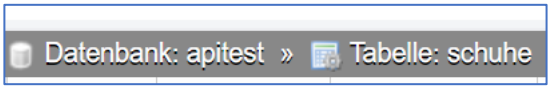
GET-Request mit Ionic 8 standalone und Angular 17
Datenbankzugriff mittels API
Liste aus der Datenbank befüllen inkl. Bild (URL) mit

1. Vorhandene Seite „shop“ abändern mit der List-Darstellung
    a. For-Schleife
    b. Interpolation {{ }}
2. In shop.page.ts mit der passenden Funktion „getSchuhe()“ ausformen, die auf das Service (shopapi) zugreift
3. shopapi mit der passenden GET-Abfrage befüllen
4. API einbinden – nur zur Info

Ziel:

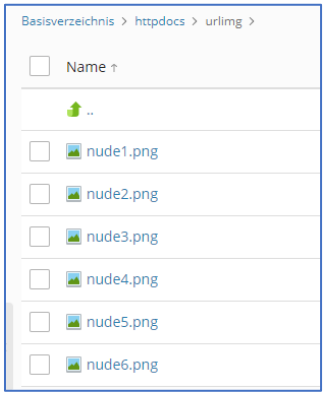
- Zugriff mittels Ionic auf Daten per GET-Request über eine API.
- Die API liegt „irgendwo“ außerhalb der App – z.B. auf einem Server (hosttech.at) und ist schon vorbereitet
- Verwendung eines „services“, welches die Verbindung zur API herstellt
- Die bestehende Seite „shop“ abändern, um Daten aus der Datenbank in der Seite „shop.page.html“ anzeigen lassen



Testen und ansehen auf:
https://ioniceinkauf.digbizmistelbach.info
Die Tabelle „schuhe“ ist bereits vorhanden

id	name	preis	rating	beschreibung	bildurl
1	Delta run	250	4.5	Keilsandalette, rainbow	https://ioniceinkauf.digbizmistelbach.info/urlimg/...
2	Delta	280	4.7	Plateausandalette, rainbow	https://ioniceinkauf.digbizmistelbach.info/urlimg/...
3	Ko Mid	200	4.2	Keilsandalette, malibu	https://ioniceinkauf.digbizmistelbach.info/urlimg/...
4	Rico	240	4.8	Plateausandalette, summer	https://ioniceinkauf.digbizmistelbach.info/urlimg/...
5	Strappy Hi	180	4.5	Hi Heal Sandalette	https://ioniceinkauf.digbizmistelbach.info/urlimg/...
6	Delta	280	4.6	Plateausandalette, bronze	https://ioniceinkauf.digbizmistelbach.info/urlimg/...

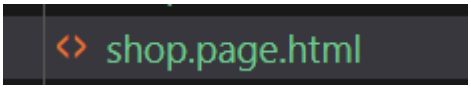
Die Bilder werden nicht als „blob“ gespeichert, sondern als URL zu einem Ordner, der auch auf dem Server liegt. Die komplette „bildurl“ ist oben im Screenshot leider nicht ganz sichtbar, lautet aber z.B.
URL: https://ioniceinkauf.digbizmistelbach.info/urlimg/nude1.png
Der Ordner auf dem Server lautet: „urlimg“



In DB:

#	Name	Typ	Kollation	Attri
1	id	int(11)		
2	name	text	utf8mb3_general_ci	
3	preis	int(11)		
4	rating	decimal(2,1)		
5	beschreibung	text	utf8mb3_general_ci	
6	bildurl	varchar(100)	utf8mb3_general_ci	

1.)Öffne im Ordner „pages“ die HTML-Seite „shop.page.html“



Im Content erfolgt die Darstellung einer Liste. Diese holt sich mit einer FOR-Schleife die Daten und zeigt diese in <ion-label> an.

@for:

Das ist eine Direktive, um durch die Liste der Manager zu iterieren (durchlaufen).

- for startet eine Schleife und wiederholt das Element und dessen Inhalte für jedes Element.
- dann eine sinnvolle Bezeichnung, hier bietet sich zwar „schuhe“ an, aber wir belassen, weil es schon da ist und sowieso häufig genutzt wird „item“.
- Nach dem „item of“ steht oft die Mehrzahl davon, hier „items“. Gerne wäre auch sinnvoll „alleSchuhe“.
- track i;
- „let“ bedeutet in TypeScript soviel wie „Variable“
- In den Label verwendet man die Anzeige mittels INTERPOLATION aus dem Bereich von Angular. Das ist nicht Ionic, sondern ein Code von Angular.
- „item“ in Einzahl, sie wie in der FOR Schleife darüber.
- Nach dem Punkt folgt das Element, wie es in der Datenbank steht angeschrieben, damit es ausgegeben werden kann

INFO: Interpolation

Kommt aus der Open-Source-Software ANGULAR, welche hier mit Ionic die Grundlage bildet.

