Angreifer APIs und DOM-Manipulationen zunutze, um

Daten von Anwendungen abzurufen oder Befehle an sie zu senden. XSS vergrößert die Angriffsfläche für Bedrohungsakteure und ermöglicht es ihnen, Benutzerkonten zu kapern, auf Browserverläufe zuzugreifen, Trojaner und Würmer zu verbreiten, Browser aus der Ferne zu steuern und vieles mehr.

Die Schulung von Entwicklern in Best Practices wie Datenverschlüsselung und Validierung verringert die Wahrscheinlichkeit dieses Risikos. Deswegen sollte man Daten bereinigen, indem man überprüft, ob es sich um den Inhalt handelt, den man für das betreffende Feld erwartet, und indem man die Daten für den "Endpoint" verschlüsselt, um einen zusätzlichen Schutz zu bieten.

2.4 Security Misconfiguration

Genauso wie falsch konfigurierte Zugriffskontrollen sind auch allgemeinere Sicherheitskonfigurationsfehler ein grosses Risiko, das Angreifern schnellen und einfachen Zugang zu sensiblen Daten und Bereichen von Webseiten ermöglicht. Ein Beispiel dazu sind Dateien wie ``config.php`` welche Passwörter, API Keys etc. Beinhalten könnten, welche nicht auf dem Webserver und auch allgemein nicht auf eine Repository hochgeladen werden sollen, falls diese auch wirklich sensitive Daten enthalten.

Dynamische Tests können helfen, falsch konfigurierte Sicherheit in Ihrer Anwendung zu entdecken. Im Falle meines Projektes ist aber das manuelle Testen am einfachsten, jedoch zu bedenken ist, ist dass man somit dann auch wirklich jede «Ecke» selbst und eben manuell überprüfen muss.

2.5 Verwendung von unsicheren Komponenten

Unabhängig davon, wie sicher der eigene Code ist, können Angreifer APIs, dependencies und andere Komponenten von Drittanbietern ausnutzen, wenn diese selbst nicht sicher sind.

Die statische Analyse in Kombination mit der Softwarekompositionsanalyse kann unsichere Komponenten in der Anwendung erkennen und neutralisieren.

2.6 Identifikations- und Authentifizierungsprobleme

Identifikations- und Authentifizierungsprobleme beziehen sich auf Schwachstellen in Anwendungen, die es Angreifern ermöglichen, sich als legitime Benutzer auszugeben (auch bekannt als **Session-Hijacking**) oder unautorisierten Zugriff auf Benutzerkonten zu erlangen. Dies kann durch eine Vielzahl von Faktoren verursacht werden, wie zum Beispiel unzureichende Passwortrichtlinien, fehlende oder schwache Authentifizierungsmethoden, unsichere Übertragung von Anmeldeinformationen oder unzureichende Validierung von Benutzereingaben.

2.7 Cross-Site Request Forgery

"Cross-Site Request Forgery" (CSRF) ist eine Art von Angriff auf Webanwendungen, welches unterschiedliche Schwachstellen ausnutzt und verschiedene Auswirkungen kann.

Bei CSRF-Angriffen wird ein Angreifer den Benutzer dazu bringen, versehentlich eine Aktion auf der betroffenen Webseite auszuführen, indem er eine speziell gestaltete Webseite erstellt, die automatisch eine Anfrage an die betroffene Webseite sendet. Durch diese Anfragen kann ein Angreifer beispielsweise einen Kommentar auf einer Webseite im Namen des Benutzers posten, ein Konto löschen oder eine Geldüberweisung tätigen. CSRF-Angriffe können in vielen Fällen durch die Verwendung von Anti-CSRF-Tokens oder anderen Methoden zur Verhinderung von CSRF-Angriffen verhindert werden.

3. Risikobewertung

Diese Risiken können auch bei meiner Applikation eine bestimmte Rolle spielen. In diesem Kapitel schreibe ich über meinen Überlegungen bezüglich dessen und somit sind diese auch nach Wichtigkeit aufgelistet.

3.1 Injection

"Injection" ist das meist kritische Risiko, da es es einem Angreifer ermöglicht, schädlichen Code in meine Applikation zu «injecten». Dies kann zur Publizierung privater Daten führen oder zur Übernahme der Kontrolle über meiner Applikation.

3.2 Broken Access Control

"Broken Access Control" ist ebenfalls ein schwerwiegender Risikofaktor, da es Angreifern ermöglicht, auf Funktionen zuzugreifen, die sie nicht verwenden sollten. In meiner PHP Login/Register Applikation ist dies ein hohes Risiko, da es unbefugten Benutzern erlauben würde, auf vertrauliche Informationen zuzugreifen oder unerwünschte Aktionen auszuführen.

3.3 Identifikations- und Authentifizierungsprobleme

"Identifikations- und Authentifizierungsprobleme" sind ein bedeutendes Risiko, da sie es Angreifern ermöglichen, sich als andere Benutzer auszugeben oder sich ohne Autorisierung Zugriff zu verschaffen.

3.4 Cryptographic Failures

"Cryptographic failures" sind ein erhebliches Risiko, insbesondere für eine Login-/Register-Applikation, da sie Schwachstellen in der Verschlüsselung und Signaturüberprüfung aufdecken können.

3.5 Security Misconfiguration

"Security Misconfiguration" kann dazu führen, dass Ihre Anwendung anfälliger für Angriffe wird, insbesondere wenn die Konfiguration für die PHPMailer SMTP mail Verifizierung falsch oder unsicher ist.

3.6 Verwendung von unsicheren Komponenten

"Verwendung von unsicheren Komponenten" kann dazu führen, dass meine Applikation anfälliger für bekannte Schwachstellen wird.

3.7 Cross-Site Request Forgery

"Cross-Site Request Forgery" ist das Risiko, das am wenigsten Priorität hat, da es in der Regel durch den Einsatz von einfachen Sicherheitsmaßnahmen wie Tokens oder Captchas verhindert werden kann.

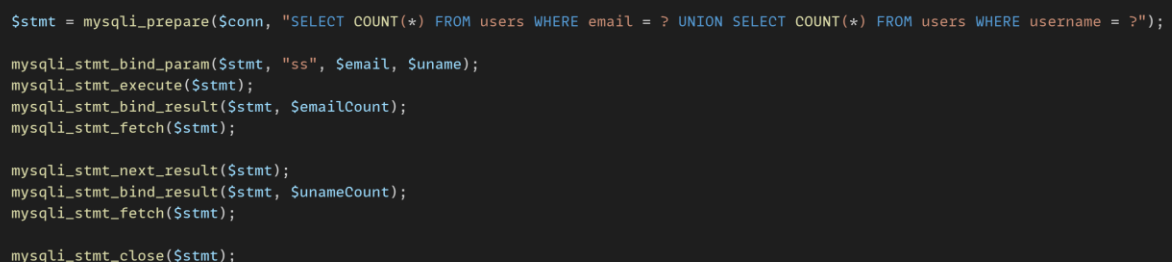
4. Risikomanagement

Erwähnt wurde zumindest, dass diese potenziellen Bedrohungen auch in meiner Applikation eine Rolle spielen (könnten), aber eben *könnten*. Dafür existieren auch Wege bzw. Sicherheitskonzepte, um diese potenziellen Bedrohungen zu verhindern.

