

WEBBASIERTER-DATENBANKMANAGER

<https://github.com/miwied/web-based-database-manager>

Gruppe 4 – Fusilli

Thomas S. @Noodle693

Marco H. @GodlesZ95

Samuel S. @Gunzli02

Michael W. @miwied

Juli – 2022

Inhaltsverzeichnis

Herangehensweise	3
Herausforderungen	3
Zeitplan	3
Aufgabenverteilung	4
Projekt Architektur	5
Docker	6
Login (Backend)	7
JSON Web Token (Backend)	8
Backend-Architektur	13
Authentifizierung	16
JSON Web Token (Frontend)	17
Austausch der Daten	17
Validierung	18
Angular	19
Angular-Material	19
Tests	20
Installation	22

Herangehensweise

- Warum haben wir uns für Angular entschieden?
 - o Neben der Frage, ob wir das ganze klassisch mit HTML und CSS umsetzen, kam uns zusätzlich die Frage auf, ob wir stattdessen ein Framework verwenden wollen. Nach einer Diskussionsrunde haben wir uns für Angular entschieden. Hauptgrund war, dass alle Projektteilnehmer bereits erste Erfahrung mit diesem Framework hatte und das dieses Projekt auch im Fokus stand, weitere Erfahrungen in Angular sammeln zu können.
- Versionierung
 - o Damit sich die Möglichkeit bietet, dass jeder unabhängig des anderen an dem Projekt arbeiten und Programmstände einfach zusammengeführt werden können, haben wir uns entschieden, das Projekt auf GitHub hochzuladen und somit Git als Versionierungstool zu benutzen.
- Deployment
 - o Da wir eine einfache Installation und eine effektive Entwicklungsinfrastruktur gewährleisten wollten, haben wir uns entschieden, alle Dependencies in Docker laufen zu lassen. Dadurch ist es möglich, durch einen einzigen Command die Anwendung zum Laufen zu kriegen.

Herausforderungen

Nicht alle von uns gesteckten Ziele konnten aufgrund des Zeitmangels umgesetzt werden:

- Prüfen, ob eine Session abgelaufen ist
- Einen 100%igen Restful Service implementieren.
- Aneignen des gleichen Wissenstand in allen Gebieten durch alle Teilnehmer.
- Viel Zeit ging auch durch das Erlernen neuer Technologien, wie z. B. Docker, PHP verloren

Zeitplan

Ca. 30% des Projekts wurde in der Berufsschule erledigt, der Rest musste in der Freizeit fertiggestellt werden.

Aufgabenverteilung

Developer:

Frontend	Backend
Samuel S.	Thomas S.
Marco H.	Michael W.

Samuel S. – Frontend Styling

Implementierung des Designs von Login und Hauptseite, sowie Styling von Elementen wie Header und Footer

Marco H. – Frontend Logik

Authentifizierung, Kommunikation der Komponenten, Anlegen und Logikimplementierungen der Services, Login, Models und teilweise Komponentenbuilding, sowie auch deren Logikimplementierung

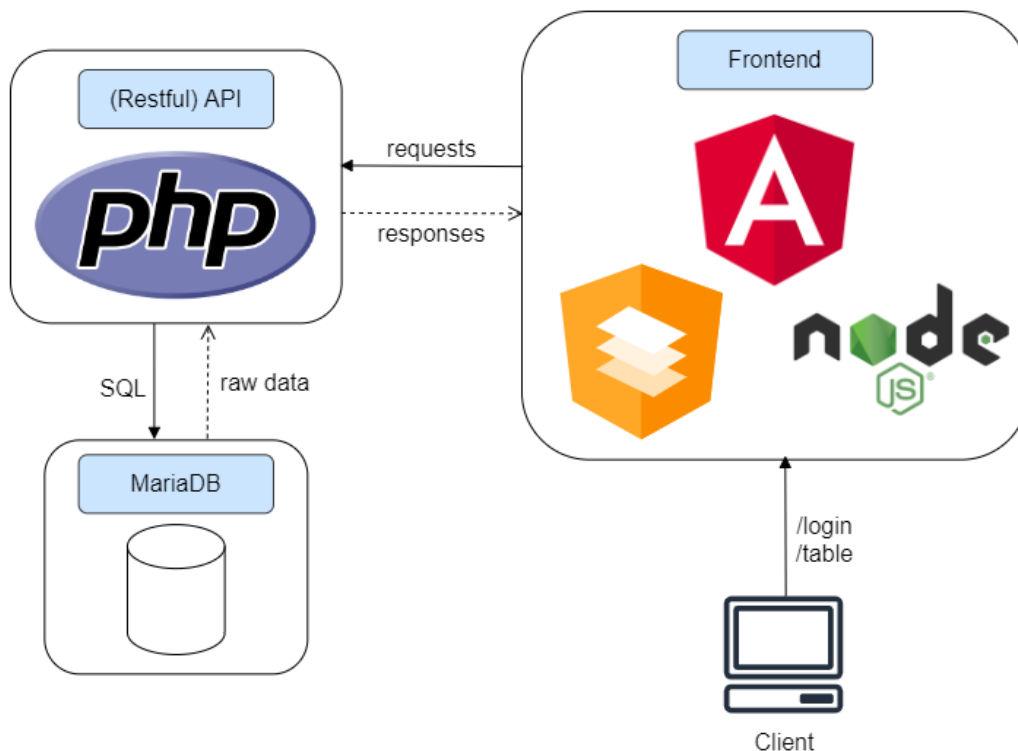
Thomas S. – Backend

Grundlegende Backend-Architektur nach MVC-Pattern, Ausarbeitung des Datenbankzugriffs mit PDO, Anlegen und Ausarbeiten der Repository Funktionen vorallem für den MemberController (+Mapping), JWT Generierung und Abgleichung, CORS Implementierung, Implementierung von Basisklassen und Methoden (bspw. für http-Errors), Änderung der Datenbank (ON CASCADE delete), Troubleshooting Docker

Michael W. – Backend

Speicherung und Validierung der Login Daten (inklusive Passwort salting, peppering und hashing), Anbindung der Login-Daten mit der Datenbank, Implementierung von Basisklassen und Methoden (bspw. für http-Errors), Anlegen des Github-Repos, Anlegen von Dokumentationsdateien (im .md Dateiformat), Einrichtung von Docker (docker-compose + dockerfiles)

Projekt Architektur



Das Frontend besteht aus einer Angular Applikation bestehend aus Komponenten des Material-Designs. Node.js beinhaltet den Node-Package-Manager (NPM), welcher die notwendigen Pakete für die Applikation installiert und bereitstellt.

