

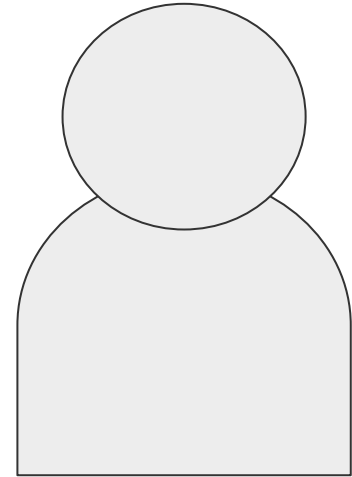
Webentwicklung

FWPM

Nutzerverwaltung und Security

Nutzeraccounts - Warum?

- Erlaubt langfristige Identifikation von Nutzern
 - Z.B. zur Personalisierung der Anwendung für Nutzer
- Ermöglicht Nutzern Einfluss auf die Anwendung zu nehmen
 - Z.B. eigenes Profil zu pflegen
 - Inhalte zu verwalten
- Erlaubt Unterscheidung von Nutzern
- Bringen aber auch viel Komplexität mit sich
 - Sicherheitsbedenken
 - Register/Login Mechanismen
 - Rechteverwaltung



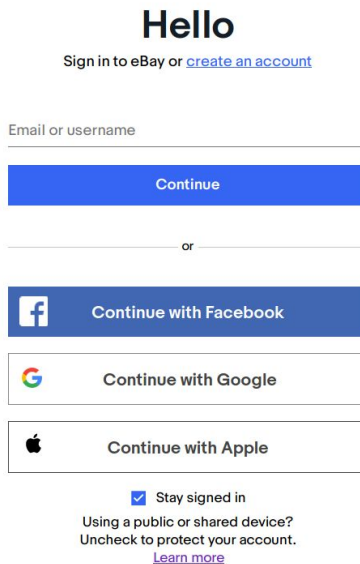
Authentifikation

“Der Nachweis einer behaupteten Eigenschaft einer Entität, beispielsweise eines Menschen.”

- Ich bin wer ich behaupte zu sein
- Drei Methoden der Authentifikation:
 - Wissen
 - Benutzername/Passwort, Geheime URL, ...
 - Besitz
 - Chipkarten, Fido-Stick, Handynummer, SSL/TLS Zertifikat, ...
 - Merkmale/Biometrie
 - Fingerabdruckscanner, FaceID, ...
- Hauptmethode im Web: Wissen zu einem Passwort
- Oft auch Kombination aus vielen (z.B. bei Two-Factor-Auth, Kontroll-Mails, ...)

Authentifikation - Login

- Klassisch über einfaches Formular
 - Input type "text" + Input type "password"
 - Gelegentlich auch in zwei Schritten
- Oft über Drittanbieter gelöst
 - Verbessert Vertrauen in eigene Seite und Nutzer
 - Technisch weniger Verantwortung
 - Erleichtert Zugang für Nutzer
- UX/Security-Minenfeld
 - Nicht zu viel Information in Rückmeldung verraten
 - "Nutzer XYZ ist unbekannt"/"Inkorrektes Passwort"
 - => "Inkorrekte Zugangsdaten"





Hello
Sign in to eBay or [create an account](#)


Email or username

Continue

or

 **Continue with Facebook**

 **Continue with Google**

 **Continue with Apple**

☒ Stay signed in
Using a public or shared device?
Uncheck to protect your account.
[Learn more](#)

Authentifikation - Das stateless Problem

- Das Web ist stateless konzipiert
 - Ohne durchgängige Verbindung zwischen Client und Server
- Wie kann Server die Client-Identität dauerhaft prüfen?
- Cookie/Session Modell
 - Geteilte Information in Client und Server