4.1 Verhinderung von Injection

4.1.1 Prepared Statements

Ich habe Prepared Statements verwendet, um SQL-Injection-Angriffe zu verhindern. Durch Prepared Statements werden Eingabeaufforderungen und Daten getrennt, wodurch es für Angreifer schwieriger wird, schädlichen Code einzufügen.



```
register_process.php

$stmt = mysqli_prepare($conn, "SELECT COUNT(*) FROM users WHERE email = ? UNION SELECT COUNT(*) FROM users WHERE username = ?");

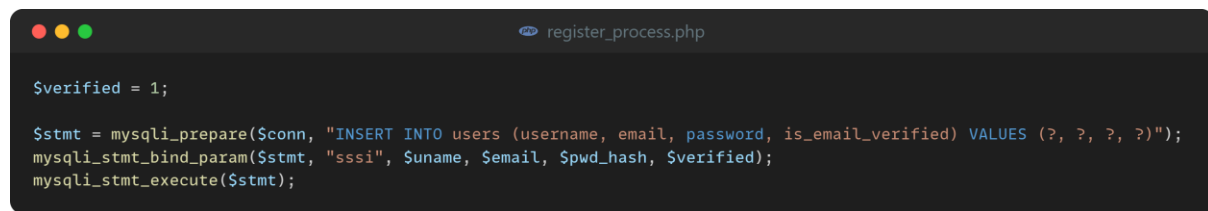
mysqli_stmt_bind_param($stmt, "ss", $email, $username);
mysqli_stmt_execute($stmt);
mysqli_stmt_bind_result($stmt, $emailCount);
mysqli_stmt_fetch($stmt);

mysqli_stmt_next_result($stmt);
mysqli_stmt_bind_result($stmt, $usernameCount);
mysqli_stmt_fetch($stmt);

mysqli_stmt_close($stmt);
```

Screenshot aus der Repository "modul-183-alperen-dev"

Hier wird überprüft, ob die eingegebene E-Mail und/oder der eingegebene Benutzername bereits in der Datenbank existieren. Diese Überprüfung mache ich nicht nur für die E-Mail, sondern auch für den Benutzernamen da bei der Anmeldung wird nur nach dem Benutzernamen und nach dem Passwort gefragt.

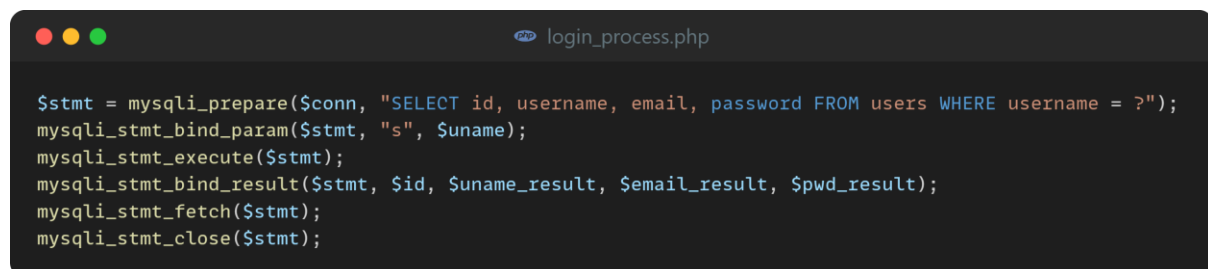
A screenshot of a code editor window titled 'register_process.php'. The code is in PHP and shows a database insertion operation. It sets a variable '\$verified' to 1, then prepares an SQL 'INSERT' statement for a 'users' table with columns 'username', 'email', 'password', and 'is_email_verified'. It binds parameters for '\$username', '\$email', '\$password_hash', and '\$verified' to the statement, and finally executes the statement.

```
$verified = 1;

$stmt = mysqli_prepare($conn, "INSERT INTO users (username, email, password, is_email_verified) VALUES (?, ?, ?, ?)");
mysqli_stmt_bind_param($stmt, "sssi", $username, $email, $pwd_hash, $verified);
mysqli_stmt_execute($stmt);
```

Screenshot aus der Repository "modul-183-alperen-dev"

In diesem Code-Snippet werden Daten in die Datenbank eingefügt. Ebenfalls wird das Feld `is_email_verified` auf 1 gesetzt, was heisst, dass die E-Mail somit verifiziert ist. Dieses Code-Snippet wird natürlich erst nach all den Filterungen, Validierungen, Verifizierungen etc. ausgeführt.

A screenshot of a code editor window titled 'login_process.php'. The code is in PHP and shows a database query operation. It prepares an SQL 'SELECT' statement to retrieve 'id', 'username', 'email', and 'password' from the 'users' table where 'username' matches a given value. It binds the username parameter, executes the query, fetches the results, and then closes the statement.

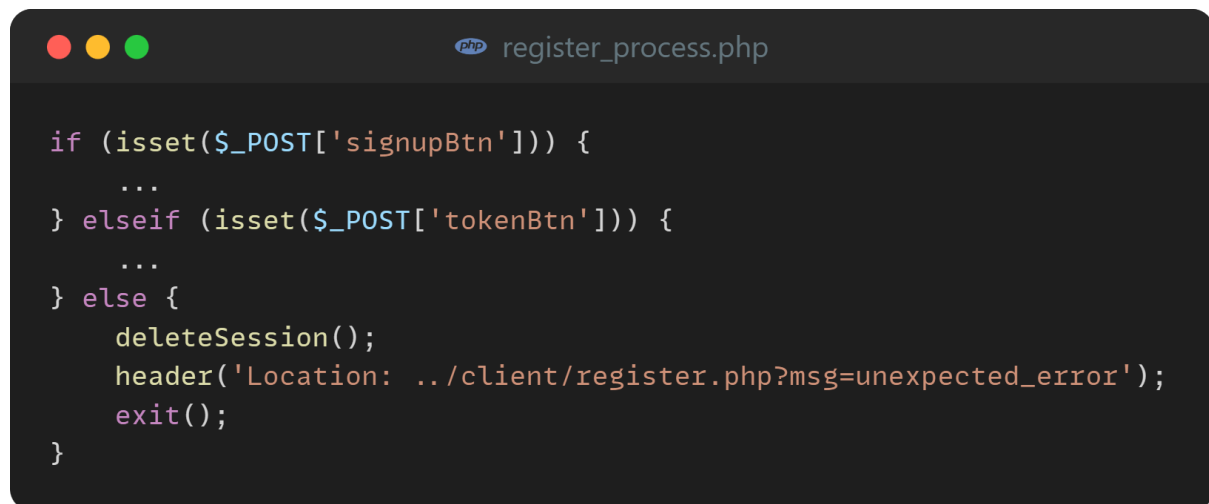
```
$stmt = mysqli_prepare($conn, "SELECT id, username, email, password FROM users WHERE username = ?");
mysqli_stmt_bind_param($stmt, "s", $username);
mysqli_stmt_execute($stmt);
mysqli_stmt_bind_result($stmt, $id, $username_result, $email_result, $pwd_result);
mysqli_stmt_fetch($stmt);
mysqli_stmt_close($stmt);
```

Screenshot aus der Repository "modul-183-alperen-dev"

Während des Login-Prozesses werden Daten aus der Datenbank geholt und die erhaltenen Resultate (anhand des vorhin angegebenen Benutzernamens, siehe `WHERE username = ?`) werden dann zu Variablen gebunden, welche später wichtig beim Setzen von Session Variablen, aber auch bei der Überprüfung des Passworts sind (siehe x.x).

4.1.2 Filterung und Validierung

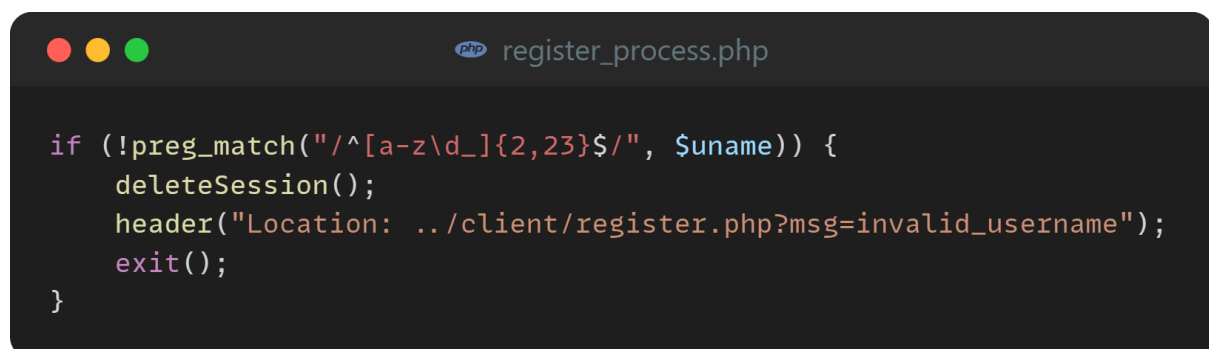
Ich habe auch verschiedene Arten von Filterungen und Validierungen implementiert, um sicherzustellen, dass nur gültige und erwartete Daten in die Datenbank eingefügt werden. Das verhindert XSS-Angriffe und schützt die Anwendung vor anderen Arten von Angriffen (wie eben Injection). Ebenfalls habe ich bestimmte Regex (Regular Expressions) verwendet, um z. B. Datenmanipulation u. ä. zu verhindern, auch aber damit die Datenbank u. a. lesbar bleibt. Zusätzlich um sicherzustellen, dass Benutzernamen, E-Mails und Passwörter gültige und erwartete Formate haben. Dadurch wird es schwieriger, unerwartete Eingaben einzugeben und somit Angriffe durch ungültige Eingaben zu verhindern.



```
if (isset($_POST['signupBtn'])) {  
    ...  
} elseif (isset($_POST['tokenBtn'])) {  
    ...  
} else {  
    deleteSession();  
    header('Location: ../client/register.php?msg=unexpected_error');  
    exit();  
}
```

