Einführung

Dieses Arbeitsblatt basiert auf bereits Gelerntem. Möge die Erinnerung mit dir sein! 🤞

Wir konzentrieren uns hier auf den Serverteil, denn Axios Calls und GUI Frontend haben wir schon in Webtechnologien gemacht. Daher verwenden wir den REST Client in VS Code zum Testen der Server Routes.

Das Grundgerüst für den Server, **Server Basis Pimped.zip**, ist unser Ausgangspunkt. Es enthält bereits in **package.json** einen Verweis, um **ESLint** lokal zu installieren. Außerdem wurde auch ein **.eslintrc.js** File bereits mit den wichtigsten Rules erstellt.

Bei jedem größeren Softwareprojekt gibt es einen bzw. mehrere verbindlichen Styleguides. Vielleicht kennst du den GUI Styleguide von Google https://material.io/design, der grob die Richtung für GUI Design vorgibt und dann oft noch auf das jeweilige Projekt bzw. Unternehmen angepasst wird. Wir konzentrieren uns aber hier auf das Programmieren.

Ein Programmier- Styleguide ist ein Leitfaden für Programmierkonventionen, Stil und Best Practices für ein Team oder Projekt. Warum? Ein Team, das einem Styleguide folgt, hilft jedem, Code auf konsistente Weise zu schreiben, und konsistenter Code ist einfacher zu lesen, zu verstehen und besser zu warten, sodass neue Funktionen schneller hinzugefügt werden können. Wenn du ein Software-Diplomarbeitsprojekt hast, empfehle ich dir auch einen Styleguide festzulegen.

Sehr populär ist der Airbnb Styleguide. Zur Erklärung lies: https://github.com/airbnb/javascript

Die Codebeispiele sind bereits auf den Airbnb Styleguide angepasst. Ich habe ein paar kleinere Anpassungen vorgenommen. Hier ist der **.eslintrc.js** File:

```
module.exports = {
  env: {
    node: true,
    commonjs: true,
    es2021: true,
  },
  extends: ['airbnb-base'],
  parserOptions: {
    ecmaVersion: 12,
  },
  rules: {
    'linebreak-style': 0,
    'object-curly-newline': 0,
    'no-console': 0,
  },
};
```

Wenn du Prettier richtig konfigurierst, hast du schon fast alles automatisch richtig gestylt (**settings.json**):

```
"prettier.singleQuote": true,
"prettier.semi": true,
"prettier.bracketSpacing": true,
"prettier.trailingComma": "all",
"prettier.printWidth": 110,
"prettier.endOfLine": "auto",
"prettier.arrowParens": "always",
```

1. Installiere in PostgreSQL die Datenbank **cocktails.sql**.

2. Lege einen User **app** in der Datenbank an und vergib passende Rechte (siehe Aufgaben später). Die Rechte sollen möglichst eingeschränkt sein und nur das was gefordert ist erlauben!

3. Füge die nötigen Variablen in .**env** hinzu.

PGHOST=
PGUSER=app
PGDATABASE=
PGPASSWORD=1234
PGPORT=5432

Achtung alle folgenden Routen sollen REST Routen sein. D. h. den Architekturprinzipien von REST folgen. Wenn du unsicher bist, frage deine Lehrer!

4. Implementiere die Route, welche Namen und Preise aller Cocktails zurückliefert.

Beispiel:

```
[
  {
    "cname": "Alice",
    "preis": "7.50"
  },
  {
    "cname": "Coconut Kiss",
    "preis": "7.20"
  },
  {
    "cname": "Florida Cocktail",
    "preis": "7.20"
  },
  {
    "cname": "Long Distance Runner",
    "preis": "7.80"
  },
   . . .
]
```

5. Implementiere die Route, welche für einen angegebenen Cocktail die Zutaten zurückliefert. Der Name des Cocktails ist nicht fix, die Route muss daher variabel sein!

Beispiel:

```
GET http://127.0.0.1:3000/cocktails/Alice/zutaten
```

```
[
 "Obers",
 "Grenadine",
 "Orangensaft",
 "Ananassaft"
]
```

6. Implementiere die Route, welche alle Cocktails, deren Preis kleiner als der angebene Wert ist, zurückliefert.

Beispiel für Preis 7.2

```
[
  {
    "cname": "Coconut Kiss",
    "preis": "7.20"
  },
  {
    "cname": "Florida Cocktail",
    "preis": "7.20"
  },
  {
    "cname": "Pina Colada Natural",
    "preis": "7.20"
  },
  {
    "cname": "Adonis",
    "preis": "6.20"
  },
  {
    "cname": "El Presidente",
    "preis": "6.80"
  },
   . . .
]
```


7. Implementiere die Route, welche einen bestimmten Cocktail, der durch seinen Namen angegeben wird, löscht. Gibt es den Cocktail nicht, gib eine Fehlermeldung aus:

```
[nodemon] 2.0.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node app.js`
Error ====>  Cocktail Cinderella not found!
DELETE /cocktails/Cinderella 500 96.792 ms - 34
```

Falls erfolgreich gib zurück:


```
"Deleted"
```


8. Implementiere die Route, um einen Cocktail einzufügen. Die id, **cid** ist vom Typ SERIAL. Gib die eingefügte id mit RETURNING zurück.


```
"cname": "Hit and Run",
"preis": 12.22,
"zubereitung": "eo ipso",
"kid": 1,
"zgid": 1,
"sgid": 1
```



```
"Inserted 54"
```


9. Implementiere die Route um den Preis eines Cocktails, der durch seinen Namen angegeben wird, zu ändern. Gib den neuen Preis mit RETURNING zurück.


```
"Updated to 99.99"
```