PHP Signup System:

db.sql:

* Tabelle anlegen in mySQL Datenbank.

Includes Ordner anlegen. DB Handler dbh.inc anlegen.

dbh.inc.php: DATENBANK VERBINDUNG HERSTELLEN

* Parameter (host, dbname etc.) anlegen um Verbindung zur DB herstellen.
* Try, catch anlegen.
* Im catch Ausnahme anlegen für den Fall, dass Verbindung fehlschlägt: PDOException
* Connection Object $pdo der Class PDO anlegen, parameter sind Datenbank mySQL, host, dbname, username und passwort
* Für diese $pdo auf Methode setAttribut verweisen mit Parametern. Parameter sind die static Methodes ATTR\_ERRMODE (Error reporting mode of PDO with value ERRMODE\_EXEPTION): ERRMODE\_EXCEPTION (Throw a [PDOException](https://www.php.net/manual/en/class.pdoexception.php) if an error occurs. As of PHP 8.0.0, this is the default mode. In addition to setting the error code, PDO will throw a[PDOException](https://www.php.net/manual/en/class.pdoexception.php) and set its properties to reflect the error code and error information.) für die Class PDO

Jetzt gehts es um die Sicherheit, bei Verwendung von session cookies. ->

Config\_session.inc: SESSION STARTEN UND REGENERIEREN

* 2x ini\_set , das sollte man immer tun bzw. ändern um die sicherheit zu erhöhen
* Cookie Parameter ändern session\_set\_cookie\_params. Das ist eine function, die ein array aufmacht in der wir parameter ändern können.
* Wir starten eine session session\_start()
* If condition mit der wir alle 30 minuten unsere session cookie regenerieren um sicherheit zu erhöhen. Session\_regenerate\_id().ist sicherer als die die automatisch gestartet wurde. (diese if ist jetzt in der else)
* If checkt ob eine session existiert oder nicht also !isset($\_SESSION[‚last\_regenaration‘])
* Falls nicht: else -> setzten wir das interval auf 60 \* 30 Sekunden.
* Ffalls mehr als 30 minuten vergangen sind regenerieren wir eine neue session is.
* Dazu legen wir eine function regenerate\_session\_id() an.

Signup.inc.php: DATENABFRAGE AUS EINGABEFELDERN / VERBINDUNG ZU DATENBANK UND MVC SCRIPTEN

* File anlegen
* Wir checken ob der user etwas in unser feld eingegeben hat mit der REQUEST\_METHOD === POST. Falls nicht, soll er zurück zur frontpage index.php. Und wir halten das Script an mit die().
* Falls er was „postet“ hat, grabben wir uns die eineggeben Werte und speichern sie in variablen $vorname, $nachname etc.
* Wir nehmen jetzt htmlspeciochars() oder so, sondern erst wenn wir einen output auf der website machen.
* Try, catch, alles etwas falsch läuft zuerst.
* Im try: verbindung herstllen mit require\_once zu dbh.inc.php.

MVC-Model:

* Wir machen weiter mit m v c Model und legen die files dafür an. Mit dem model kommunizieren wir mit der DB. Mit der view zeigen wir etwas auf der webpage an. Mit der controle werden wir unseren input auf der website handlen.
* Da in der \_model sehr sensitive funktionen liegen, die die die queries an die db senden, kommuniziert nur die contr.inc mit ihr, da sie die daten aus den Inputs handlet

Model.inc:

* Wir aktivieren strict\_types was uns erlaubt type declerationen zu machen.. das machen wir auch in der contr. Und view.

Signup.inc: ANLEGEN EINES ERROR ARRAYS

* Wir verbinden nun auch zu contr.inc und view.inc. Reihenfolge der verbindung ist wichtig! Immer m, v, c
* Weiter im try Block. Wir aktivieren error handlers. Das machen wir in der contr.inc

Contr.inc: EINGABEN KONTROLLIEREN AUF ERRORS MIT FUNKTIONEN

* Function anlegen ob der input leer ist: is\_input\_empty für alle inputfelder. Mit eingebauter function empty() im if stmt. Falls ein Fehler vorhanden returnen wir true.
* Wir checken noch ob email valide ist mit mit eingebauter filter\_var und build in validation FILTER\_VALIDATE\_EMAIL
* Jetzt lege ich function is\_name\_wrong um zu schauen ob Vor- und Nachname mindestens zwei Buchstaben lang ist. Und is\_pwd\_wrong und checke auch hier das erforderliche pattern.
* Nun schauen wir noch, ob email bereits vergeben ist. Da wir hierfür mit db kommunizieren, müssen wir das in der model.inc. machen und erstmal email abfragen