Das Backend (implementiert in php) bezieht die Daten direkt aus der Datenbank (MariaDB) welche mithilfe von PDO (Hilfslibrary für Datenbankzugriff) in strukturierte Objekte aufgeteilt werden, die jeweiligen Objekte werden dann in das JSON-Format kodiert und bei einer entsprechenden Anfrage mittels einer HTTP-API an das Frontend kommuniziert.

Docker

Docker ist eine Open-Source Software zur Isolierung von Anwendungen mit Hilfe von Containervirtualisierung.

Docker vereinfacht die Bereitstellung von Anwendungen, weil sich Container, die alle nötigen Pakete enthalten, leicht als Dateien transportieren und installieren lassen.

Container gewährleisten die Trennung und Verwaltung der auf einem Rechner genutzten Ressourcen.

In unserem Projekt nutzen wir Docker, um die Anwendung nicht nur später für den Endverbraucher einfach installierbar zu machen, sondern auch um die Entwicklung effektiver und effizienter zu gestalten.

Mithilfe von Docker hat jeder Entwickler die gleichen Voraussetzungen.

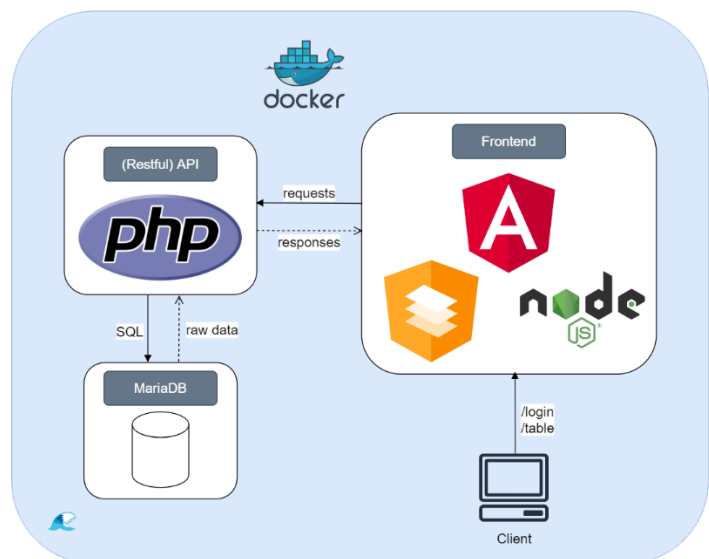
Dinge wie PHP Webserver, Maria DB- und Angular-Hosting werden mithilfe von Docker bereitgestellt und müssen somit nicht über externe Programme / Lösungen zur Verfügung gestellt werden.

Unsere Infrastruktur wird somit in Docker ausgeführt und stellt sich aus folgenden Containern zusammen:

Angular-App
Zum Bauen der Angular App, mithilfe von node.js

MariaDB
Datenbank für Sportverein- und Login-Daten

PHP-Apache
Webserver für das Backend



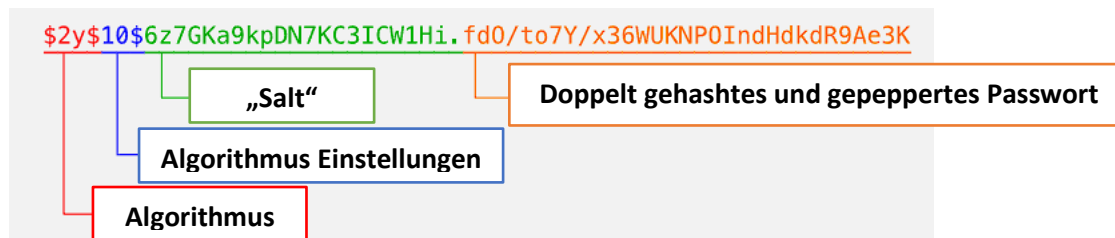
Login (Backend)

Registrierung

Bevor man sich anmelden kann, muss erstmal ein Nutzer angelegt bzw. registriert werden. Über das Frontend wird ein POST-Request mit dem Nutzernamen und dem Passwort als Parameter ausgeführt, hierzu wird folgende Route verwendet:

```
http://localhost/login.php/createUser?username=USERNAME&password=PASSWORD
```

Das Passwort wird dann mit der Funktion „hash_mac()“ und einem Konfigurierbaren „pepper“ mit dem „sha256“ Algorithmus hashed, anschließend wird dann das gehashte + gepeppte Passwort mit der „password_hash()“ Funktion nochmals gehashed und „gesalted“. Das Resultat, welches in der Datenbank gespeichert wird, sieht wie folgt aus:



Anmeldung

Will sich ein Nutzer anmelden muss dieser über das Frontend seinen Nutzernamen und das Passwort eingeben, das Frontend sendet nun einen POST-Request mit dem eingegebenen Nutzernamen und dem Passwort an das Backend und nutzt dafür diese Route:

```
http://localhost/login.php/getToken?username=USERNAME&password=PASSWORD
```

Das eingegebene Klartext-Passwort wird nun wieder gehashed + gepeppter und anschließend mit dem Passwort-Hash aus der Datenbank mithilfe der „password_verify()“ Funktion verglichen, bei Richtigkeit wird ein JSON-Webtoken erstellt und der HTTP-Response übergeben.