Screenshot aus der Repository "modul-183-alperen-dev"

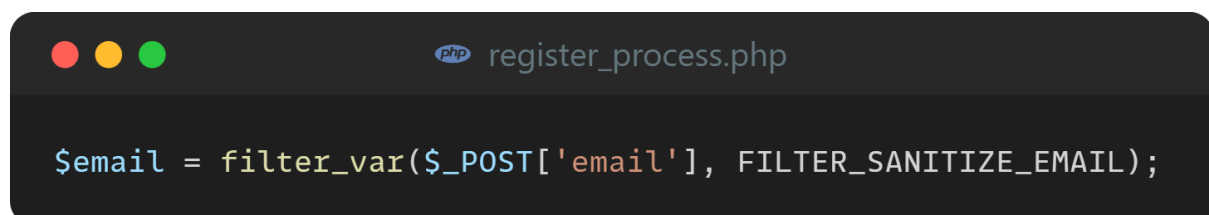
In diesem Code-Snippet wird z. B. überprüft bzw. validiert, ob der Benutzer durch den Klick eines Buttons auf `register_process.php` gekommen ist oder nicht. Falls nicht, wird die Session gelöscht und der Benutzer wird zurückgeschickt.



```
if (!preg_match("/^[a-z\d_]{2,23}$/", $uname)) {  
    deleteSession();  
    header("Location: ../client/register.php?msg=invalid_username");  
    exit();  
}
```

Screenshot aus der Repository "modul-183-alperen-dev"

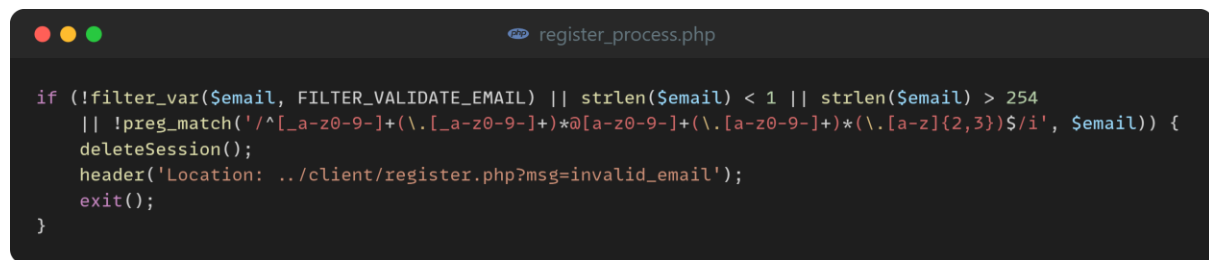
Dies ist ein Regex für den Benutzernamen. Es besagt, dass der Benutzername ausschliesslich aus "a-z", "A-Z", "0-9" und "_" zwischen 2-23 Zeichen bestehen kann.



```
$email = filter_var($_POST['email'], FILTER_SANITIZE_EMAIL);
```

Screenshot aus der Repository "modul-183-alperen-dev"

Hierbei wird die Variable `\$email` gesetzt mit `\$_POST['email']`. Gleichzeitig wird diese E-Mail dann auch mit Hilfe der `filter_var()` Funktion bzw. durch `FILTER_SANITIZE_EMAIL` saniert (sanitized). Dies ist wichtig, damit die E-Mail-Adresse korrekt formatiert und frei von unerwünschten Zeichen ist, die möglicherweise Sicherheitsprobleme verursachen könnten.



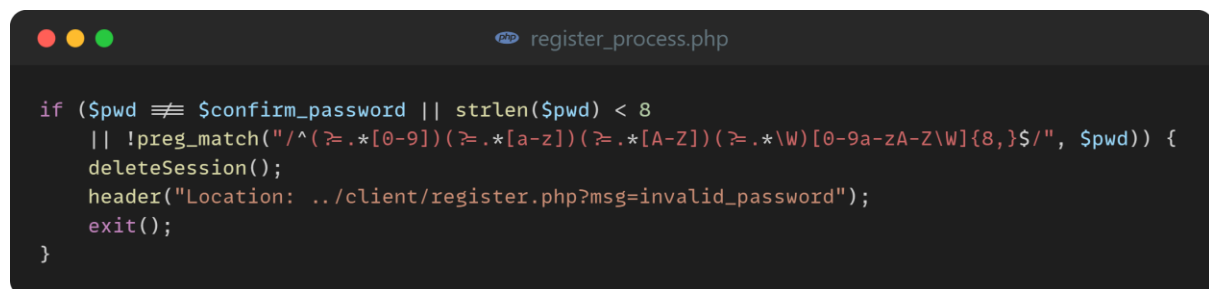
```
register_process.php

if (!filter_var($email, FILTER_VALIDATE_EMAIL) || strlen($email) < 1 || strlen($email) > 254
    || !preg_match('/^[a-z0-9-]+(\.[a-z0-9-]+)*@[a-z0-9-]+(\.[a-z0-9-]+)*(\.[a-z]{2,3})$/i', $email)) {
    deleteSession();
    header('Location: ../client/register.php?msg=invalid_email');
    exit();
}
```

Screenshot aus der Repository "modul-183-alperen-dev"

Dieser Code-Snippet validiert die E-Mail-Adresse und fügt ebenfalls zu der Sanitierung auch noch eine zusätzliche Überprüfung (bzw. Regex) hinzu.

‘FILTER_SANITIZE_EMAIL’ ist zwar ein guter erster Schritt, um eine E-Mail-Adresse zu sanieren (sanitize), kann aber nicht alle möglichen Fälle abdecken, wie z. B. ungültige Zeichen oder falsche Syntax. Durch die Verwendung von Regex zusätzlich zu ‘FILTER_SANITIZE_EMAIL’ kann man eine gründlichere Validierung durchführen und sicherstellen, dass die E-Mail-Adresse wohlgeformt und gültig ist.



```
register_process.php

if ($pwd !== $confirm_password || strlen($pwd) < 8
    || !preg_match("/^(?=.*[0-9])(?=.*[a-z])(?=.*[A-Z])(?=.*\W)[0-9a-zA-Z\W]{8,}$/", $pwd)) {
    deleteSession();
    header("Location: ../client/register.php?msg=invalid_password");
    exit();
}
```

Screenshot aus der Repository "modul-183-alperen-dev"

Dies überprüft, ob das eingegebene Passwort ‘\$pwd’ dem Feld, wo man das Passwort (also ‘\$pwd’) nochmals eintippen muss, entspricht. Ebenfalls darf das Passwort somit:

- Nicht weniger als 8 Zeichen beinhalten
- Muss mind. ein spezielles Zeichen wie "!"_<>\/+ "%&()=?" etc. beinhalten
- Muss mind. eine Zahl zwischen 0-9 beinhalten
- (Nicht nur) Lateinische Buchstaben beinhalten
- ...

"Warum verwende ich aber keine "Password-Blacklists" bzw. warum überhaupt ein Regex?"

