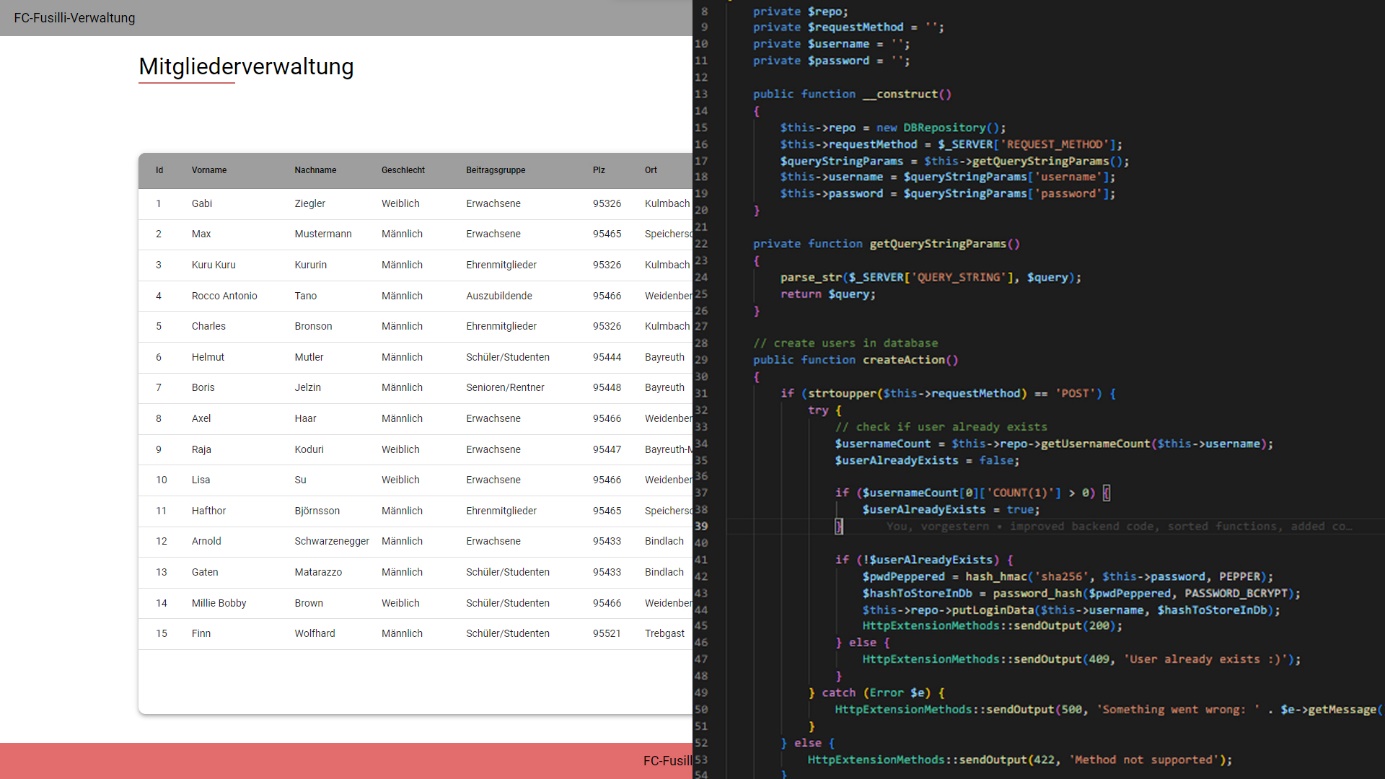
Juli - 2022

WEBBASIERTER-DATENBANKMANAGER



Gruppe 4 – Fusilli

|  |  |
| --- | --- |
| Thomas S. | @Noodle693 |
| Marco H. | @GodlesZ95 |
| Samuel S. | @Gunzli02 |
| Michael W. | @miwied |

Inhaltsverzeichnis

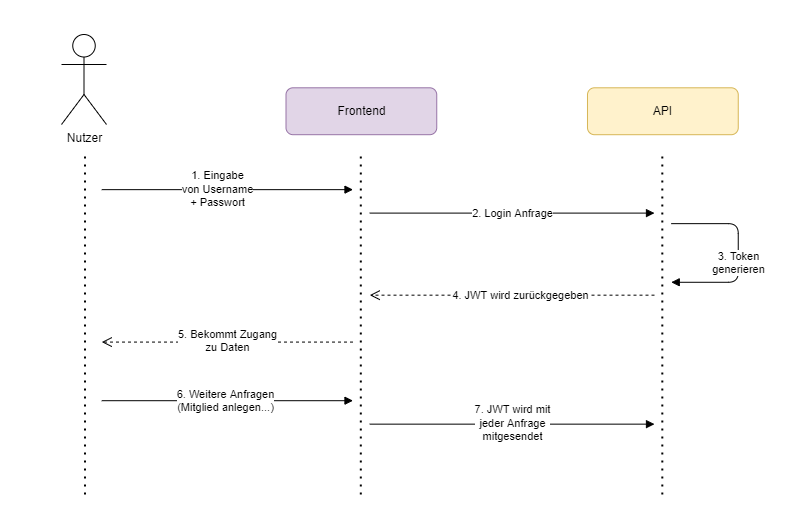
**JSON Web Token (JWT)**

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.