Schutz durch pepper und salt

Sollte ein Angreifer zugriff zur Datenbank bekommen, wird dieser keinerlei Möglichkeiten haben die Passwörter wieder in Klartext zurückzuführen.

Aufgrund der Kombination aus gepepten und gesalteten Passwörter sind Vergleiche aus einer „Rainbow Tabelle“ nutzlos, auch Brute-Force- oder Dictionary-Attacken sind nahezu unbrauchbar bei dieser Passwortspeicherungs-Methode.

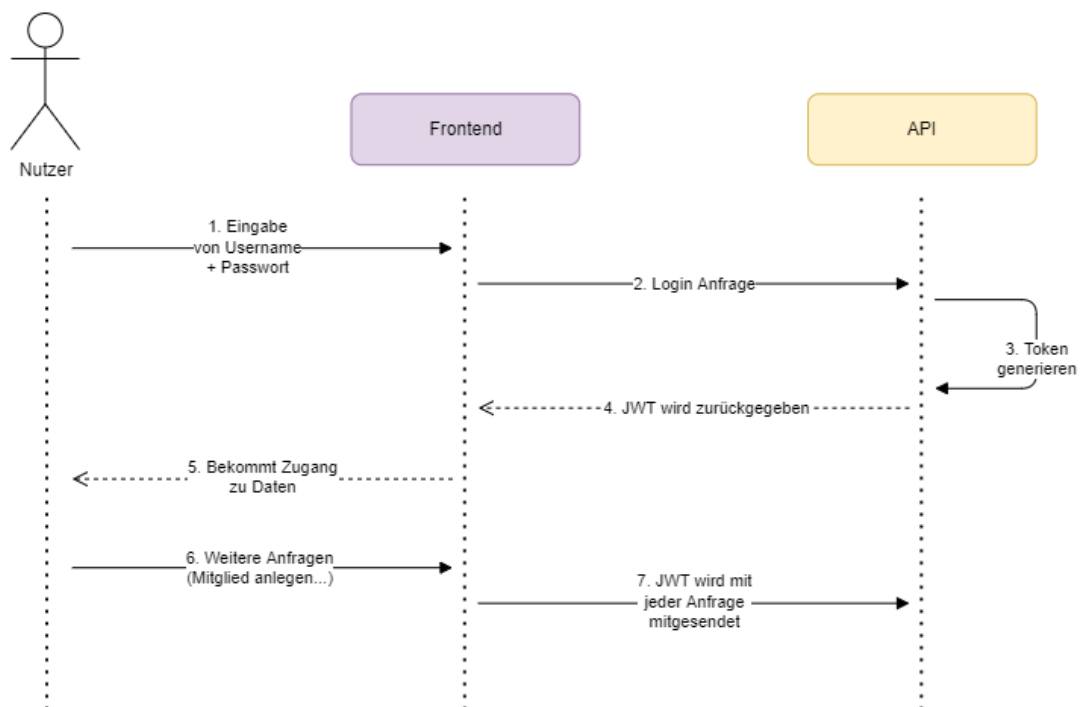
JSON Web Token (Backend)

Wir nutzen für die Autorisierung aller HTTP Anfragen, über die wir Daten von der API nach außen geben, einen JSON Web Token, der beim erfolgreichen Login generiert wird.

Wir gleichen die Gültigkeit dieses Tokens bei jeder Anfrage ab, damit ein unautorisierter Zugriff auf die personenbezogenen Daten unmöglich ist.

Das Generieren und der Abgleich finden über das Firebase/PHP-JWT Package statt. Wir haben uns für dieses Package entschieden, da es genau diese Funktionalität für unser Projekt bereits enthält.

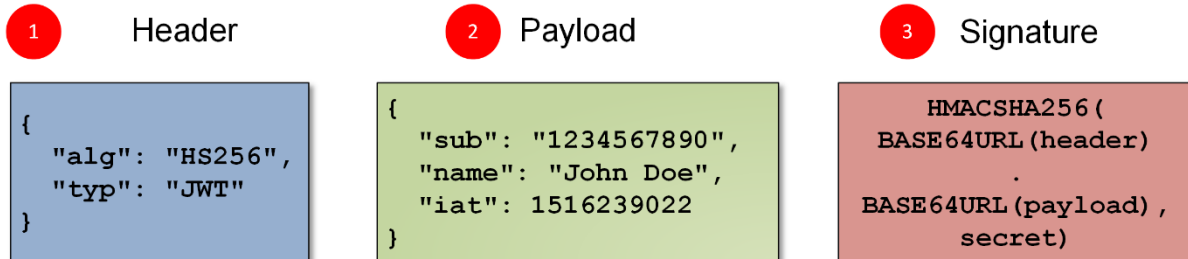
Dieses Diagramm soll erstmal den Ablauf visualisieren. Im Anschluss gehen wir genauer auf die Generierung, den Aufbau des Tokens und die technische Abfrage ein.



Zunächst erklären wir den Aufbau des JWT – im Anschluss beleuchten wir den 3. Schritt im Diagramm, die Generierung, und dann die Abfrage bevor die HTTP Anfrage weiter verarbeitet wird.

Aufbau

1 eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjMONTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.XbPfbIHMI6arZ3Y922BhjWgQzWXcXNrZ0ogtVhfEd2o 2 3



Der JWT besteht aus 3 verschlüsselten Teilen, die durch Punkte getrennt sind.