Der Grund hierfür ist hauptsächlich die Usability, aber auch ist der Regex dafür da, um den Benutzern einiges mehr an Sicherheit und somit Datenschutz zur Verfügung zu stellen. Ein gutes Beispiel hierzu wären sogenannte "Password-Cracker", welche einfache Passwörter in kürzester Zeit knacken können. Jedoch verhindern diese "Password-Blacklists" auch nicht den Fakt, dass viele trotz des Password-Blacklists ihre Passwörter einfach ratbar machen, was dann sogar auch dazu führen kann, dass ein potenzieller Angreifer das Passwort manuell erraten kann. Um all diese Probleme

zu umgehen, wird in meiner Applikation einfach ein Regex erstellt, was ehrlich gesagt auch viel simpler (aber eben auch sicherer) ist. Das Argument "es kann für den Benutzer zu kompliziert werden" spielt heute auch gar keine Rolle mehr, denn wie man weiss, ist das Finden und somit "cracken" von Passwörtern heutzutage immens einfacher, warum man eben auch ein schwer ratbares Passwort erstellen kann (oder einfach ein Passwort-Manager wie KeepassXC verwenden).



```
register_process.php

$stmt = mysqli_prepare($conn, "SELECT COUNT(*) FROM users WHERE email = ? UNION SELECT COUNT(*) FROM users WHERE username = ?");

mysqli_stmt_bind_param($stmt, "ss", $email, $username);
mysqli_stmt_execute($stmt);
mysqli_stmt_bind_result($stmt, $emailCount);
mysqli_stmt_fetch($stmt);

mysqli_stmt_next_result($stmt);
mysqli_stmt_bind_result($stmt, $usernameCount);
mysqli_stmt_fetch($stmt);

mysqli_stmt_close($stmt);

if ($emailCount > 0) {
    deleteSession();
    header("Location: ../client/register.php?msg=email_exists");
    exit();
}

if ($usernameCount > 0) {
    deleteSession();
    header("Location: ../client/register.php?msg=username_exists");
    exit();
}
```

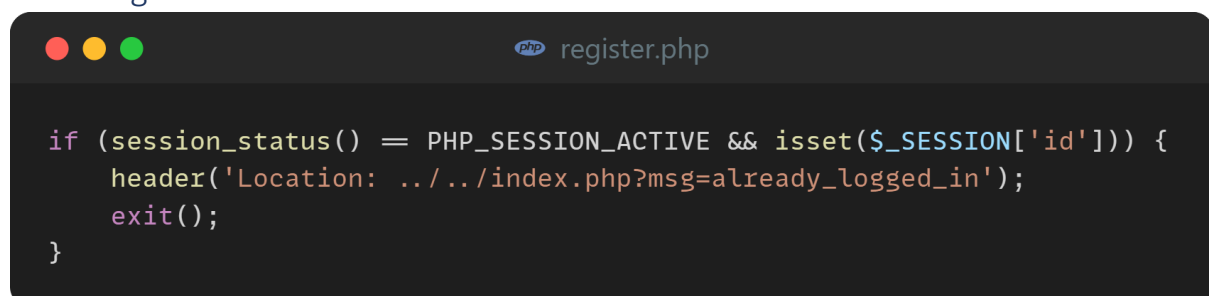
Screenshot aus der Repository "modul-183-alperen-dev"

Hier wird überprüft, ob die eingegebene E-Mail-Adresse oder der eingegebene Benutzername bereits (in der Datenbank) existiert.

4.2 Verhinderung von Broken Access Control

Um Broken Access Control zu verhindern, existieren normalerweise **Rollen**. Jedoch da es bei meiner Applikation nur um eine schlichte Login/Register-Applikation geht, ist es nicht nötig, diese einzubauen. Trotzdem muss man aber immer eine gewisse Zugriffskontrolle einbauen, z. B. wollen wir nicht, dass nicht-eingeloggte Benutzer Zugriff auf `profile.php` haben, worauf eigentlich eingeloggte Benutzer ihre Benutzerdaten betrachten können.

4.2.1 Zugriffskontrolle




```
register.php

if (session_status() == PHP_SESSION_ACTIVE && isset($_SESSION['id'])) {
    header('Location: ../../index.php?msg=already_logged_in');
    exit();
}
```

Screenshot aus der Repository "modul-183-alperen-dev"

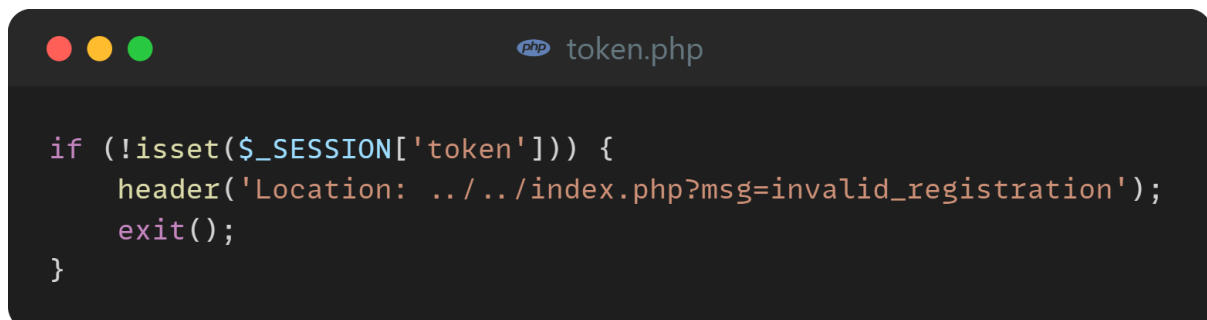
Hierbei wird überprüft, ob jemand, der bereits eingeloggt ist, versucht, auf die Seite `register.php` zuzugreifen. Denn nur korrekt eingeloggte Benutzer erhalten eine bereits gesetzte `\$_SESSION['id']` variable. Der gleiche Snippet ist im `login.php` und auch im `token.php` zu finden.

A screenshot of a code editor window titled "register.php" with a PHP icon. The code is as follows:

```
if (isset($_SESSION['token'])) {  
    session_unset();  
    $_SESSION = array();  
    session_destroy();  
    session_regenerate_id(true);  
    header('Location: ../../index.php?msg=invalid_registration');  
    exit();  
}
```

Screenshot aus der Repository "modul-183-alperen-dev"

Dies ist eine Überprüfung davon, ob `\$_SESSION['token']` bereits gesetzt ist oder nicht. Falls es gesetzt ist, wird der Benutzer zurückgeschickt und die vorhandene Session wird gelöscht. Aber warum? Im Falle meiner Applikation wird die Sessionvariable `\$_SESSION['token']` erst dann gesetzt, wenn der Benutzer versucht seine E-Mail-Adresse zu verifizieren aber anstatt den Token einzugeben z. B. durch die Adressleiste versucht, auf andere Dateien zuzugreifen. Aus diesem Grund (und auch wegen z. B. Session Hijacking u. ä.) ist dieses Snippet im source-code (und auch allgemein die Funktion `deleteSession()`, siehe x.x) öfters zu finden.

A screenshot of a code editor window titled "token.php" with a PHP icon. The code is as follows:

```
if (!isset($_SESSION['token'])) {  
    header('Location: ../../index.php?msg=invalid_registration');  
    exit();  
}
```

Screenshot aus der Repository "modul-183-alperen-dev"

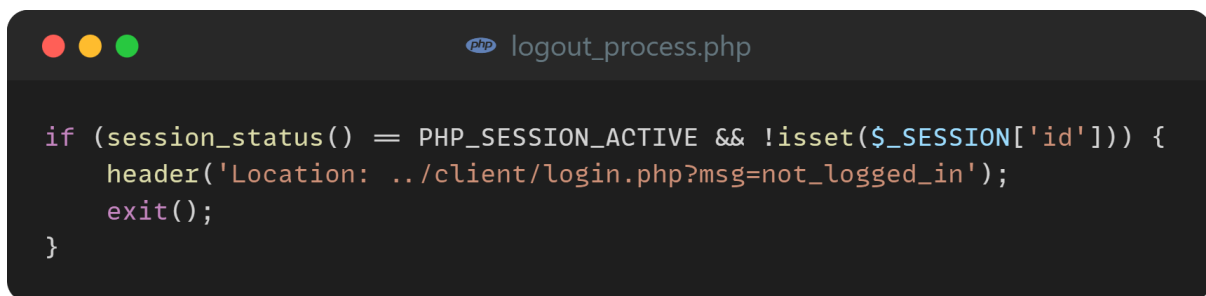
Wiederum hier wird überprüft, ob `\$_SESSION['token']` bereits gesetzt ist oder nicht. Falls nicht, wird der Benutzer zurückgeschickt. `\$_SESSION['token']` wird nach dem Eingeben der Daten bei der Registrierung gesetzt, wonach der Benutzer auf `token.php` mit der gesetzten Sessionvariable geschickt wird, worin er seine E-Mail-Adresse verifizieren muss.



```
if (basename($_SERVER['SCRIPT_FILENAME']) == 'db.php'){
    header('Location: ../.. /index.php?msg=access_denied');
    exit();
}
```