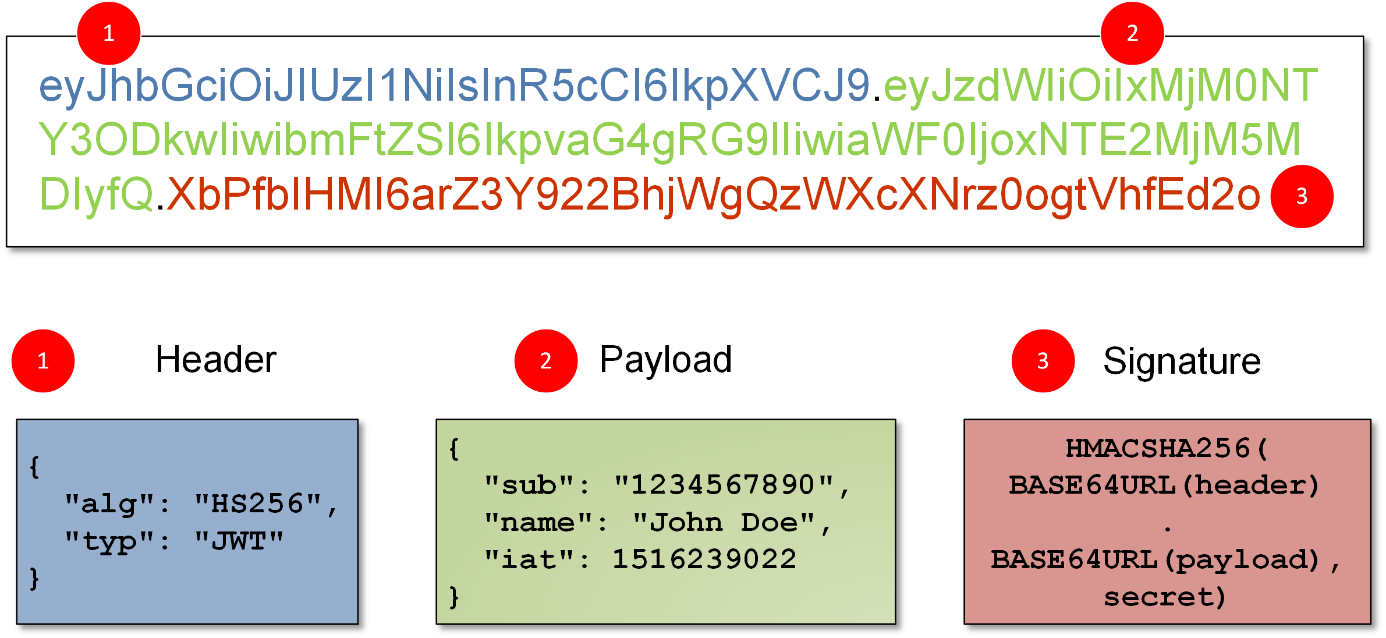
Wir gleichen die Gültigkeit dieses Tokens bei jeder Anfrage ab, damit ein unautorisierter Zugriff auf die personenbezogenen Daten unmöglich ist.

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**

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).

1. 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**

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungDie Generierung sieht wie folgt aus:

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungZunächst 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.

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.eyJpYXQiOjE2NTc3MTU5NTcsImlzcyI6InNwb3J0dmVyZWluIiwibmJmIjoxNjU3NzE1OTU3LCJleHAiOjE2NTc4MDIzNTcsInVzZXJOYW1lIjoiYWRtaW4ifQ.1jOKVfDLzEkr3DuhWxeRIl-23devOB6w9nmOwtjZOnAXKZ1r\_Bga2YIYdiqFYlXqwjxROB27BAG6Qt65J9FW5w”

**Abgleichung**

Text

Description automatically generatedDer 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.

Text

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

Text

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

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

Text

Description automatically generatedWir versuchen diesen mit dem JWT\_SECRET\_KEY und dem Algorithmus zu dekodieren.