1. Zunächst die Header-Information worin der Typ des Tokens sowie der Verschlüsselungsalgorithmus enthalten sind. Wie Sie später sehen werden, verwenden wir selbst den HS512 Algorithmus – dies steht für HMAC SHA512, also die SHA512 Hashfunktion in Verbindung mit einem Nachrichtenauthentifizierungscode.
2. Die Payload. Hier finden sich selbst festgelegte sowie allgemein empfohlene „Claims“ (key-value-pairs). Alle Zeitpunkte sind von Beginn der Unixzeit an in Sekunden gerechnet angegeben. In kurz: die Anzahl der Sekunden seit dem 01.01.1970 um 00:00 Uhr UTC.
Wir senden also den Zeitpunkt, wann der Token generiert wurde (iat – issued at), den Herausgeber des Tokens (iss – issuer), das Datum ab wann der Token gültig ist (nbf – not before), wann er abgelaufen ist (exp – expiration) und den Nutzernamen im Klartext (userName).
3. Die Signatur der gesamten Message, welche verifiziert, dass die Nachricht während dem Senden nicht verändert wurde. Hier wird das Paket aus Header, Payload und einem eigens festgelegten Secret-Key verschlüsselt (bei uns ebenso mit HS512).



Generierung

Die Generierung sieht wie folgt aus:

```
67 $date = new DateTimeImmutable(); // get current dateTime
68 $expire_at = $date->modify(JWT_MODIFIER)->getTimestamp();
69 $request_data = [
70     'iat' => $date->getTimestamp(),           // Issued at: time when the token was generated
71     'iss' => JWT_DOMAIN_NAME,                 // Issuer
72     'nbf' => $date->getTimestamp(),           // Not before: time before which the token is invalid
73     'exp' => $expire_at,                     // Expiration: time after which the token is invalid
74     'userName' => $this->username,            // Username
75 ];
76 $token = JWT::encode(
77     $request_data,
78     JWT_SECRET_KEY,
79     'HS512'
80 );
81 HttpExtensionMethods::sendOutput(200, json_encode($token));
```

Zunächst holen wir uns das aktuelle Datum (Z.67). Mit `getTimestamp()` können wir den Zeitstempel in Sekunden seit Beginn der Unixzeit abrufen (Z.70+72). Das Ablaufdatum legen wir mit der globalen Variable `JWT_MODIFIER` (Z.68) fest, welche wir genau wie `JWT_DOMAIN_NAME` (Z.71) und `JWT_SECRET_KEY` (Z.78) in der `config.php` definieren und deren Werte festlegen.

```
backend > config.php > ...
...
1  <?php
2  define("DB_HOST", "mariadb");
3  define("DB_USERNAME", "root");
4  define("DB_PASSWORD", "root");
5  define("DB_DATABASE_NAME", "2021sportverein");
6  define("DB_PORT", 3306);
7  define("JWT_SECRET_KEY", "gdajgpdjogjdsklgjsdgsd3419385923tajgfdak");
8  define("JWT_MODIFIER", "+1440 minutes"); // add time to token expiration
9  define("JWT_DOMAIN_NAME", "sportverein");
10 define("PEPPER", "c1isvFdxMDdmjOlvxpecFw"); // pepper for password
```

Die Variable `$request_data` ist ein assoziatives Array, welches wie im Aufbau auf der vorherigen Seite erwähnt, den issued at Zeitstempel, den issuer selbst, das not before Datum, das expiration Datum sowie den Nutzernamen enthält.

Wie bereits erwähnt enkodieren wir die `$request_data` und den `JWT_SECRET_KEY` mit dem HMAC SHA512 Algorithmus. Am Ende bekommen wir einen Token, der wie folgt aussieht:

```
"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiJ9.eyJpYXQiOiJlY2NTc3MTU5NTcsImZlcyI6InNwb3J0dmVhZlUuIiwibmJmIjojNjU3NzE1OTU3LCJleHAiOiJlY2NTc4MDIzNTcsInVzZXJ0YW1lIjojYWRtaW4ifQ.1j0KVfDLzEkr3DuhWxeRIl-23devOB6w9nmOwtjZ0nAXKZ1r_Bga2YIYdiqFYlXqwjxROB27BAG6Qt65J9FW5w"
```

Abgleichung

Der Einstiegspunkt in unsere Applikation ist die Index.php, welche auch bei jeder Backend-Route angegeben wird (dazu mehr in der Sektion zu Routen).

Hier wird eine statische Funktion checkJWT() in der accessControl.php aufgerufen.

```
backend > index.php > ...
miwied, 3 days ago | 4 authors (Michael Wiederkehr and others)
1  <?php
2
3  require __DIR__ . "/bootstrap.php";
4
5  AccessControl::setCORS();
6
7  AccessControl::checkJWT();
```

Diese prüft zunächst ob in den HTTP Request Headern der Header „Authorization“ gesetzt ist.

```
34  // JWT checking
35  // is 'Authorization' header present?
36  if (!array_key_exists('Authorization', getallheaders())) {
37      header('HTTP/1.1 400 Bad Request');
38      echo 'No Authorization header found';
39      exit;
40  }
```

Ist dieser gesetzt so überprüfen wir per Regex-Matching, ob der Wert des Headers „Bearer“ beinhaltet und fügen diesen einem Array (\$matches) hinzu.

```
42  // does the 'Authorization' header content match the expected format `Bearer ...`?
43  // if it does contain `Bearer ...` it will fill the $matches array split by the whitespace
44  if (!preg_match('/Bearer\s(\S+)/', getallheaders()["Authorization"], $matches)) {
45      header('HTTP/1.1 400 Bad Request');
46      echo 'Token not found in request';
47      exit;
48  }
```

Mit \$matches[1] können wir auf den Wert nach dem Whitespace hinter „Bearer“ zugreifen. Dies ist unser JWT.

