Study

Uitleg over theorie zodat ge de werking beter begrijpt.

HTTP

Basic
HTTP is het protocol om te communiceren tussen een web-client (uw app) en een webserver (backend + databank).

Wanneer ge bijvoorbeeld http://google.com ingeeft in uw browser wordt er een HTTP-request gestuurd.

Een HTTP-request bestaat uit een URL, een body en headers.

De URL is de locatie waarnaar de request gestuurd wordt, de body bevat de data die meegestuurd wordt, en de headers bevatten meta-data zoals bijvoorbeeld session-ids, cookies, datum,..

Soorten requests
Er zijn verschillende soorten requests. De meest gebruikte zijn GET en POST.

GET wordt gebruikt om data op te vragen.
`HTTP GET https://google.be`
Vraagt bijvoorbeeld de HTML van google.be op om die dan in uw browser weer te geven.

POST wordt gebruikt om data te versturen.
Als ge een website met een formulier hebt wat ge invult wordt er zoiets gestuurd:
`HTTP POST https://mysite.com/submitform.php`

```
input_veld1=abc&input_veld2=xyz
```

https://developer.mozilla.org/en-US/docs/Learn/Forms/Sending_and_retrieving_form_data

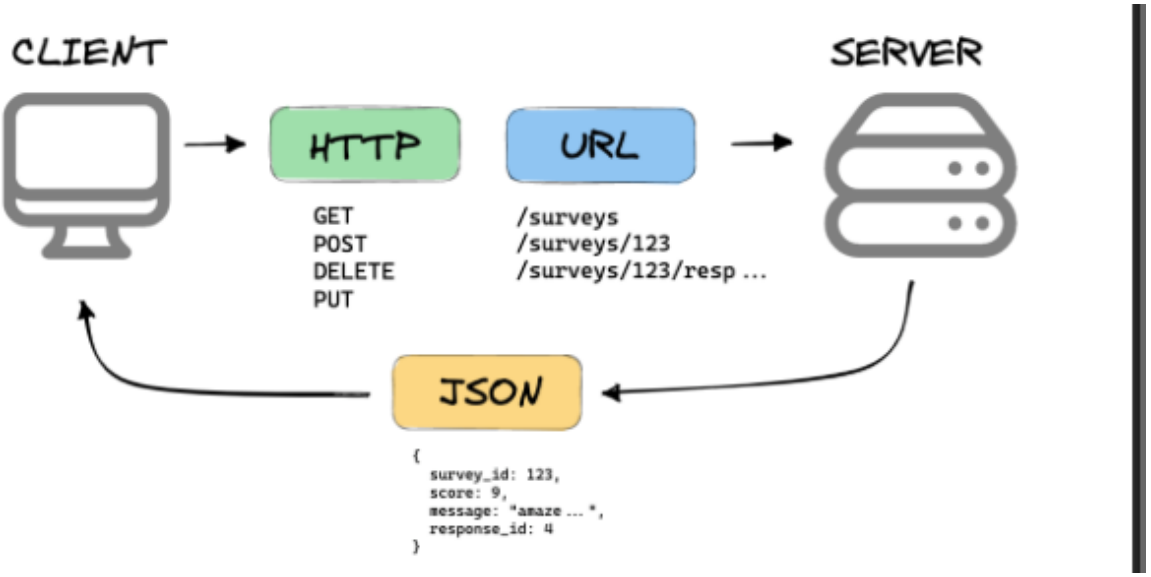
Responses
Wanneer een client een request stuurt naar de webserver stuurt de server een respons terug. Een respons bestaat uit een statuscode, de body en headers.

In het GET-voorbeeld van Google is de body de HTML van de google-pagina. De statuscode geeft aan of de request gelukt is.

https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

HTTPS
Zorgt det ge altijd https:// gebruikt en nooit http:// (behalve voor localhost)

REST API



API
Een API is een systeem om twee systemen met elkaar te kunnen laten communiceren. De data van uw app staat bijvoorbeeld op uw webserver. Via een API kan uw app die data dan opvragen.

Andersom ook, de gebruiker geeft info in op de app en deze moet verzonden worden naar de webserver. Dit wordt gedaan via de API.