Dabei werden eingesetzt:

- doppelte geschwungene Klammern vorne und hinten
- Name der Variable aus der For-Schleife
- Nach dem Punkt der Name des Elements aus der Datenbank – die in der .ts – Datei mithilfe des Services aus dem API geholt wird.

Es wird die „**Interpolation**“ („{{}}“) verwendet, um Propertys der Komponentenklasse im Template auszugeben. Die Komponenten-Klasse wird später in der passenden „ts“ in „@Component“ erstellt werden.

{{item.name}} passt nicht in das input, nur in das label

Info:
Das Element aus der Datenbank (nach dem Punkt) wird in der „shop.page.ts“ über die

- Funktion „getSchuhe()“ und einem
- „Observable“ (mit „subscribe“) über dem Service „shopapi“

aus der Datenbank geholt. Dazwischen steht hier, zur Absicherung, das API, welches per http-Request im Service angesteuert wird.

ÄNDERE:

- Lösche die Fragezeichen nach den „item“ jeweils
- Erstelle den Bildzugriff neu:
  <ion-img src={{item.bildurl}}></ion-img>

```
11      <ion-col sizeLg="3" sizeMd="4" sizeSm="6" sizeXL="3" sizeXs="6">
12      <!--Aufteilung bei 12 Spalten pro Reihe-->
13      <ion-card>
14      <ion-thumbnail>
15        <ion-img src={{item.bildurl}}></ion-img>
16        </ion-thumbnail>
17        <ion-label>
18          <ion-text color="dark"><strong>
19            {{item.name}}
20          </strong></ion-text>
21                    <p>
```

```
22                  <ion-text color="dark">
23                    <strong>
24                    € {{item.preis}}
25                    </strong>
26                  </ion-text>
27                  <ion-text class="rating" color="dark">
28                    {{item.rating}}
29                  </ion-text>
30                </p>
31          </ion-label>
32          </ion-card>
33      </ion-col>
34      }
35    </ion-row>
36  </ion-content>
```

Die HTML ist somit fast fertig.

2)shop.page.ts vorbereiten

In der export-Klasse dieser Seite muss man ein Array erstellen, welches jeden Wert annehmen kann (any) und den Namen hat, der als 2. Name in der FOR-Schleife aus der HTML-Datei vorkommt, hier, wie immer, die Mehrzahl „items“. Diese beiden müssen übereinstimmen.

Lösche „allItems“ und schreibe „items“ um:

```
10  export class ShopiPage implements OnInit {
11
12    items: any[] = [];
13    allItems: any[] = [];
14    private api = inject(ApiService);
```

Ergebnis: NEU

```
15  export class ShopPage implements OnInit {
16
17    items: any[] = [];
18
19    private api = inject(Api);
```

Lösche die dazupassende Funktion „getItems“ weiter unten, da diese nicht mehr benötigt wird.

```
25    getItems(){
26      this.allItems = this.api.items;
27      this.items = [... this.allItems];
28    }
```

Nun die Verbindung mit der API (shoeapi.st):

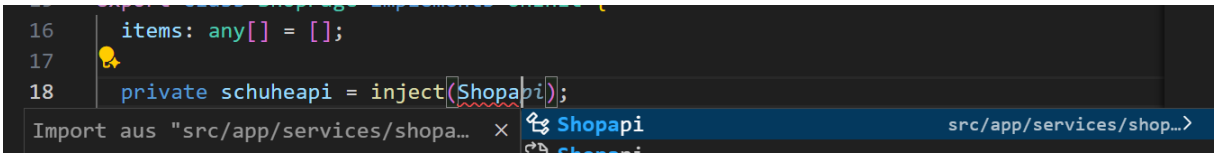
Wie bekannt lautet in der Service-Datei „shopapi.ts“ die „export class Shopapi“, also mit Großbuchstaben „S“ und dann Shopapi.

```
6   })
7   export class Shopapi {
8
9   private http = inject(HttpClient);
```

Diese genaue Bezeichnung muss man dann in der dazu passenden .ts, hier die „shop.page.ts“ übernehmen – hinten in der „inject“ – wenn man es wieder sinnvoll eingibt, kann man den Import automatisieren - ansonsten muss man das händisch importieren.

```
16  export class ShopPage implements OnInit {
17    items: any[] = [];
18
19    private schuheapi = inject(Shopapi);
20
21    constructor() {}
```

Die Bezeichnung nach dem „private“ muss sich aber unterscheiden, daher ist hier „schuheapi“ verwendet.