Wir versuchen diesen mit dem JWT_SECRET_KEY und dem Algorithmus zu dekodieren.

```
50  // $matches[0] == 'Bearer' and $matches[1] == JWT which we need to continue
51  $jwt = $matches[1];
52
53  // we decode the JWT by providing the hashed JWT, the secret key and the hashing algorithm
54  // the function also checks the iat, nbf and exp dates with the current date
55  try {
56      $token = JWT::decode($jwt, new key(JWT_SECRET_KEY, 'HS512'));
57  } catch (Exception $e) {
58      header('HTTP/1.1 400 Bad Request');
59      echo ('Token decoding failed - see exception for details: ' . $e->getMessage());
60      exit;
61  }
```

Das PHP-JWT package überprüft dann die verschiedenen Claims, die wir unserem Token in der \$request_data gegeben haben.

Zunächst wird validiert, dass der Token überhaupt schon gültig ist (nbf Claim) und ob der Token vor dem jetzigen Zeitpunkt erstellt wurde.

```
143 // Check the nbf if it is defined. This is the time that the
144 // token can actually be used. If it's not yet that time, abort.
145 if (isset($payload->nbf) && $payload->nbf > ($timestamp + static::$leeway)) {
146     throw new BeforeValidException(
147         'Cannot handle token prior to ' . \date(DateTime::ISO8601, $payload->nbf)
148     );
149 }

151 // Check that this token has been created before 'now'. This prevents
152 // using tokens that have been created for later use (and haven't
153 // correctly used the nbf claim).
154 if (isset($payload->iat) && $payload->iat > ($timestamp + static::$leeway)) {
155     throw new BeforeValidException(
156         'Cannot handle token prior to ' . \date(DateTime::ISO8601, $payload->iat)
157     );
158 }
```

Wenn der Token soweit zeitlich akzeptiert werden kann wird überprüft, ob das Expiration Datum, welches wir mit +1440 Minuten auf einen Tag festlegen, nicht überschritten wurde.

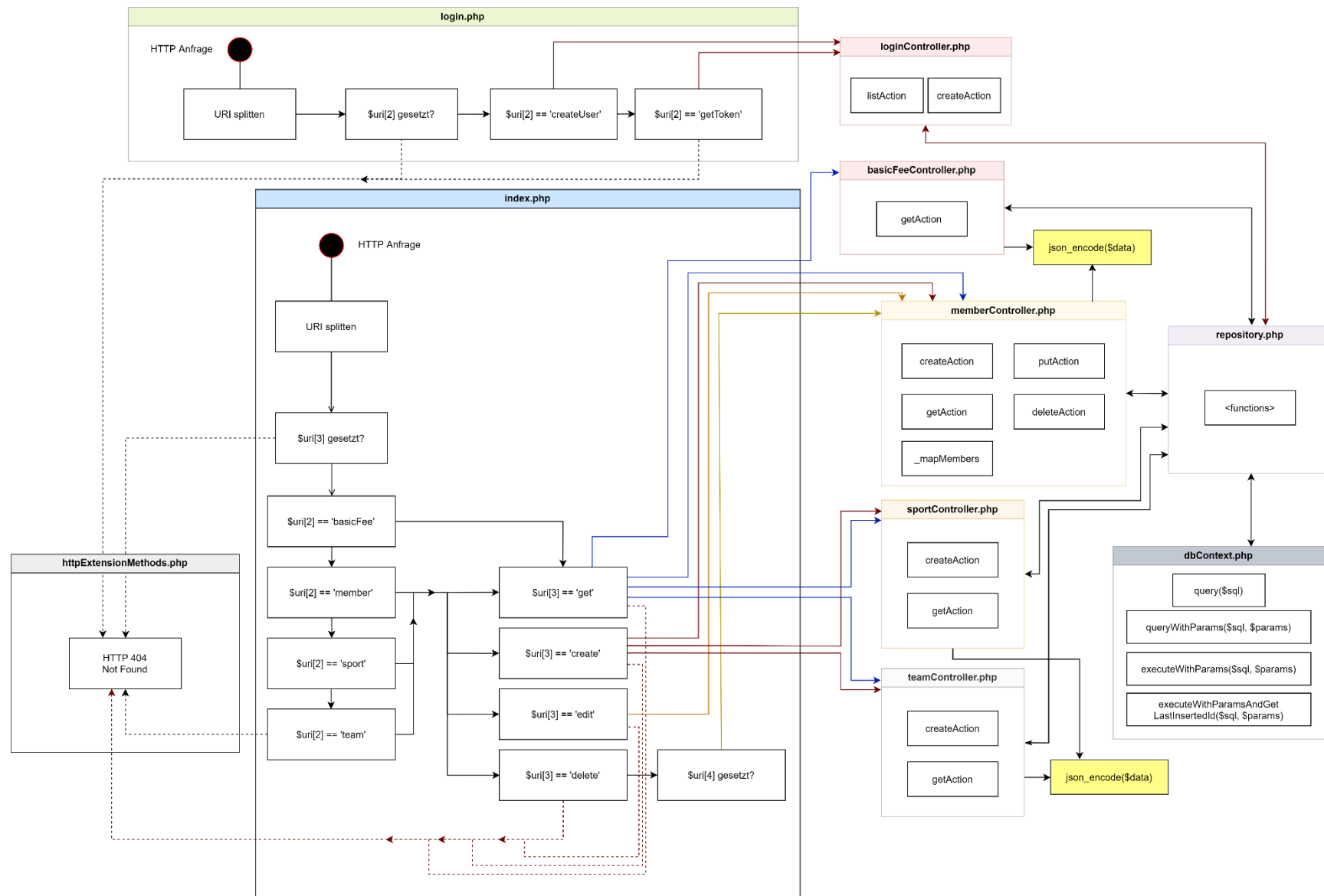
```
160 // Check if this token has expired.
161 if (isset($payload->exp) && ($timestamp - static::$leeway) >= $payload->exp) {
162     throw new ExpiredException('Expired token');
163 }
```

Zuletzt wird (wieder in der accessControl.php) der Herausgeber mit unserer JWT_DOMAIN_NAME Variable abgeglichen.

```
63 // check if the token issuer matches the domain name we set before
64 // if the check matches the user is unauthorized and can't continue
65 if ($token->iss !== JWT_DOMAIN_NAME) {
66     header('HTTP/1.1 401 Unauthorized');
67     exit;
68 }
```

Nach all diesen Checks ist der Token valide und die Anfrage kann weiter verarbeitet werden.