REST
REST slaat puur op het principe dat we objecten (gebruikers, clubs, events, tickets,...) representeren in de vorm van een URL.

http://clubcentral.local/api/users.php/**61**

Is bijvoorbeeld het object User met ID 61.

```
{
  "ID": 61,
  "name": "Foo Bar",
  "telnr": "+3247583838",
  "email": "foobar@mail.com",
  "password": "9b8769a4a742959a2d0
}
```

Acties
Via de REST-API kunnen we objecten opvragen, toevoegen, wijzigen of verwijderen van onze databank. Dit doet ge door gebruik te maken van de verschillende soorten HTTP-requests.

- **GET:** Opvragen van object.
- **POST:** Aanmaken van object.
- **PUT:** Wijzigen van object.
- **DELETE:** Verwijderen van object.

In Dart kunt ge wisselen tussen deze HTTP-methodes door de volgende functies.

```dart
// Opvragen van data
final response = await http.get(Uri.parse(url));
```

```dart
// Toevoegen van data
final response = await http.post(
    Uri.parse(url),
    headers: <String, String>{
        'Content-Type': 'application/json; charse
    },
    body: json,
);
```

```dart
  // Wijzigen van data
  final response = await http.put(
      Uri.parse(url),
      headers: <String, String>{
          'Content-Type': 'application/json; charset=UTF-8',
      },
      body: user,
  );
```

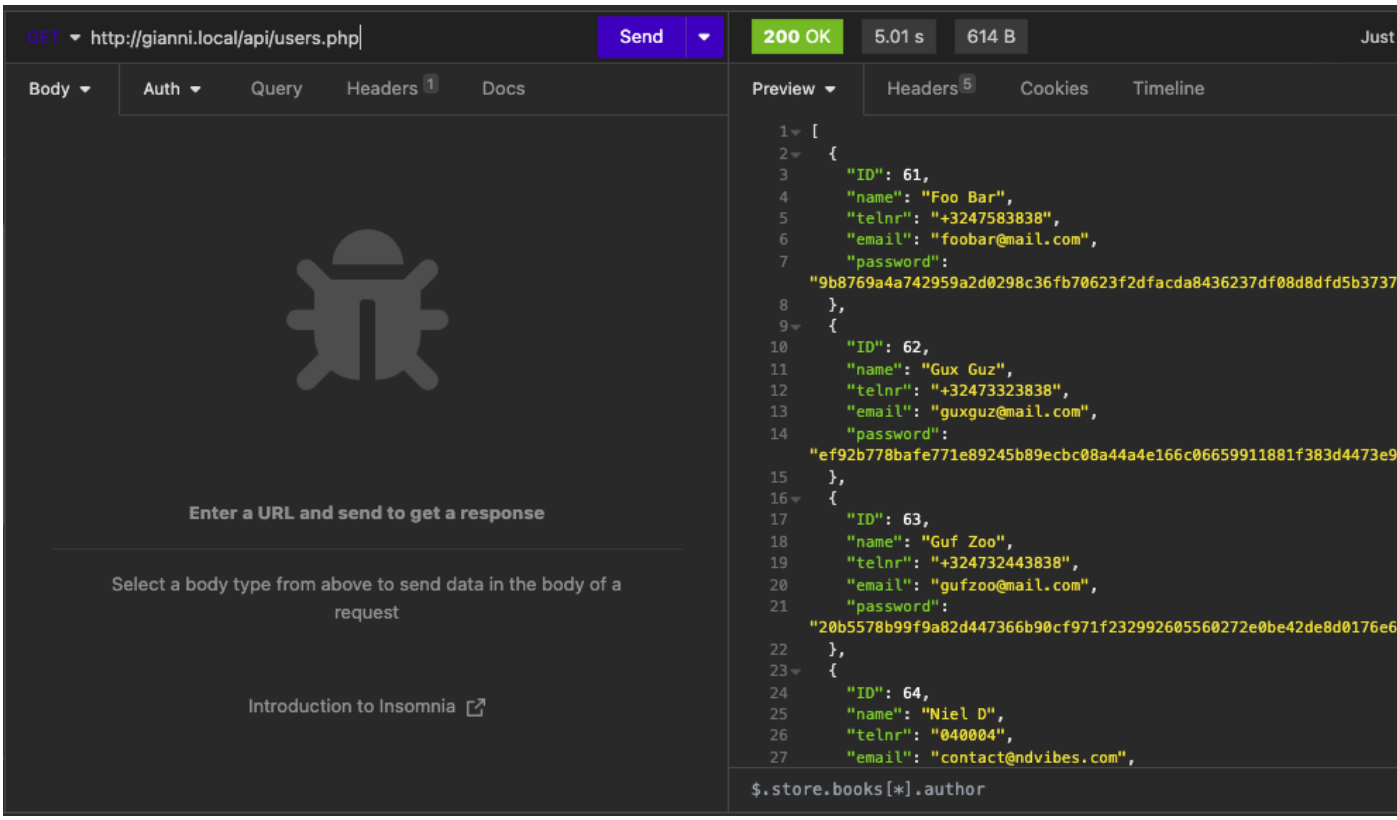
```dart
// Verwijderen van data
final response = await http.delete(Uri.parse(url));
```