Model.inc: EMAIL ABFRAGE AUS DB FÜR ERROR KONTROLLE

* Function get\_email: mit connection $pdo können wir $query starte.
* Um es sicher zu machen, machen wir es mit prepare statement um sql injection zu verhindern.
* Das stmt wiederum nutzt die methode bindParam und sagt wir bindendie daten von $email und binden sie an den platzhalter :email
* In $result wird des ergebnis gespeichert und zwar nutzt das object $stmt die methode fetch, wodurch nur 1 datensatz undzwar der erste geschnappt wird. Und zwar als assoziatives array mit FETCH\_ASSOC. Wodurch wir auch den spaltennamen nutzen können, also email = die Email adresse. Die function kommtin die if condition in der contr.inc

Contr.inc.: EMAIL KONTROLLIEREN AUF ERRORS MIT FUNKTIONEN

* Function is\_email\_registered anlegen

Signup.inc: ERROR AUSGABEN ERSTELLEN UND IN ERROR ARRAY SPEICHERN!

* If-stmt für is\_email\_registered erstellen.
* Error ausgaben erstellen und im Array $errors speichern mit keys und values.. Also falls es irgendein error gibt, kann nicht registriert werden. Wir vergeben selbst einen key z.B. „wrong\_name“. Wenn es keine Fehler gibt ist das Array leer.
* If-stmt gibt es Fehler? If($errors) GIBT ES FEHLER?
* Wir starten eine session und storen die $errors in der session unter dem namen „errors\_signup“. SPEICHER SIE IN EINER SESSION VARIABLE
* Und gehen zurück zu unserer index.php mit header(), um da die errors auszugeben!

(Index.php:

* Hier soll die function check\_signup\_errors gestarten werden! Diese werden wir in der view.inc. anlegen, da etwas ausgegeben wird.)

View.php: HOL DIE DIR FEHLER AUS DER SESSION VARIABLE UND ZEG SIE AUF DER WS!)

* (If-stmt: gibt es eine $\_SESSION mit namen „errors\_signup“, falls ja, dann soll das Array, das in der $\_SESSION gespeichert wurde in der Variablen $errors gespeichert werden.
* Gib die Values der Fehler auf der Website aus!
* Mit unset wird abschließend die session geleert, weil wir die fehler nicht mehr brauchen. Bzw. wird beim nächsten versuch wieder darin gespeichert. LEERE SESSION!

Index.php:

Php code oben einsetzten damit wir auf alle files zugreifen können mit require\_once

Signup.inc.php: USER ERSTELLEN IN DB mit Aufruf der funktion create\_user()

Signup\_contr.inc.php: FUNCTION CREATE\_USER ERSTELLEN, die wiederum function set\_user aufruft, die in model.inc. definiert ist, da kommuniktion mit db.

Model.inc: LEGT USER IN DB AN!

* Set\_user greift auf db zu!
* Die values werden als platzhalter übergeben und dann an die variablen gebunden
* Mit options geben wir einen cost factor an, damit der hash gut wird ☺
* Dann hashen wir das pwd mit einem hashing algorithmus und option
* Wir storen das hashed pwd und nicht das pwd in unserer db

Signup.inc.php:

* Wir übergeben der create\_user die parameter.
* Wir leiten die user wieder zu unserer index.php Seite und geben an dass signup=suuccess in url. Das geht über die header.
* Wir beenden die connection und statement mit =null und die()
* USIBILITY FEATURE: DATEN BEI ERROR AN SIGNUP FORM-ZURÜCKSENDEN UND IM INPUT ZEIGEN
* - in den if($errors) ein Array mit Daten kreieren -> Signup\_data, die wir bereits eingegeben hatten. Bis auf pwd. Da sollte man jedes mal neu eingeben.
* Die Daten aus signup\_data speichern wir in einer $\_SESSION Variable

View.php:

* Signup\_success function anlegen. Gibt signup\_succes aus, wenn in der url ?signup===success steht (siehe signup.inc)

Index.php:

* Signup\_success starten nach dem button

Index.php:

* Wir ersetzen die html sign up form durch php -> signup\_inputs() und checken ob wir bereits daten zurückgeschickt haben in der session. Falls ja, wollen wir eine andere version (mit Daten) anzeigen!

View.php:

* Hier kreieren wir die signup\_inputs(), in der wir error messages anzeigen und input felder mit Daten falls in Session vorhanden oder ohne Daten.