Screenshot aus der Repository "modul-183-alperen-dev"

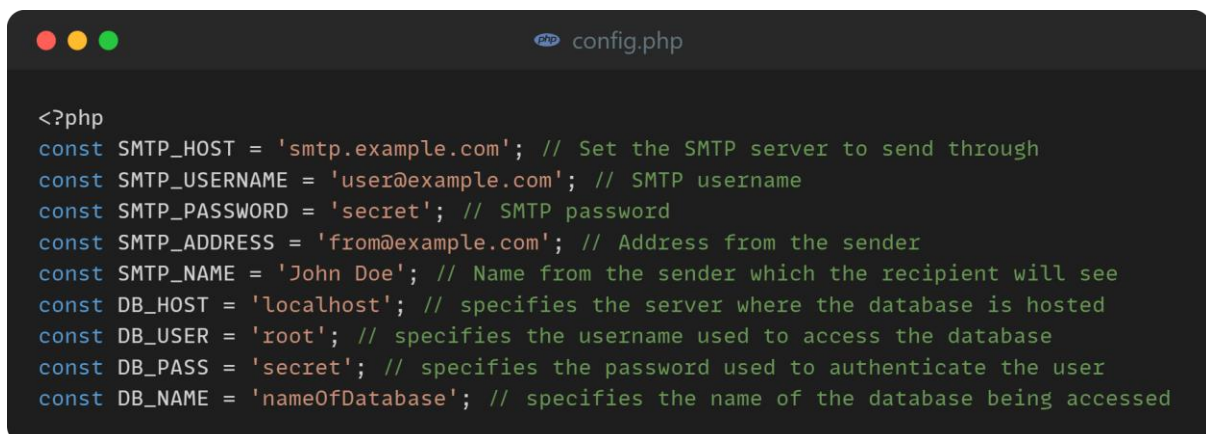
Ebenfalls überprüft wird, ob der Benutzer, unabhängig davon ob eingeloggt oder nicht, versucht, auf `db.php` zuzugreifen (im Falle meiner Applikation zu finden in `db.php`, `error.php` und `csrf.php`). Der Grund hierfür ist es aus Sicherheitsgründen, da wir z. B. nicht möchten, dass irgendjemand auf Dateien Zugriff hat, womit er/sie nichts zu tun hat.



```
if (session_status() == PHP_SESSION_ACTIVE && !isset($_SESSION['id'])) {
    header('Location: ../client/login.php?msg=not_logged_in');
    exit();
}
```

Screenshot aus der Repository "modul-183-alperen-dev"

Was wir auch nicht möchten, ist, dass nicht-eingeloggte Benutzer versuchen, auf `logout_process.php` zuzugreifen und somit auch versuchen, sich auszuloggen.



```
<?php
const SMTP_HOST = 'smtp.example.com'; // Set the SMTP server to send through
const SMTP_USERNAME = 'user@example.com'; // SMTP username
const SMTP_PASSWORD = 'secret'; // SMTP password
const SMTP_ADDRESS = 'from@example.com'; // Address from the sender
const SMTP_NAME = 'John Doe'; // Name from the sender which the recipient will see
const DB_HOST = 'localhost'; // specifies the server where the database is hosted
const DB_USER = 'root'; // specifies the username used to access the database
const DB_PASS = 'secret'; // specifies the password used to authenticate the user
const DB_NAME = 'nameOfDatabase'; // specifies the name of the database being accessed
```

Screenshot aus der Repository "modul-183-alperen-dev"

Damit z. B. API-Keys oder Passwörter (Datenbanken, Libraries etc.) nicht im source-code sind, da wir nicht wollen, dass andere darauf Zugriff haben, verwende ich eine Datei namens `config.php` was somit dann im `.gitignore` ist und nicht in der Repository zu finden ist. Dies ist essenziell, um mit MySQL und PHPMailer zu arbeiten.

4.3 Verhinderung von Identifizierungs- und Authentifizierungsproblemen

Durch bereits vorhin erwähnten Passwortrichtlinien sollten diese "Identifizierungs- und Authentifizierungsprobleme" bereits zu einem gewissen Grad verhindert werden. Jedoch aber eben nur "zu einem gewissen Grad". Deswegen wurden auch ein paar andere Dinge implementiert wie die E-Mail-Verifizierung, wodurch man davon ausgehen kann, dass der registrierte Benutzer auch der/die richtige ist und somit auch Zugriff auf seine/ihre E-Mail-Adresse hat.

Noch zu sagen ist, ist dass hierbei auch noch ein Limit anhand der IP-Adresse des Benutzers angewendet werden könnte, um Brute-Force-Angriffe zu verhindern, was aber in meiner Applikation nicht implementiert wurde. Auch wenn es eine gute Idee zu sein scheint, die Anzahl der Login-/Registerversuche zu begrenzen, um Brute-Force-Angriffe zu verhindern, besteht die Gefahr, dass dieser Ansatz mehr schadet als nützt. Bei einer einfachen Login/Register-Applikation, bei der die Anzahl der Benutzer sehr wahrscheinlich gering ist, ist ein Limit möglicherweise nicht notwendig. Sie kann sogar kontraproduktiv sein, da Benutzer, die sich bei der Eingabe ihrer Anmeldedaten vertippen oder ihr Passwort vergessen, durch die Beschränkungen frustriert werden und die App ganz aufgeben.