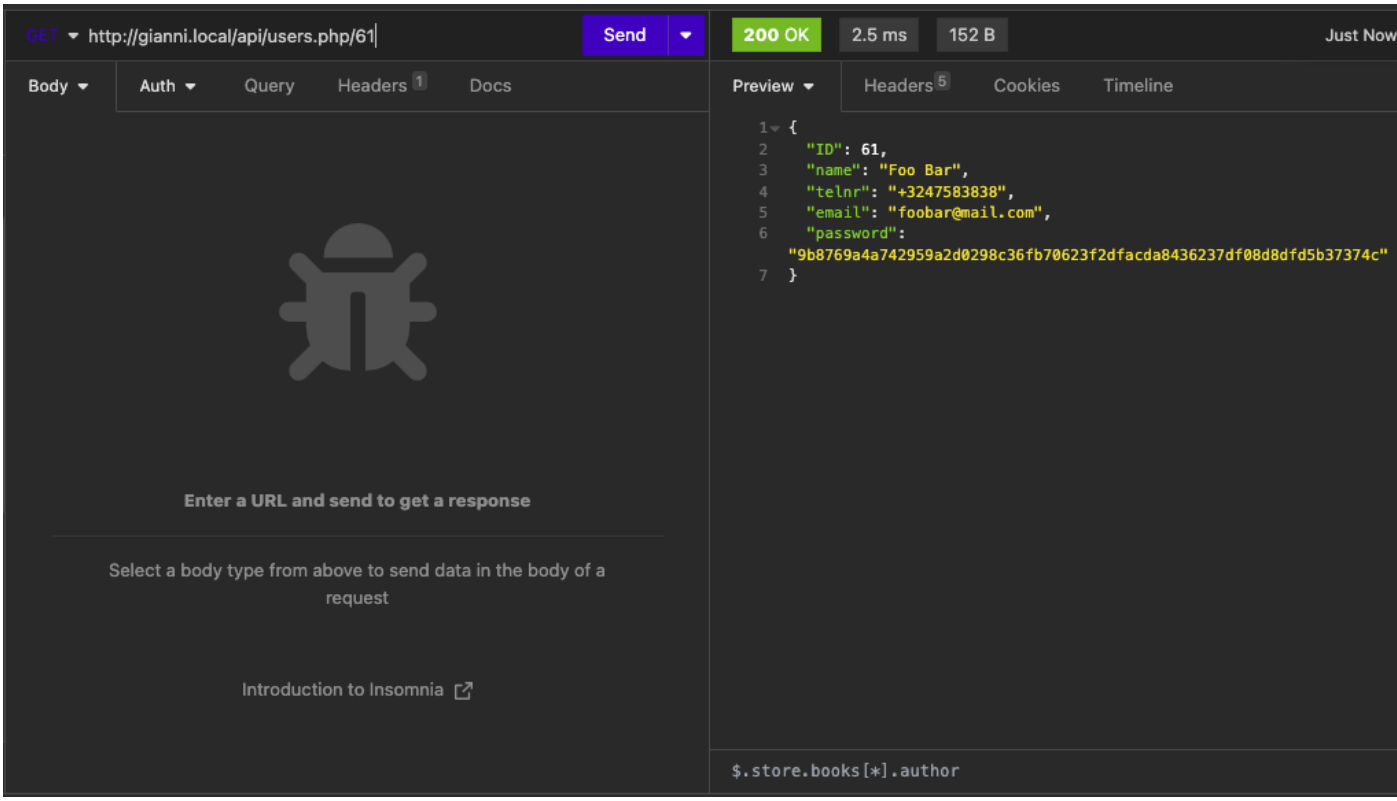
Endpoints

Een REST API heeft verschillende endpoints. Dit zijn de URL's die beschikbaar zijn. Elke endpoint heeft betrekking op een specifiek object.

Bijvoorbeeld http://clubcentral.local/api/users.php gaat over de gebruikers en http://clubcentral.local/api/clubs.php over de clubs.

Een endpoint kan ook parameters hebben. Een GET naar http://clubcentral.local/api/users.php vraagt ALLE gebruikers op. Als we enkel een specifieke gebruiker willen hebben voegen we de ID van de gewenste gebruiker toe als parameter. http://clubcentral.local/api/users.php/61





Het is ook mogelijk een object op te vragen niet bij het ID maar ook d.m.v een andere kolom. Wanneer een gebruiker bijvoorbeeld inlogt geeft deze zijn e-mail (of telnr) en paswoord op. In dit moment weten we dus de ID van de gebruiker niet. We kunnen de gebruiker opvragen door zijn e-mail adres via de volgende endpoint:

http://clubcentral.local/api/users.php/email/voorbeeld@mail.com

Body (data versturen)
Wanneer we een object willen aanmaken (via POST) of willen wijzigen (via GET) moeten we een body meesturen met onze request naar de endpoint.

Deze body is altijd in JSON.

```
// User data
Map<String, dynamic> data = {
    'name': 'Foo Bar',
    'telnr': '+3247583838',
    'email': 'foobar@mail.com',
    'password': sha256.convert(utf8.encode("pass123")).toString()
};
```

```
// Convert to JSON
String json = jsonEncode(data);

// Making request
// Adding a user HTTP POST
final response = await http.post(
    Uri.parse(url),
    headers: <String, String>{
        'Content-Type': 'application/json; charset=UTF-8',
    },
    body: json,
);
```

Headers
De enige Header in de requests die gij (normaal gezien) gaat moeten gebruiken is Content-Type om aan te geven aan de API dat uw body bestaat uit JSON.