Die private Methode „schuheapi“ wird dann verwendet, wenn eine neue Funktion die Verbindung zur API herstellt. Sie wird daher gleich in der „getSchuhe()“-Funktion eingebaut werden.

```
export class ShopPage implements OnInit {
  items: any[] = [];

  private schuheapi = inject(Shopapi);

  constructor() {
    this.getSchuhe();
  }

  ngOnInit() { }

getSchuhe() {
}
}
```

Erstelle im „constructor() { }“ eine neue Funktion mit dem passenden Namen getSchuhe();

```
  private schuheapi = inject(Shopapi);

  constructor() {
    this.getSchuhe();
  }
  ngOnInit() { }
```

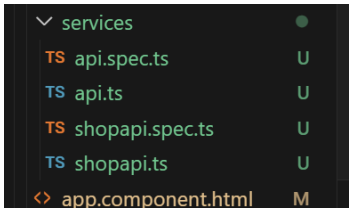
Danach wird noch vor der gelbe Klammer (dem Ende), eine **neue Funktion** erstellt.

- „getSchuhe()“
- Diese greift auf des Service zu und
- repliziert auf sich (this) und nutzt die Methode „schuheapi“,
- danach folgt die Funktion und sie soll sich ein wenig von der übergordneteten Funktion und daher „getSchuheApi()“ heißen.
- Mit „subscribe“ wird die Möglichkeit von „Observables“ genutzt.
- darin wird „items“ (in Mehrzahl, so wie oben in der For-Schleife der HTML)

```
21    constructor() {
22      this.getSchuhe();
23    }
24
25    ngOnInit() { }
26
27  getSchuhe() {
28    this.schuheapi.getSchuheApi().subscribe((response: any) => {
29      this.items = response;
30      console.log(this.items);
31    });
32  }
33  }
34
```

```
getSchuhe() {
 this.schuheapi.getSchuheApi().subscribe((response: any) => {
  this.items = response;
  console.log(this.items);
 });
```

Nebeneffekt: api.ts = nicht mehr nötig:
Dadurch hat das alte, vorher benötigte Service mit dem Namen „api.ts“ ausgedient und wird durch die „shopapi.ts" nun ersetzt, die ja nun auf eine Datenbank zugreift.



3)Service: shopapi.ts - Funktion erstellen

Erstelle ganz unten, noch vor der abschließenden Klammer eine neue Funktion, die den gleichen Namen hat, wie gerade in der shop.page.ts erstellt. Und zwar die, die in der getSchuhe() genannt wurde getSchuheApi().

```
 9    private http = inject(HttpClient);
10
11    getSchuheApi() {
12    }
13    }
14
```

- Diese Funktion erstellt eine GET-Abfrage in die API, die in diesem Fall schon auf dem Hosttech-Server liegt. Ansonsten kann man das natürlich auch „localhost“ betreiben.
- Diese Abfrage zielt auf die API. Im Detail auf die index.php mit dem Pfad „schuhe“. Dort wird in PHP die Datenbankabfrage erstellt und das Ergebnis als JSON-Datei übergeben.

Funktion für die Abfrage aus der Datenbank

Diese Funktion soll über den direkten Link zum API auf die Datenbank kommen.

- Dabei ist es EGAL wo die API liegt, weil man mit dem Http-Protokoll ja sowieso überall hinkommt.

Die Funktion mit einem passenden aussagekräftigen Namen:

- get – weil es eine get-Abfrage sein wird, im Gegensatz zu post oder update oder delete
- Schuhe – weil es um die Darstellung der Schuhe geht.
- Api wäre gut, damit man es von der bereits erstellten Funktion „getSchuhe“ in der „shop.page.ts“ besser unterscheiden kann, weil dort beide unmittelbar aufeinandertreffen werden.
- DAHER heißt die Funktion: getSchuheApi()

Die Methode „http.get()“ fragt die Daten per GET vom Server ab.