Backend-Architektur



Bei der Weiterverarbeitung der HTTP-Anfrage in der Index.php wird zunächst CORS (Cross-Origin Resource Sharing) erlaubt, da sich unser Frontend auf einem anderen Port befindet und die API die Anfrage sonst verweigern würde.

Wird teilen die URI anhand der Slashes ,/' auf und lesen die angesprochene Route anhand der URI-Segmente aus.

Sollte keiner unserer Endpunkte angesprochen werden werfen wir einen HTTP 404 Error.

Im nächsten Schritt wird für den jeweiligen Endpunkt der Controller initialisiert und eine bestimmte Methode aufgerufen.

Der Controller überprüft die Anfrage und ruft das Repository mit seiner jeweiligen Funktion auf.

Das Repository greift auf den DBContext zu, welcher per PDO auf die Datenbank zugreift und holt sich die benötigten Daten oder führt eine Anweisung aus.

Beispiel:

➔ <http://localhost/index.php/member/get>

/member-Route und /get-Endpunkt werden erkannt, `getAction()` vom MemberController wird aufgerufen.

Nachdem die Anfrage auf die HTTP-GET Methode überprüft wurde wird im Repository ein SQL-Statement vorbereitet und losgeschickt.

```
52 public function getMembers()
53 {
54     $sql = "SELECT * FROM mitglied";
55     return $this->db->query($sql);
56 }
```

```
11 public function query($sql)
12 {
13     $stmt = $this->pdo->prepare($sql, array(PDO::ATTR_CURSOR => PDO::CURSOR_SCROLL));
14     $stmt->execute();
15     $result = $stmt->fetchAll(PDO::FETCH_ASSOC);
16     return $result;
17 }
```

Im DBContext geben wir an, dass alle Zeilen aus der Tabelle nach und nach eingelesen werden sollen und in ein Array aus assoziativen Arrays gespeichert werden sollen.

Als Sonderfall beim Abrufen aller Mitglieder führen wir noch ein Mapping durch damit wir für das Model was an das Frontend gesendet wird auch jegliche Information aus den Beziehungstabellen eingelesen haben.

Alle anderen Anfragen enden nach dem Datenbankzugriff und geben die Information als JSON-Objekt nach außen.

Wir haben uns primär für das Mapping entschieden um weniger Endpunkte ansprechen zu müssen, wenn wir die Tabelle laden, und damit wir im Frontend für das Objekt bis zum Submit keine weiteren Anfragen losschicken müssen, wenn wir es editieren wollen.

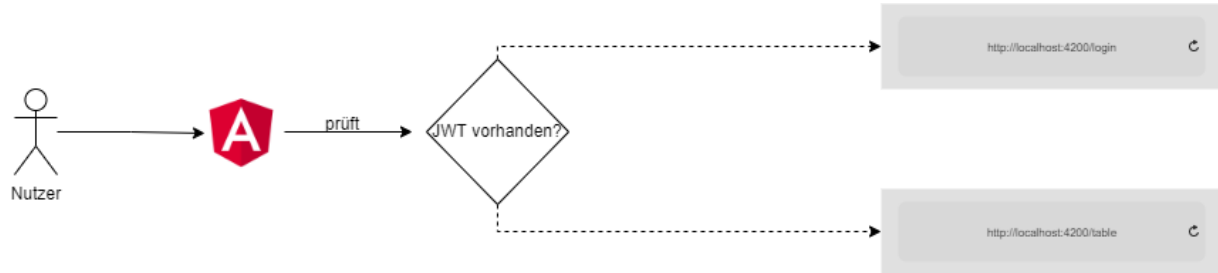
Hier noch ein kleiner Ausschnitt von dem Mapping was wir durchführen:

```
foreach ($input as $k => $value) {  
    // get sports info  
    $sportsIds = $this->repo->getSportsByMemberId($value['mi_id']);  
    $sports = array();  
    foreach ($sportsIds as $key => $id) {  
        array_push($sports, $this->repo->getSportsInfoById($id['sa_id']));  
    }  
}
```

Hier wird anhand der ID des Mitglieds sowohl ein Array mit allen IDs der Sportarten sowie eines mit allen Namens- und Beitragsinformationen aus der Datenbank abgerufen und im weiteren Code auf ein Objekt gemapped. Dies sieht final so aus:

```
$tmp = (object) [  
    'memberId' => $value['mi_id'],  
    'firstName' => $value['vorname'],  
    'lastName' => $value['nachname'],  
    'zipCode' => $value['plz'],  
    'city' => $value['ort'],  
    'gender' => $value['geschlecht'],  
    'feeId' => $value['gb_id'],  
    'feeGroup' => $feeGroup,  
    'fee' => $fee,  
    'sportIds' => $sportsIds,  
    'sports' => $sports,  
    'isPlayer' => $isPlayer,  
    'playerTeamId' => $playerTeamId,  
    'playerTeamName' => $playerTeamName,  
    'isTrainer' => $isTrainer,  
    'trainerTeamId' => $trainerTeamId,  
    'trainerTeamName' => $trainerTeamName  
];
```

Authentifizierung



Vorgehensweise

Der Nutzer kann nur auf die Daten zugreifen, wenn er sich vorher mittels Log-in authentifiziert und dadurch einen JWT erhalten hat.

Der Token wird per Local-Storage in der Anwendung abgelegt und bleibt dort so lange erhalten, bis er gelöscht oder die Anwendung beendet wird.

```
setToken(token: string) {  
  localStorage.setItem('token', token);  
}
```

Anschließend wird beim Aufrufen der Route geprüft, ob der User eingeloggt ist.


```
isLoggedIn() {  
  const usertoken = this.getToken();  
  if (usertoken !== null) {  
    return true;  
  }  
  return false;  
}
```

Wenn kein Token vorhanden ist, wird der User wieder auf die Login-Seite weitergeleitet.