Cookies

Speichern Daten(Strings) im Client

Können vom Server aus verwaltet werden

Werden immer an Server übertragen

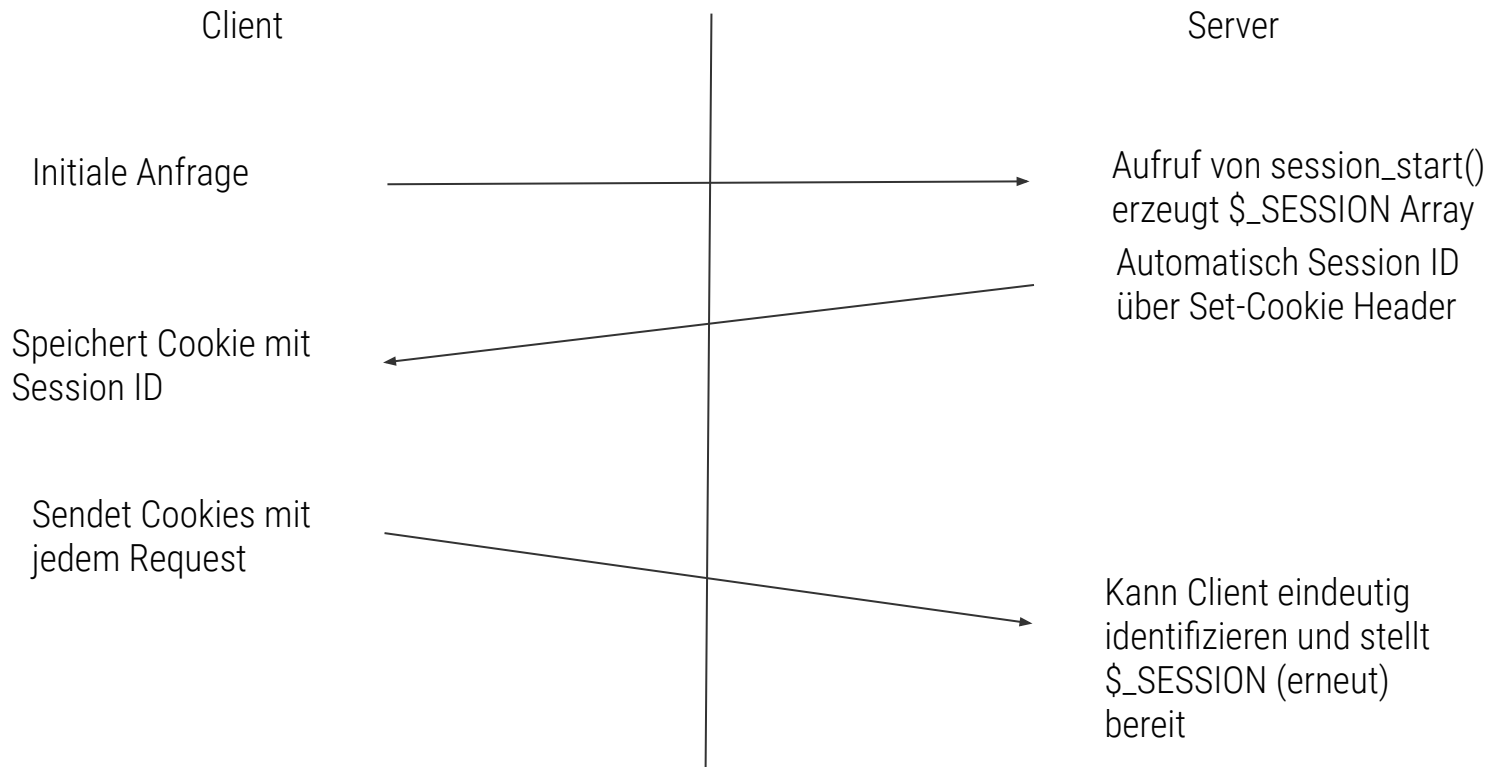
Sessions

Stateful Informationsspeicher im Server

Sollten pro User einmalig sein

Sollte Identität enthalten

Authentifikation - Cookie/Session Interaktion



Authentifikation - Umgang mit Passwörtern

- PHP bietet integrierte Funktionen zur Passwortverwaltung
 - *password_hash, password_verify, ...*
 - **Diese sollten auf jeden Fall benutzt werden!**
- Bieten moderne Hashing-Algorithmen (auch im Default)
- Erlauben die Migration von Passwörtern zu aktuelleren Algorithmen
 - Mit Hilfe von *password_needs_rehash*
- Nutzt sog. Salts
- Schützt z.B. vor Rainbow Tables und Timing Attacks
- Speichert Algorithmus, Salt und Kosten im Hash:
 - `$2y$10$.vGA109wmRjrwAVXD98HNOgsNpDczlqm3Jq7KnEd1rVAGv3Fykk1a`