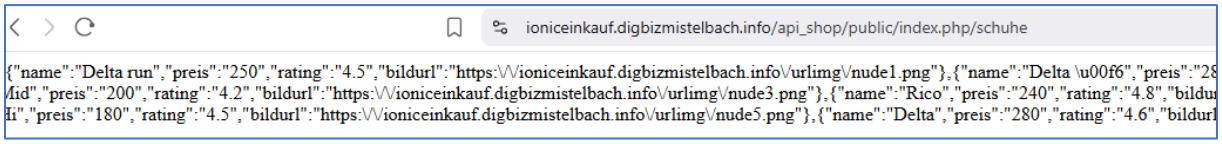
- Diese Methode „get“ soll ein Array mit verschiedenen Schuhen aus der Datenbank zurückgeben
- In der get-Methode wird die URL eingefügt.
- Die „get-Methode“ gibt eine Observable zurück. Dieses Observabel besteht aus einem Array mit allen Schuhen aus der Datenbank.

```
10
11   getSchuheApi() {
12     return this.http.get('https://');
13   }
14   }
15
```

Die URL wird folgende sein:

https://ioniceinkauf.digbizmistelbach.info/api_shop/public/index.php/schuhe

Teste vorher die URL im Browser:



Die Daten am Server werden durch eine schon vorhandene API vom Typ SLIM bereitgestellt, an die in der obigen URL die Anfrage (=GET) gestellt wird. Wie diese gebaut und hergerichtet wird werden wir später separat behandeln.

Hier ist nur wichtig, dass sie funktioniert.

NUR zur INFO

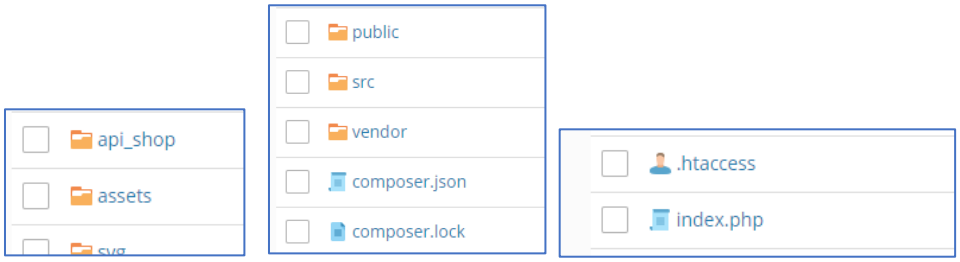
4)API einbinden

ist eine eigene Software außerhalb von IONIC – die Verbindung ist ja bereits in der oben bearbeiteten „shopapi.ts“ schon vorhanden

```
14  getSchuheApi(){
15  return this.http.get('https://ioniceinkauf.digbizmistelbach.info/api_shop/public/index.php/schuhe');
```

Diese Software ist bei uns eine SLIM-API.

Sie liegt in einem Online-Ordner auf hosttech (339.hostserv.eu)



Die Verbindung zur Datenbank liegt in der Datei „src/config/db.php“

db.php

```php
<?php
/**
 * Connect MySQL with PDO class
 */
class db {

  private $dbhost = 'localhost:3306';
  private $dbuser = 'xxxxxxxxxx';
  private $dbpass = 'xxxxxxxxxxxxxxxxx';
  private $dbname = 'xxxxxxxxxxx';

  public function connect() {
```

Die Schnittstelle der API liegt in der Datei: public/index.php

Der Endpunkt ist „schuhe“ und ist hier eine neue „get-Abfrage“.
SELECT-Abfrage:

API:

```php
$app->get('/schuhe', function( Request $request, Response $response){

    $sql = "SELECT name,preis,rating,bildurl FROM schuhe";

    try {
      // Get DB Object
      $db = new db();

      // connect to DB
      $db = $db->connect();

      // query
      $stmt = $db->query( $sql );

      $schuhe = $stmt->fetchAll( PDO::FETCH_OBJ );
      $db = null; // clear db object

      // print out the result as json format
      echo json_encode( $schuhe );


    } catch( PDOException $e ) {

      // show error message as Json format
      echo '{"error": {"msg": ' . $e->getMessage() . '}';
    }

  });
```

```
$app->get('/schuhe', function( Request $request, Response $response){

   $sql = "SELECT name,preis,rating,bildurl FROM schuhe";

   try {
    // Get DB Object
    $db = new db();

    // connect to DB
    $db = $db->connect();

    // query
    $stmt = $db->query( $sql );

    $schuhe = $stmt->fetchAll( PDO::FETCH_OBJ );
    $db = null; // clear db object

    // print out the result as json format
    echo json_encode( $schuhe );
```

```
  } catch( PDOException $e ) {

    // show error message as Json format
    echo '{"error": {"msg": ' . $e->getMessage() . '}';
  }

});
```

Info: Bild anzeigen – nämlich als URL

https://www.reddit.com/r/dotnet/comments/16bhmpo/what_is_best_way_to_fetch_images_from_database/?tl=de&rdt=42599

https://www.youtube.com/watch?v=VO828_BkalA