# 2 Full-Stack Vanilla

## 2.1 Einleitung

In dieser Übung bauen wir eine sehr kleine, aber vollständige Full-Stack Web-Anwendung. Findet euch dazu bitte in zwei Gruppen zusammen, sodass in jeder Gruppe mindestens ein:e Teilnehmer:in ist, der:die am Frontend-Modul teilgenommen hat.

Wir arbeiten im Backend mit PHP und SQLite und im Frontend mit HTML, CSS und JavaScript. Dabei verwenden wir keine Frameworks - Vanilla eben.

Da wir uns folglich um *alles* selbst kümmern müssen, werden wir zunächst auf alles verzichten, das nicht unbedingt nötig ist, um die gewünschte Funktionalität zu erreichen. Wir gehen dazu davon aus, dass Benutzer keine Fehler machen und alle Anfragen immer korrekt sind. Eine steile These… in der Praxis würden wir natürlich jede Eingabe validieren und eine ordentliche Fehlerbehandlung implementieren.

## 2.2 Anforderungen

Das Backend stellt eine REST-API mit folgenden vier Endpunkten bereit:

```
GET /api/menu.php
GET /api/orders.php
GET /api/orders.php?id={order_id}
POST /api/orders.php
```

Das Frontend ist eine Single-Page-Application (SPA), die ein Menü anzeigt und Bestellungen ermöglicht. Dabei werden die Inhalte für das Menü dynamisch von der REST-API abgerufen. Durch Klicken auf einzelne Einträge im Menü kann die Benutzerin eine Bestellung zusammenstellen und schließlich absenden.

Verwendet folgendes Datenmodell für eure Datenbank:

| MenuItems |
|---|
| id |

| MenuItems |
| --- |
| name |
| description |
| price |

| Orders |
| --- |
| id |
| created |

| OrderItems |
| --- |
| id |
| order_id |
| menu_item_id |
| quantity |

## 2.3 Arbeitsschritte

### 2.3.1 Vorbereitung

Ich schlage folgende Projekt-Struktur vor:

```
/api
    menu.php
    orders.php
db.sqlite
index.css
index.html
index.js
```

Richtet das Projekt so für euch ein, dass ihr es über euren lokalen Web-Server z.B. unter `http://localhost:8000` erreichen könnt.

### 2.3.2 Menü anzeigen

Implementiert zuerst den Endpunkt `GET /api/menu.php`:

```
GET /api/menu.php

[
    { "id": 1, "name": "Vanille", "description": "Vanille-Eis", "price": 1.5 },
    { "id": 2, "name": "Schokolade", "description": "Schokoladen-Eis", "price": 1.5 },
    { "id": 3, "name": "Erdbeer", "description": "Erdbeer-Eis", "price": 1.5 }
]
```

Legt dazu die entsprechende Datenbank-Tabelle an und befüllt sie mit Beispiel-Daten.

Verbindet euch in der `menu.php` mit der Datenbank und stellt eine Anfrage an die eben erstellte Tabelle.

Dazu könnt ihr folgenden PHP-Code verwenden:

```php
$db = new SQLite3('./db.sqlite', SQLITE3_OPEN_CREATE | SQLITE3_OPEN_READWRITE);

$menu = [];

$stmt = $db->prepare('...'); // TODO: SQL query
$result = $stmt->execute();

while ($row = $result->fetchArray(SQLITE3_ASSOC)) {
    array_push($menu, $row);
}
```

Implementiert dann das Frontend, um das Menü anzuzeigen.

### 2.3.3 Bestellungen aufgeben

Implementiert dann den Endpunkt `POST /api/orders.php`:

```
POST /api/orders.php
{
    "items": [
        { "menu_item_id": 1, "quantity": 1 },
        { "menu_item_id": 2, "quantity": 2 }
    ]
}
```

Legt dazu die entsprechenden Datenbank-Tabellen an und befüllt sie mit Beispiel-Daten.

In der `orders.php` müsst ihr zunächst prüfen, ob die aktuelle Anfrage ein `POST` ist.

Falls ja, könnt ihr folgenden PHP-Code verwenden, um an die übertragenen Daten zu gelangen:

```
function get_post_data() {
    return json_decode(file_get_contents('php://input'));
}
```

Verbindet euch mit der Datenbank und fügt die Daten in die entsprechenden Tabellen ein. Nachdem ihr Daten eingefügt habt, könnt ihr die neu vergebene ID mit folgender Funktion abfragen (ihr braucht sie für die folgenden Einfüge-Operationen):

```
$order_id = $db->lastInsertRowID();
```

Implementiert dann im Frotend die Funktion, eine Bestellung zu erstellen und abzusenden.

### 2.3.4 Bestellungen einsehen (optional)

Implementiert die Endpunkte `GET /api/orders.php` und `GET /api/orders.php?id={order_id}`.