Authentifikation - Login als Code

```
$session = new Session(); // Wrapper around $_SESSION magic constant
$session->start(); // wrapper around session_start() function

$username = $request->getQueryParam('username'); // coming from our form via $_POST/$_REQUEST
$password = $request->getQueryParam('password'); // coming from our form via $_POST/$_REQUEST

// test if user exists
$user = $userRepository->getByUsername($username);
$hash = null;
// user testing deferred for timing reasons
if ($user instanceof User ) {
    $hash = $user->getPassword();
}
// test if the password is correct
if (password_verify($password, $hash)) {
    // test if the password needs rehash
    if (password_needs_rehash($hash, algo: PASSWORD_DEFAULT)) {
        $user->setPassword(password_hash($password, algo: PASSWORD_DEFAULT));
        $userRepository->update($user);
    }
    // login SUCCESSFUL
    $session->set('username', $username); // setter for the $_SESSION array
}
```

Authentifikation - Überprüfen und Beenden

- Useridentifikation kommt aus Session
- Auslagern in abstrakte Basisklasse oder Funktionalität in Router

```
$session = new Session(); // Wrapper around $_SESSION magic constant
$session->start(); // wrapper around session_start() function

// test if the session variable contains an username, which is done at login
if (!$session->has( foo: 'username')) {
    // user NOT logged in, redirecting
    $this->redirect('/login'); // wrapper around the Location header
    // END REQUEST HANDLING here as redirects can be ignored by the client!
    return;
}
// user IS logged in, do stuff
```

- Logout durch Aufruf von `session_destroy()`

Authentifikation - Alternativmodelle

- Basic Auth
 - Überträgt Nutzernamen und Passwort bei jedem Request erneut
 - Gut in Browser integriert
 - Einfach aber wenig flexibel
- Auth Provider (z.B. OpenID über OAuth)
 - Lagern Login an Dritte Partei aus (z.B. "Login mit Google")
 - Können auch Nutzerdaten bereitstellen
- Tokens
 - Authentifikation über digitalen Besitz
 - Wie digitaler Schlüssel der nur bestimmte Türen öffnet
 - Erlauben leichte Weitergabe von Identität und Berechtigung
- Viele mehr (z.B. JWT)

Authentifikation - Security

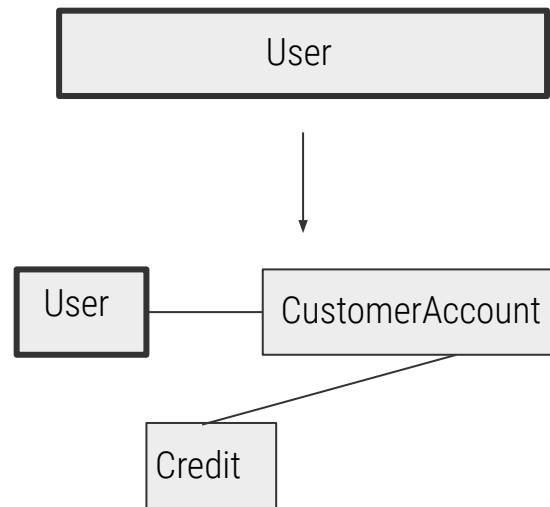
- Formulare mit sensiblen Daten **immer über POST**
- **HTTPS ist ein Muss**, zumindest beim Login
- Übertragung oder Verarbeitung von Klartext-Passwörtern auf jeden Fall vermeiden
- Cookies absichern
 - HttpOnly Flag sperrt Zugriff durch Javascript (XSS Cookie-Diebstahl)
 - Cookies signieren
- CSRF zum Schutz vor manipulierten Links für Login Formulare
 - Über unsichtbares Feld mit Zufallsstring den der Server kennt
- "Passwort vergessen" absichern
 - Über Kontrollmechanismus (z.B. Mail), nicht Sicherheitsfragen
 - Keine Aussage über Existenz des Nutzers
 - Non-Blocking (Nutzer nicht sperren)