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

Text

Description automatically generatedText

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

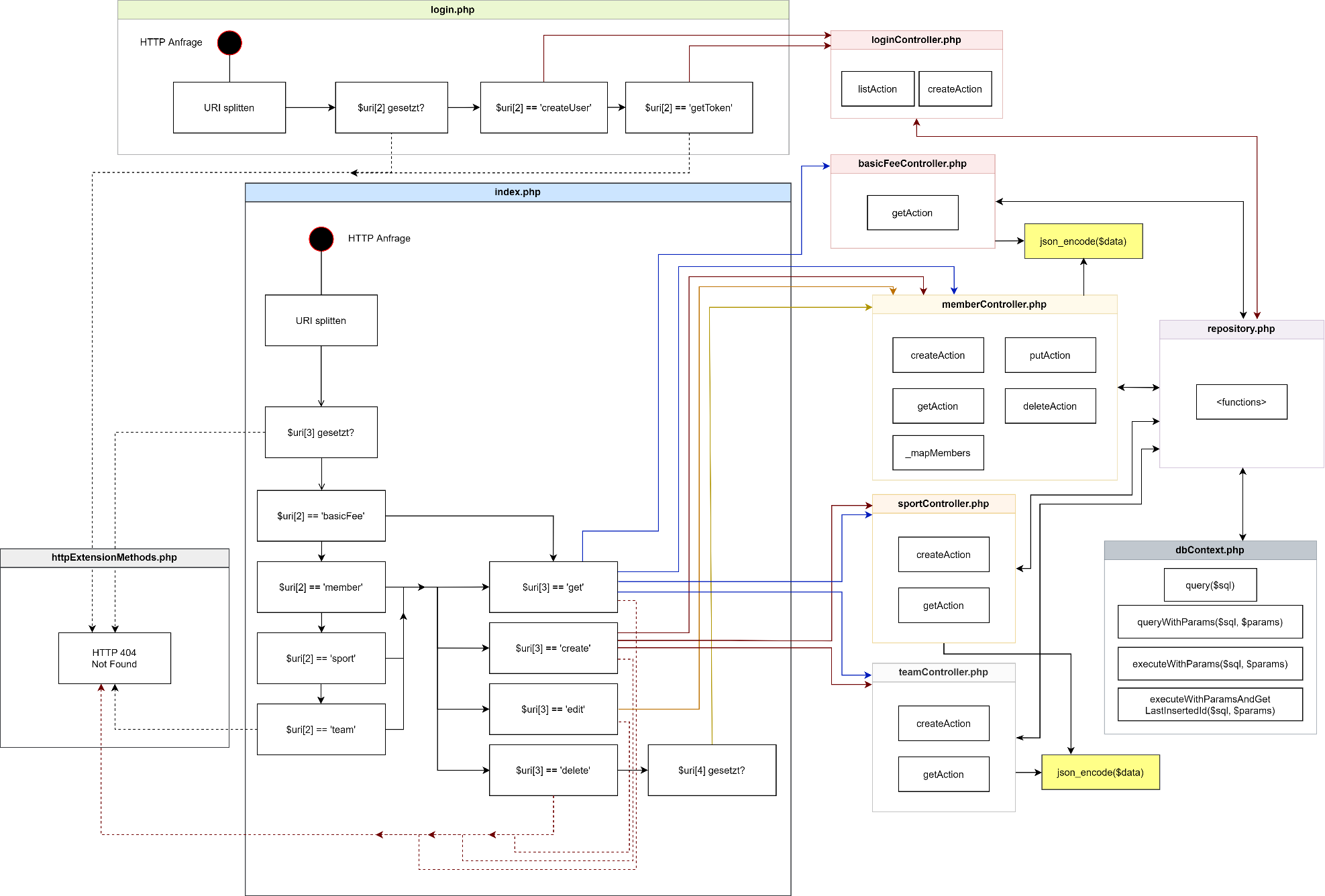
A screenshot of a computer

Description automatically generated with medium confidenceWenn 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.

Text

Description automatically generatedZuletzt wird (wieder in der accessControl.php) der Herausgeber mit unserer JWT\_DOMAIN\_NAME Variable abgeglichen.

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.

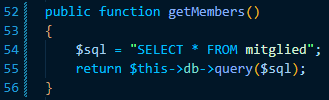
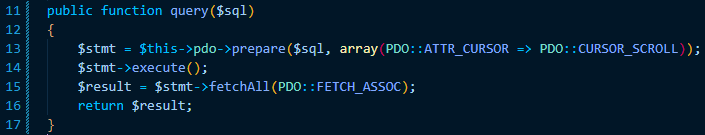
Der Controller überprüft die Anfrage und ruft das Repository mit seiner jeweiligen Funktion auf.

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.

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.

Text

Description automatically generatedHier noch ein kleiner Ausschnitt von dem Mapping was wir durchführen:

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:

A screenshot of a computer

Description automatically generated with medium confidence