4.3.1 E-Mail-Verifizierung

```
register_process.php

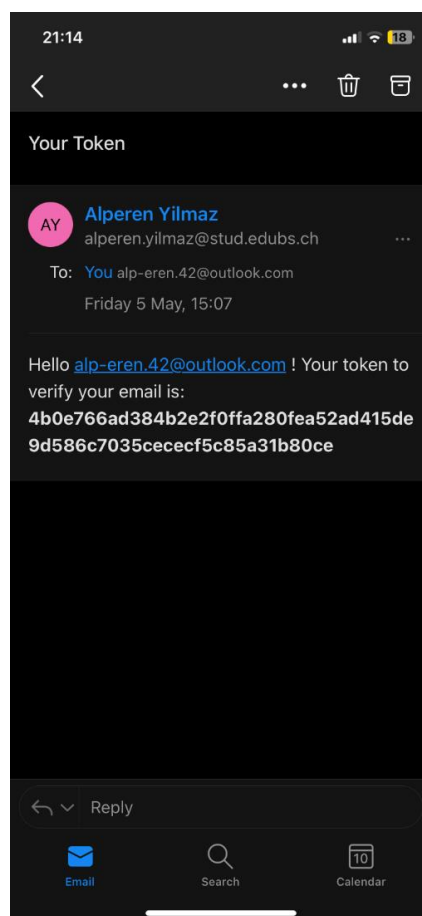
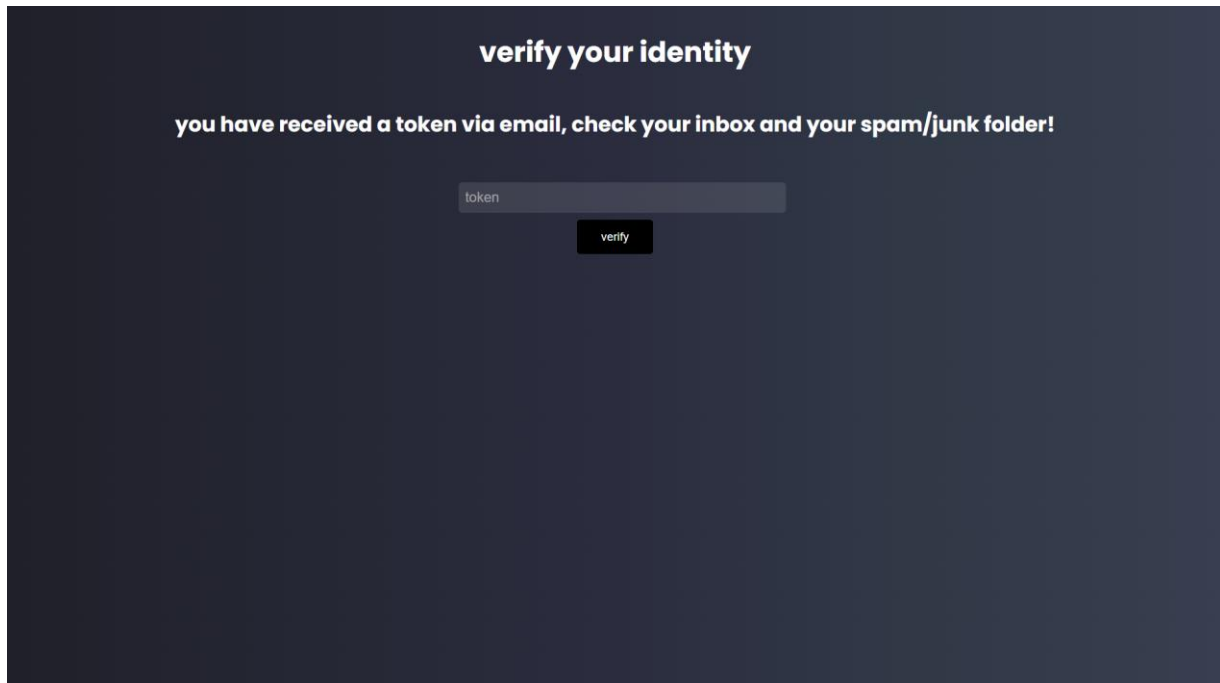
//starting instance of PHPMailer, following with the try/catch for the mail verification
$mail = new PHPMailer(true);
try {
    // $mail->SMTPDebug = SMTP::DEBUG_SERVER; // use only if something doesn't work after pressing signupBtn
    $mail->isSMTP();
    $mail->Host = SMTP_HOST;
    $mail->SMTPAuth = true;
    $mail->Username = SMTP_USERNAME;
    $mail->Password = SMTP_PASSWORD;
    $mail->SMTPSecure = PHPMailer::ENCRYPTION_STARTTLS;
    $mail->Port = 587;

    //Sender and recipient of the mail
    $mail->setFrom(SMTP_ADDRESS, SMTP_NAME);
    $mail->addAddress($email);

    //Content of the mail
    $mail->isHTML(true); // why html? because of rich formatting, better branding, better tracking, ...
    $mail->Subject = 'Your Token';
    $mail->Body = 'Hello ' . $email . ' ! Your token to verify your email is: <b>' . $_SESSION['token'] . '</b>';
    /*
    * Why use an "AltBody"? because, some (ancient) email providers do NOT support html
    * such as mutt or Eudora, and to ease this process we also send the mail with plain-text characters
    */
    $mail->AltBody = 'Hello ' . $email . ' ! Your token to verify your email is: ' . $_SESSION['token'];
    $mail->send();
    header('Location: ../client/token.php');
} catch (Exception $e) {
    echo "Message could not be sent. ERR_INFO: {$mail->ErrorInfo}";
    exit();
}
```

Wie bereits erwähnt, um die Benutzer einfacher und besser zu identifizieren, ist es nötig, bei solchen Applikationen Multi-Faktor-Authentifizierung anzuwenden, im Fall

meiner Applikation kommt dies in Form von einer E-Mail-Verifizierung vor mit Hilfe der PHPMailer Library. Durch SMTP wird eine E-Mail mit einem Token zu der vorher angegebenen E-Mail des Benutzers gesendet. Diesen Token muss dann der Benutzer eingeben, um sich erfolgreich zu registrieren.



4.3.2 Session-Hijacking

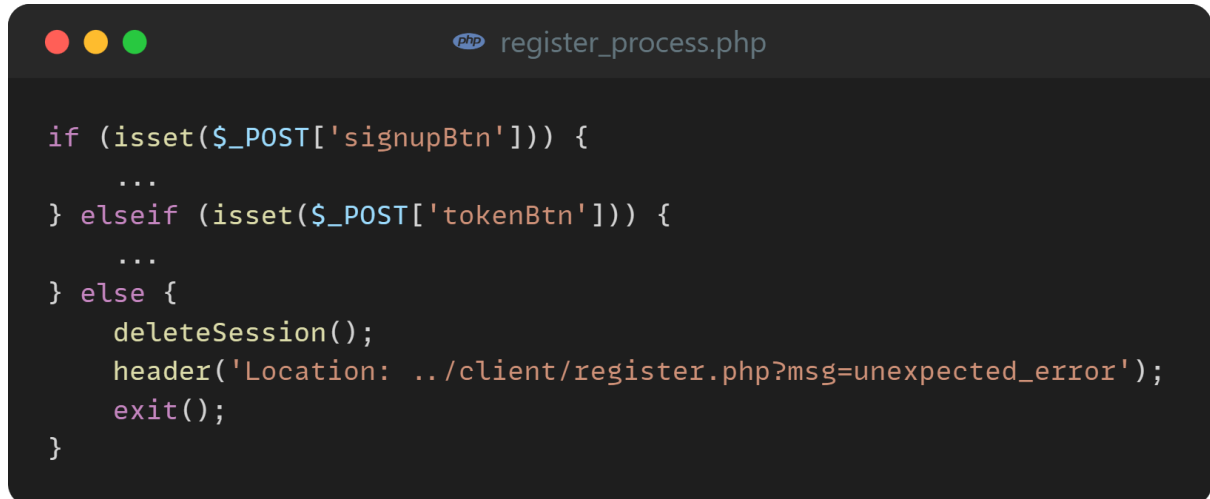
Was jedoch auch passieren kann, ist das sogenannte Session-Hijacking. Wie dies verhindert wird, sehen Sie im Folgenden:

A screenshot of a code editor window titled 'register_process.php'. The editor has a dark background and shows a PHP function named 'deleteSession()' which is void. The function contains four lines of code: 'session_unset();', '\$_SESSION = array();', 'session_destroy();', and 'session_regenerate_id(true);'. The function is enclosed in curly braces.

```
function deleteSession(): void {  
    session_unset();  
    $_SESSION = array();  
    session_destroy();  
    session_regenerate_id(true);  
}
```

Screenshot aus der Repository "modul-183-alperen-dev"

Dies ist eine (in meiner Applikation oft verwendete) Funktion, welche die bestehende Session leert, löscht und dann auch noch die Session IDs regeneriert. Dies ist in meinem Fall vor allem wichtig, da ich viel mit Sessions arbeite, z. B. beim Übergeben von Variablen wie `\$_uname`, `\$_email`, `\$_pwd_hash` oder der Token, womit man sich verifizieren muss. Dies ist, weil es in meinem Fall nicht wirklich anders geht, siehe:

A screenshot of a code editor window titled 'register_process.php'. The editor has a dark background and shows a PHP conditional block. It starts with 'if (isset(\$_POST['signupBtn'])) {' followed by three dots. Then it has 'elseif (isset(\$_POST['tokenBtn'])) {' followed by three dots. Then it has 'else {' followed by 'deleteSession();', 'header('Location: ../client/register.php?msg=unexpected_error');', and 'exit();'. The block is closed with '}'.

```
if (isset($_POST['signupBtn'])) {  
    ...  
} elseif (isset($_POST['tokenBtn'])) {  
    ...  
} else {  
    deleteSession();  
    header('Location: ../client/register.php?msg=unexpected_error');  
    exit();  
}
```

Screenshot aus der Repository "modul-183-alperen-dev"

Variablen, welche im ersten `if` bestimmt werden (zu diesem `if` kommt man nur durch `register.php`, worin sich der Button `signupBtn` befindet) sind nicht überall zugänglich. Auf das `elseif` kommt man nur dann, wenn man im `token.php` auf den Button `tokenBtn` klickt. Aber was ist, wenn man all diese Variablen ausserhalb von diesen Statements schreibt? Dann jedoch haben wir das Problem, dass es im `elseif`, aber auch im `if` zu Problemen kommt. Denn `\$_uname`, `\$_email` und `\$_pwd` werden nur vom `register.php`-Form gePOSTet, wodurch dann das `elseif`-Statement nicht

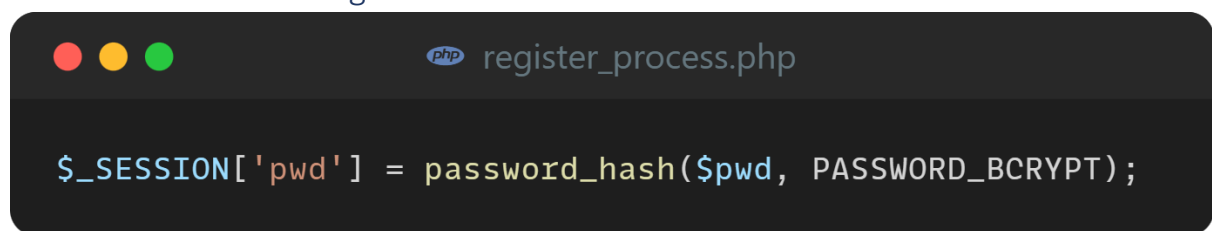
weiss, was diese sind, falls es dann später ausgeführt wird. Das gleiche Konzept wäre dann auch bezüglich des Tokens, womit der Benutzer seine E-Mail verifizieren muss.