Authentifikation - Registrierung

- Hauptweg neue Nutzer in Anwendung zu bekommen
- Sammelt alle notwendigen Informationen zum Nutzer
 - Minimal Nutzernamen
 - Passwort oft automatisch vergeben
 - E-Mail als Standard-Kontaktmöglichkeit
- Oft in mehrere Schritte getrennt
 - Erstellen von Konto über wenig Daten
 - Bei Nutzung von Diensten mehr Informationen (z.B. Realname, Adresse,...) nachtragen
- Registrierung immer über Zweitmechanismus z.B. Kontrollmail absichern!
 - Erzwingt gewissen Aufwand und erlaubt etwas Sicherheit

Speichern von Accounts - Datenschutz

- Nutzer werden wie andere Entitäten/Models in der Datenbank gespeichert
- Accountdaten haben potentiell sensiblen Inhalt (nicht nur Passwörter)
 - Datenschutz und -sicherheit wichtiges Thema
 - DSGVO im Hinterkopf behalten
- Gut durchdachtes Datenbankmodell sinnvoll
 - Z.B. Abtrennung von Nicht-Login Daten in eigene Entitäten
 - **Datensparsamkeit** üben (nur speichern was man wirklich braucht)
- Verhindert das "Herumreichen" von sensiblen Daten
 - Innerhalb der Anwendung
 - Über (Web-) Schnittstellen



Speichern von Accounts - Datenhoheit

- Nutzung von Daten muss durch Nutzer bestimmbar sein
 - Privacy by Design
- Nutzer muss Verwendung seiner Daten
 - Nachvollziehen können
 - Jederzeit ablehnen können
- Nutzer sollte seine Daten selbst verwalten können
 - Z.B. Passwort ändern, Nachname ändern
- Nutzer muss seinen Account restlos löschen können
 - Achtung! Eventuell gibt es rechtliche Aufbewahrungsfristen

Autorisierung

“Die Zustimmung oder Erlaubnis, spezieller die Einräumung von Rechten [...], gegebenenfalls als Nutzungsrecht gegenüber Dritten. ”

- Meistens: Ich darf bestimmte Dinge tun
- Erst notwendig wenn Nutzer in Berechtigungen unterschieden werden müssen
 - Z.B. es Admins geben soll
- Je nach Umfang und Detailgrad sehr komplex
- Im Standard möglichst wenig Rechte an möglichst viel Nutzer
 - Principle of least Privilege (PoLP) bzw. secure by default

Autorisierung - Rechtemodell

- Es kann unterschieden werden zwischen
 - **Wer** darf etwas
 - **Was** darf er
 - **Womit/Worauf** darf er es
 - Z.B. Nutzer XYZ darf Bankdaten anderer Nutzer lesen und verändern
- Über sog. Access Control Lists (ACL) auf Objektebene überprüfbar
 - Enthalten sequenzielle Liste von Rechten
 - Greifen auf Schnittstellenebene
 - In Repositories für Models
 - Im Router für Controller/Routen
- Abstraktion des “wer” über Rollen ist gängige Praxis
 - Nicht Nutzer darf, sondern Rolle darf und Nutzer hat Rolle
 - Bei unterschiedlichen oder pflegbaren Berechtigungen, z.B. Admin, Redakteur, normaler Nutzer, ...

Autorisierung - Überprüfen

```
// user IS logged in, do stuff
$user = $userRepository->getByUsername($session->get('username'));
if (!$user instanceof User || !$user->isAdmin()) {
    throw new ForbiddenException(
        sprintf(
            format: 'The user %s has insufficient rights for the %s route',
            $user->getUsername(),
            $this->getRoute()
        )
    );
}
// we got an admin, go on
```

```
*/
public function isAdmin()
{
    foreach ($this->getRoles() as $role) {
        if ($role->isAdmin()) {
            return true;
        }
    }
    return false;
}
```

Quellen:

- <https://de.wikipedia.org/wiki/Authentifizierung>
- <https://de.wikipedia.org/wiki/Authentifizierung>
- <https://www.php.net/manual/en/function.password-hash.php>
- <https://blog.risingstack.com/web-authentication-methods-explained/>
- <https://www.techrepublic.com/article/understanding-and-selecting-authentication-methods/>
- https://en.wikipedia.org/wiki/Access-control_list

Bildquellen:

-