JSON Web Token (Frontend)

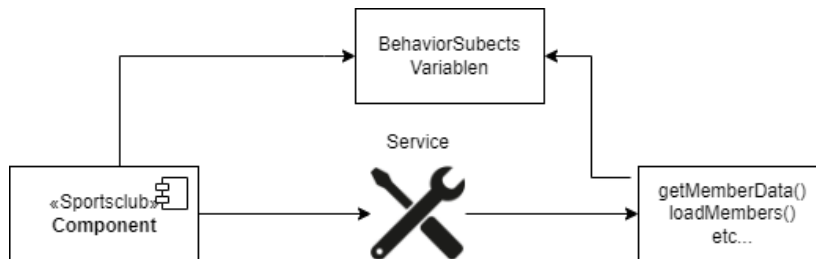
Neben der Authentifizierung wird der JSON Web Token außerdem dafür verwendet, den Namen aus dem Token zu extrahieren und im Frontend anzuzeigen

```
ngAfterContentInit() {  
  const helper = new JwtHelperService();  
  var decodedToken = helper.decodeToken<any>(  
    this.apiService.getToken()?.toString()  
  );  
  this.tokenUserName = decodedToken.userName;  
}
```

Eingeloggt als: Marco1 

Mithilfe des JwtHelperService Packages kann der Token decodiert werden. Danach kann der Name ausgelesen und angezeigt werden.

Austausch der Daten



In Angular arbeiten wir mit Komponenten. Diese beinhalten das HTML-Gerüst und die Logik in Form von Typescript. Zusätzlich haben wir verschiedene Services, die zum einen als Repository die Daten im Backend abrufen und zum anderen dem Datenaustausch intern dienen, integriert. Im Kern stehen hier sogenannte BehaviorSubjects und Observables.

Observable, Subject und BehaviorSubject stammen aus dem RxJS Framework. Diese Klassen sind nicht Bestandteil von JavaScript und Angular. BehaviorSubject erbt von Subject und das erbt von Observable. Alle haben den gleichen Feature-Umfang wie ein Observable.

Ein Subject speichert das letzte Event, welches über das Observable verschickt wurde. Ein Zugriff auf das letzte Event ist in einem Observable nicht möglich.

Ein BehaviorSubject hat zusätzlich einen initialen Wert, der zurückgeliefert wird, wenn noch kein Event verschickt wurde.

Validierung

Damit die Integrität der Daten gewährleistet wird, ist es im Frontend nur möglich, valide Anfragen an das Backend zu schicken. Angular gibt uns hierbei die Möglichkeit, mithilfe der FormControl individuelle Validierungsregeln für einzelne Eingabefelder zu definieren. So können z. B. bei der PLZ lediglich Zahlen eingegeben werden.

```
<mat-form-field appearance="outline" name="zipCode">
  <mat-label>PLZ</mat-label>
  <input matInput formControlName="zipCode" placeholder="95448" />
  <mat-error *ngIf="addMemberForm.get('zipCode')?.errors">
    Genau <strong>5 Zahlen</strong> benötigt
  </mat-error>
</mat-form-field>
```

```
zipCode: new FormControl(null, [
  Validators.pattern('^[0-9]*$'),
  Validators.minLength(5),
  Validators.maxLength(5),
]),
```

<div>Gabi</div> <div>1337</div> <div>Mind. 3 Buchstaben und keine Zahlen</div>	<div>Ziegler</div> <div>1337</div> <div>Mind. 3 Buchstaben und keine Zahlen</div>	<div>95326</div> <div>ABCDE</div> <div>Genau 5 Zahlen benötigt</div>
<div>Kulmbach</div> <div>1337</div> <div>Mind. 3 Buchstaben und keine Zahlen</div>	<div>Weiblich</div> <div>Geschlecht</div>	<div>Erwachsene</div> <div>Beitragsgruppe</div>

Angular

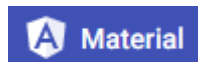


Angular ist ein Entwicklungsframework, welches auf TypeScript basiert und ein paar Vorteile bietet:

- Es ist ein komponentenbasiertes Framework für die Erstellung skalierbarer Webanwendungen
- Eine Sammlung gut integrierter Bibliotheken, die eine Vielzahl von Funktionen abdecken, darunter Routing, Formularverwaltung, Client-Server-Kommunikation und mehr
- Eine Reihe von Entwicklertools, die Sie beim Entwickeln, Erstellen, Testen und Aktualisieren Ihres Codes unterstützen

Es wurde von Google entwickelt, und wird durch Google sowie die Online-Community weiterentwickelt.

Angular-Material



Angular Material ist eine UI-Komponentenbibliothek, die Entwickler in Angular Projekten verwenden können, um die Entwicklung von eleganten Benutzeroberflächen zu beschleunigen. Angular Material bietet wiederverwendbare und schöne UI-Komponenten wie Karten, Eingaben, Datentabellen, Datepicker und vieles mehr.

Tests

Test #1

Der erste Test prüft den Login durch Eingabe richtiger Login-Daten.

Erwartung:

Durch das Eingeben existenter und richtiger Login-Daten soll bei dem Submit die neue Seite mit der Tabelle angezeigt werden.

localhost:4200/login

Login

Register

test

....

Submit

localhost:4200/table

Realität:

Das erwartete Szenario tritt ein.

Test #2

Der zweite Test prüft, ob der Login den Zugriff verweigert, wenn falsche Daten eingegeben werden.

Erwartung:

Durch das Eingeben falscher oder nicht existenter Login-Daten, soll die Seite eine Fehlermeldung zeigen, welche aufzeigt, dass die Daten nicht übereinstimmen.