```
headers: <String, String>{
    'Content-Type': 'application/json; charset=UTF-8',
},
```

Response
De API geeft een response terug. Deze is altijd in JSON. Dit kan een enkel JSON-object zijn wanneer ge een specifiek object opvraagt door bijvoorbeeld het ID mee te geven in uw endpoint. Of een lijst van objecten wanneer ge geen ID meegeeft en dus alle objecten opvraagt.

Gebruik `json.decode` om de JSON-String van uw respons om te zetten in een bruikbaar JSON-object.

```
Future<Map<String, dynamic>?> get(int id) async {
    String url = "${globals.URL}/api/clubs.php/$id?api_key=${globals.API_KEY}";

    final response = await http.get(Uri.parse(url));

    if (response.statusCode == 200) {
        return json.decode(response.body);
    } else {
        return null;
    }
}
```

Dan kunt ge op deze manier de properties van uw object (zoals naam, email,..) uitlezen:

```
var user_name = user['name'];
print("Username: $user_name");
```

Response code
Uw response bevat ook een response code.

https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

De voornaamste in deze API zijn:
200 OK: Request was ok.
201 Created: Object goed aangemaakt.
401 Unauthorized: Geen toegang tot API.
403 Not Found: Endpoint of object niet gevonden.

Response voorbeeld
Dit is de Dart-code om de response van de functie add_user te checken:

```
if (response.statusCode == 201) {
    print("[Success] User ID: ${response.body}");
    return int.parse(response.body);
} else {
    print("[Failed] status code ${response.statusCode}: ${response.body}");
    exit(0);
}
```

Als de gebruiker goed is aangemaakt krijgen we 201 terug. Na het aanmaken van een object geeft de API in de response body het ID van het aangemaakte object terug.