Um diese Probleme zu lösen, arbeite ich hierbei mit Sessionvariablen, welche aber häufig geleert/gelöscht/regeneriert werden, um eben Session Hijacking u. ä. zu verhindern. Ausserdem wird nur das gehashte Passwort in die Sessionvariable übertragen, worüber man sich also keine grossen Sorgen machen muss (siehe x.x).

4.4 Verhinderung von "Cryptographic failures"

Um "Cryptographic failures" zu verhindern, wende ich in meiner Applikation zwei verschiedene Verschlüsselungsalgorithmen bei zwei verschiedenen Fällen.

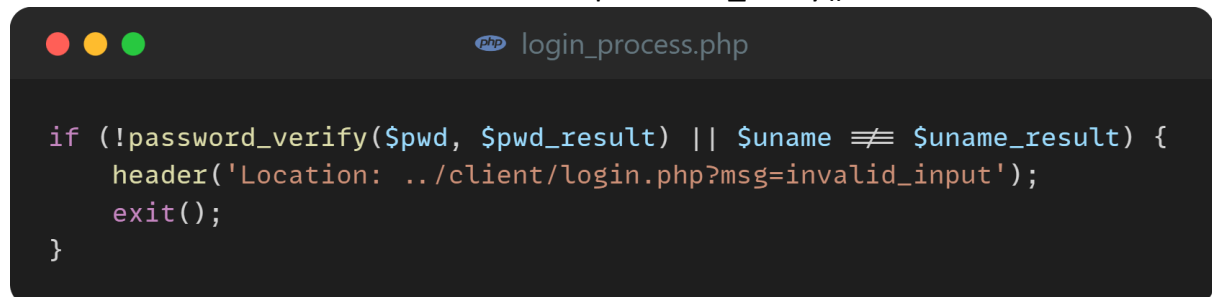
4.4.1 Password-Hashing



Screenshot aus der Repository "modul-183-alperen-dev"

In diesem Code-Snippet wird die Sessionvariable `pwd` mit einem Hash gesetzt. Genauer gesagt wird `\$pwd` mit dem Algorithmus "bcrypt" gehasht, da dies ein starker, sicherer und weitverbreiteter Hashing-Algorithmus ist. Ausserdem generiert bcrypt automatisch einen zufälligen "Salt" für jeden Hash. Im Zusammenhang mit bcrypt ist ein Salt ein zufällig generierter Wert, der zu einem Passwort hinzugefügt wird, bevor es gehasht wird. Dieser Hash wird im Falle meiner Applikation dann später in die Datenbank eingefügt, und somit nicht das "ungehashte" Passwort selbst. Aus diesen Gründen finde ich es mehr als angenehm, auch bcrypt als das Hashing-Algorithmus für das Passwort zu verwenden.

Um im Login dann zu überprüfen, ob das eingegebene Passwort das gleiche wie der Hash in der Datenbank ist, verwende ich `password_verify()`:

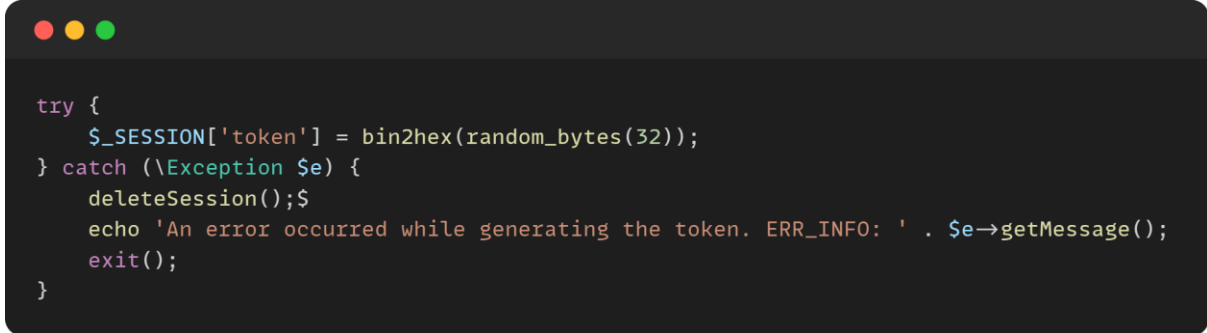


Screenshot aus der Repository "modul-183-alperen-dev"

In diesem Fall ist `\$pwd_result` das Passwort aus der Datenbank und `\$pwd` das vom Benutzer eingegebene Passwort (beim Einloggen).

4.4.2 Tokenisierung

Um den ganzen Verifizierungsprozess noch sicherer zu machen, erstelle ich ein Token, welchen der Benutzer per E-Mail erhält und anschliessend angeben muss, um seine E-Mail zu bestätigen.



```
try {  
    $_SESSION['token'] = bin2hex(random_bytes(32));  
} catch (\Exception $e) {  
    deleteSession();  
    echo 'An error occurred while generating the token. ERR_INFO: ' . $e->getMessage();  
    exit();  
}
```

Screenshot aus der Repository "modul-183-alperen-dev"

Hierbei wird die Sessionvariable `\$_SESSION['token']` mit einem sicheren, zufällig generierten 32-bit Token gesetzt.

4.5 Verhinderung der "Verwendung von unsicheren Komponenten"

Für die E-Mail-Verifizierung verwende ich eine Library namens PHPMailer, was auch die einzige Library ist, welche ich bei meiner Applikation verwende.

PHPMailer ist eine beliebte Bibliothek für den E-Mail-Versand in PHP, die den Versand von E-Mail-Nachrichten aus einem PHP-Skript vereinfacht. Die Bibliothek bietet eine Reihe von Sicherheitsfunktionen, die sie zu einer sicheren Wahl für den E-Mail-Versand machen.

Die wichtigsten Sicherheitsfunktionen von PHPMailer sind:

SSL/TLS-Verschlüsselung: PHPMailer unterstützt die SSL/TLS-Verschlüsselung, mit der die E-Mail-Nachricht bei der Übertragung vom Server zum Mailserver des Empfängers verschlüsselt werden kann. Dies hilft, Abhören zu verhindern und stellt sicher, dass die Nachricht nicht von Unbefugten abgefangen und gelesen werden kann.

SMTP-Authentifizierung: PHPMailer unterstützt die SMTP-Authentifizierung, bei der der Benutzer einen Benutzernamen und ein Passwort angeben muss, um sich gegenüber dem Mailserver zu authentifizieren. Dies hilft, unbefugten Zugriff auf den Mailserver zu verhindern und stellt sicher, dass nur autorisierte Benutzer E-Mail-Nachrichten versenden können.

Content validation: PHPMailer bietet eine integrierte Inhaltsüberprüfung, um Angriffe durch E-Mail-Injections zu verhindern. Dies beinhaltet die Überprüfung der E-Mail-Adresse des Absenders und stellt sicher, dass der Inhalt der E-Mail-Nachricht richtig formatiert ist und keinen böartigen Code enthält.

Error reporting: PHPMailer bietet eine detailliertes Error reporting, um Sicherheitsprobleme, die während des E-Mail-Versands auftreten können, zu identifizieren und zu beheben. Dadurch wird sichergestellt, dass Schwachstellen schnell erkannt und behoben werden.