Nutzername oder Passwort ungültig

X

Login

Register

test

.....

Submit

Realität:

Das erwartete Szenario tritt ein.

Test #3

Der dritte Test testet, ob unser Routing verhindert, dass man solange man den Link zu der Seite mit der Tabelle weiß, den Login umgehen kann.

Erwartung:

Wenn man den Link zur nächsten Seite eingibt, wird der Zugriff auf die Seite verhindert und man wird wieder auf die Loginseite geroutet.

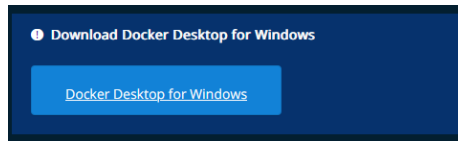


Realität:

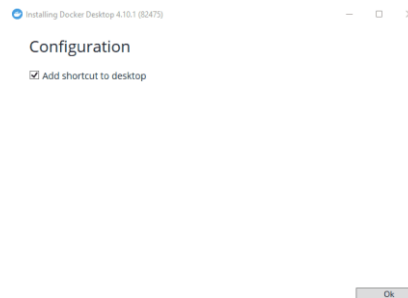
Das erwartete Szenario tritt ein.

Installation

1. [Docker Desktop herunterladen und installieren](#) (Auf Link klicken + Strg gedrückt halten)

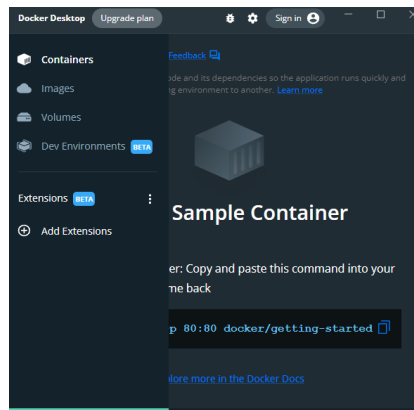


2. Installation starten und nach Installationsanweisung installieren

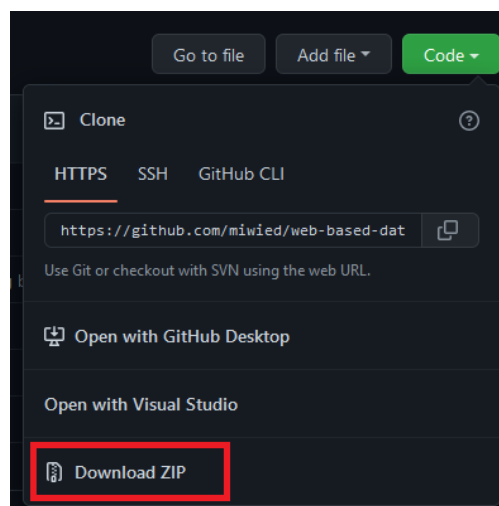


3. Docker Desktop starten -> Lizenzvereinbarung akzeptieren und warten bis Docker gestartet ist.

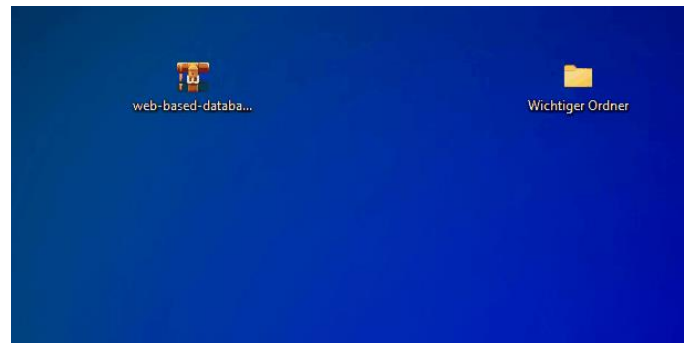
(In der Zwischenzeit kann mit den nächsten Schritten fortgefahren werden.)



4. [Projektrepository herunterladen](#) (Auf Link klicken + Strg gedrückt)
 - a. Auf Code klicken (Grüner Button, dann „Download ZIP“)

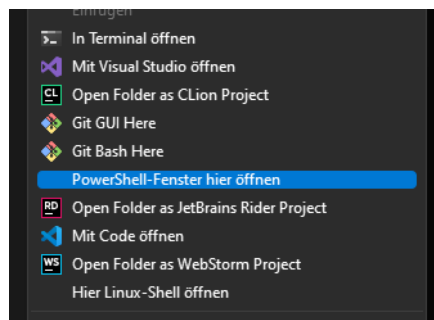


5. ZIP herunterladen und an beliebigen Ort entpacken

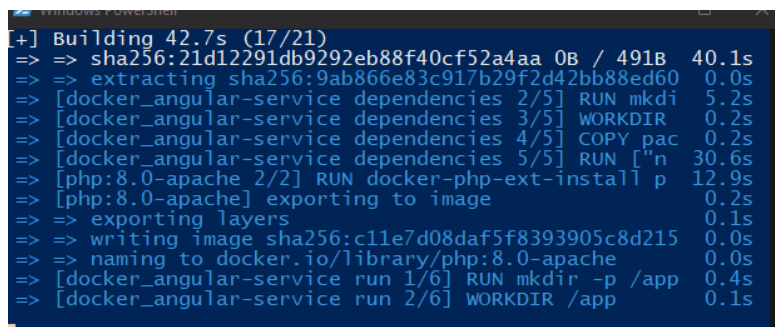


6. Im Entpackten Archiv den Ordner „docker“ öffnen

7. Im Ordner mit gedrückter Shifttaste und rechtem Mausklick die Powershell öffnen



8. Den Befehl: `docker compose up --build` in die Powershell eingeben und mit Enter bestätigen



9. Wenn Powershell „Compiled successfully“ anzeigt, dann ist die Installation abgeschlossen

10. Beliebigen Browser öffnen und localhost:4200/login in der Adressleiste eingeben