Als de response code niet 201 is, is er iets fout gelopen en stoppen we de applicatie.

Doordat de API het ID van een aangemaakt object teruggeeft wanneer het succesvol is aangemaakt. Kunnen we bijvoorbeeld bij het aanmaken van een gebruiker (lees: registeren) het ID opnemen in de sessie en de gebruiker automatisch na een registratie laten inloggen:

```
int user_id = await users.add_user(data);
```

Dart

In Dart stoppen we al deze requests in hun eigen functies, zodat die requests niet telkens manueel aangemaakt moeten worden. Bijvoorbeeld:

Aanmaken van gebruiker.

```
Future<int?> add(Map<String, dynamic> data) async {
    String url = "${globals.URL}/api/users.php?api_key=${globals.API_KEY}";

    String json = jsonEncode(data);

    final response = await http.post(
        Uri.parse(url),
        headers: <String, String>{
            'Content-Type': 'application/json; charset=UTF-8',
        },
        body: json,
    );

    if (response.statusCode == 201) {
        return int.parse(response.body);
    } else {
        return null;
    }
}
```

Deze functie kunnen we dan aanroepen:

```
    // Make user JSON
    Map<String, dynamic> user = {
        'name': name,
        'telnr': telnr,
        'email': email,
        'password': pass
    };

    int? user_id = await users.add(user);
```

Opvragen van gebruiker via email.

```
Future<Map<String, dynamic>?> get_by_email(String email) async {
    String url = "${globals.URL}/api/users.php/email/$email?api_key=${globals.API_

    final response = await http.get(Uri.parse(url));
    if (response.statusCode == 200) {
        return json.decode(response.body)[0];
    } else {
        return null;
    }
}
```

Aanroep:

```
Map<String, dynamic>? user = await users.get_by_email(email);
```

Uit de variable user lezen we vervolgens al de gegevens van die user, bijvoorbeeld zo:

```
user['password']
```

Wat je tussen de haakjes zet zijn de properties van dat object, oftewel de kolommen van de overeenkomende tabel.

Async

De functies in Dart moeten asynchroon zijn. Dit omdat we HTTP-requests gebruiken. Synchrone apps (zoals veel basic apps) worden lijn-per-lijn uitgevoerd. Bijvoorbeeld

```
print("1");
some_function();
print("2");
```

In deze synchrone code wordt some_function pas uitgevoerd wanneer “1” is uitgeprint. En “2” wordt pas uitgeprint wanneer some_function is uitgevoerd.

In de meeste toepassingen is dit ok. Het probleem net HTTP-requests is dat ge niet weet hoelang dat request duurt. Als ge een trage webserver hebt die pas na 5 seconden reageert zou dat betekenen dat uw app 5 seconden vasthangt.

Een asynchrone functie wordt op een aparte tijdlijn uitgevoerd. Bijvoorbeeld:

```
print("1");
some_async_function();
print("2");
```

Er zal eerst “1” uitgeprint worden. some_async_function en het uitprinten van “2” worden tegelijkertijd uitgevoerd. Afhankelijk van de snelheid van some_async_function wordt eerst diens output of eerst “2” getoond. Dat weet ge op voorhand niet.

Willen we toch zeker zijn dat “2” pas geprint wordt nadat some_async_function klaar is gebruiker we het await keyword. Het sleutelwoord await kan enkel gebruikt worden in functies die zelf ook async zijn.

```
print("1");
await some_async_function();
print("2");
```

Nu wordt “2” 100% zeker pas uitgevoerd wanneer some_async_function gereed is.

Een asynchrone functie definieren we zo:

```
Future<String> get_user(int id) async {
```

Let op het gebruikt van Future en async.

get_user aan roepen returnt het type Future. Wilt ge de String verkrijgen die deze functie returnt doet ge dat door het sleutelwoord await voor uw functie aanroep te zetten.

```
var user = json.decode(await users.get_user(user_id));
```