Für mehr Details, siehe x.x

4.6 Verhinderung von Cross-Site Request Forgery

Um CSRF zu verhindern, habe ich dies einfach mit einem Token gelöst. *(nicht zu verwechseln mit dem Token, welches der Benutzer per E-Mail für die E-Mail-Verifizierung erhält!)*

Die Datei `csrf.php` enthält Funktionen, welche diese Tokens generieren:

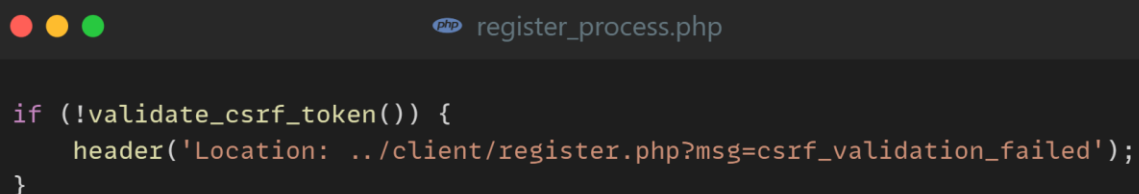
A screenshot of a code editor window titled 'csrf.php'. The code defines two functions: 'generate_csrf_token()' and 'validate_csrf_token()'. The first function generates a random 32-byte token and stores it in the session. The second function checks if the token in the POST data matches the one in the session.

```
function generate_csrf_token(): string {
    if (!isset($_SESSION['csrf_token'])) {
        try {
            $_SESSION['csrf_token'] = bin2hex(random_bytes(32));
        } catch (Exception $e) {
            echo 'Something went wrong. ERR_INFO: ' . $e->getMessage();
            exit();
        }
    }
    return $_SESSION['csrf_token'];
}

function validate_csrf_token(): bool {
    if (isset($_POST['csrf_token']) && isset($_SESSION['csrf_token']) && $_POST['csrf_token'] === $_SESSION['csrf_token']) {
        return true;
    }
    return false;
}
```

Screenshot aus der Repository "modul-183-alperen-dev"

Meine Applikation enthält genau drei Formulare, und zwar im `register.php`, `login.php` und `token.php`. Im Falle der Registrierung läuft es etwa so ab: `generate_csrf_token()` generiert ganz oben in der Datei `register.php` ein Token, welcher dann in ein `type=hidden` Inputfeld reingetan wird. Dieses Formular schickt dann diese zu `register_process.php` über POST weiter und dann wird im `register_process` überprüft, ob der erhaltene CSRF Token der ist, den er auch erwartet (der erwartete Token ist eben der, der in `register.php` generiert wird). Das gleiche ist es dann auch bei den anderen Formularen.

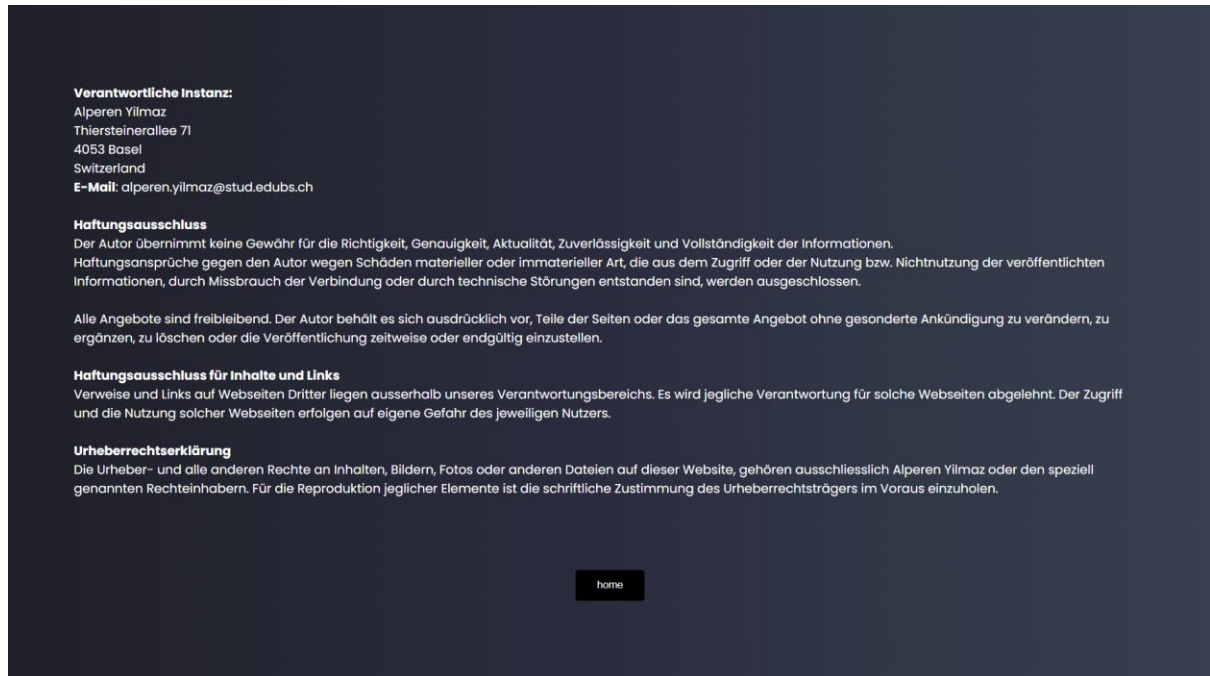
A screenshot of a code editor window titled 'register_process.php'. The code shows a conditional statement that checks if the CSRF token is valid. If it's not valid, it sets a header to redirect the user to a message page.

```
if (!validate_csrf_token()) {
    header('Location: ../client/register.php?msg=csrf_validation_failed');
}
```

5. Weiteres

5.1 Impressum

Für meine Applikation habe ich ebenfalls noch aus Datenschutzgründen ein Impressum erstellt, siehe:



Verantwortliche Instanz:
Alperen Yilmaz
Thiersteinerallee 71
4053 Basel
Switzerland
E-Mail: alperen.yilmaz@stud.edubs.ch

Haftungsausschluss
Der Autor übernimmt keine Gewähr für die Richtigkeit, Genauigkeit, Aktualität, Zuverlässigkeit und Vollständigkeit der Informationen. Haftungsansprüche gegen den Autor wegen Schäden materieller oder immaterieller Art, die aus dem Zugriff oder der Nutzung bzw. Nichtnutzung der veröffentlichten Informationen, durch Missbrauch der Verbindung oder durch technische Störungen entstanden sind, werden ausgeschlossen.

Alle Angebote sind freibleibend. Der Autor behält es sich ausdrücklich vor, Teile der Seiten oder das gesamte Angebot ohne gesonderte Ankündigung zu verändern, zu ergänzen, zu löschen oder die Veröffentlichung zeitweise oder endgültig einzustellen.

Haftungsausschluss für Inhalte und Links
Verweise und Links auf Webseiten Dritter liegen ausserhalb unseres Verantwortungsbereichs. Es wird jegliche Verantwortung für solche Webseiten abgelehnt. Der Zugriff und die Nutzung solcher Webseiten erfolgen auf eigene Gefahr des jeweiligen Nutzers.

Urheberrechtserklärung
Die Urheber- und alle anderen Rechte an Inhalten, Bildern, Fotos oder anderen Dateien auf dieser Website, gehören ausschliesslich Alperen Yilmaz oder den speziell genannten Rechteinhabern. Für die Reproduktion jeglicher Elemente ist die schriftliche Zustimmung des Urheberrechtsträgers im Voraus einzuholen.

[home](#)

6. Fazit

Auch wenn es nur eine schlichte Login/Register-Applikation war, hat mir dieses Projekt im Grossen und Ganzen sehr gefallen, da ich mich schon immer mehr mit Sicherheitskonzepten und auch mit Verifizierungen (wie mit PHPMailer bzw. SMTP) befassen wollte. Ich habe vieles dazu gelernt, leider aber ist dies mein letztes Projekt bei der Informatikmittelschule... aber dafür freue ich mich auf das Praktikumsjahr!

Geschrieben von Alperen Yilmaz, 10. Mai 2023 um 23:03 Uhr