

Anleitung

Backend / API - Praktikum WebAnwendungen 2

Robert Kuti
21.02.2023

Zusammenfassung

Titel Bedienungsanleitung zum Backend / API – Praktikum WebAnwendungen 2
Autor Robert Kuti (kuti@hs-albsig.de)
Datum 21.02.2023

Bitte pflegen:

Geändert von...	...am...	Kapitel
Robert Kuti	15.10.2019	Dokument erstellt
Robert Kuti	18.10.2019	Dokument erweitert
Robert Kuti	07.03.2021	Dokument erweitert
Robert Kuti	21.12.2021	Dokument erweitert
Robert Kuti	07.04.2022	Dokument erweitert
Robert Kuti	21.02.2023	Dokument erweitert

Inhalt

Zusammenfassung.....	2
Inhalt.....	3
1 Einführung	7
1.1 Node.js	7
1.1.1 Node.js Module	7
1.2 SQLite.....	8
1.2.1 Datenbank GUI / DBeaver / JAR Treiber	8
1.2.2 Datentypen.....	8
1.3 Verzeichnisstruktur des Backend	9
1.4 Verwendeter Editor / IDE.....	9
1.5 Sprachen und Standards	9
2 Datenbankstruktur	10
2.1 Dateiuploads und Datenbank	11
3 Installation und Inbetriebnahme	12
3.1 Vorbedingung	12
3.2 Installation und Betrieb des Backends	12
3.2.1 Hinweise zur Arbeit am Backend (Developer – Mode).....	13
3.2.2 Hinweis zur normalen Ausführung des Servers (Production – Mode).....	13
3.3 Aufruf des Backends / der API mit dem Browser	13
3.4 Server beenden.....	14
3.5 Änderungen am Backend / Code.....	14
3.6 Änderungen an der Datenbank	14
4 API.....	15
4.1 Aufruf der einzelnen Services	15
4.2 Übersicht über die einzelnen Serviceklassen	15
4.3 Verwendete HTTP Methoden	16
4.4 Namensschema der HTTP Methoden	16
4.5 Daten an Servicemethoden senden	17
4.5.1 Beispiel JSON Objekt zum Hinzufügen.....	17
4.5.2 Beispiel JSON Objekt zum Ändern eines Eintrages	17
4.6 Daten vom Service zurückerhalten	18
4.6.1 Beispiel im Erfolgsfall.....	18
4.6.2 Beispiel im Fehlerfall.....	18
4.7 Details zu Adresse	19
4.7.1 HTTP Requests für Adresse	19
4.7.2 Objekt vom Typ Adresse	19
4.8 Details zu Benutzer.....	21
4.8.1 HTTP Requests für Benutzer	21
4.8.2 Objekt vom Typ Benutzer	21
4.8.3 Abfrageobjekt vom Typ Benutzereindeutig.....	22
4.8.4 Abfrageobjekt vom Typ Benutzerzugang	22
4.9 Details zu Benutzerrolle.....	23
4.9.1 HTTP Requests für Benutzerrolle	23
4.9.2 Objekt vom Typ Benutzerrolle	23
4.10 Details zu Bestellposition	24

4.10.1	HTTP Requests für Bestellposition	24
4.10.2	Objekt vom Typ Bestellposition	24
4.11	Details zu Bestellung.....	25
4.11.1	HTTP Requests für Bestellung	25
4.11.2	Objekt vom Typ Bestellung	25
4.12	Details zu Bewertung	27
4.12.1	HTTP Requests für Bewertung	27
4.12.2	Objekt vom Typ Bewertung	27
4.13	Details zu Branche.....	29
4.13.1	HTTP Requests für Branche.....	29
4.13.2	Objekt vom Typ Branche	29
4.14	Details zu Einheit.....	30
4.14.1	HTTP Requests für Einheit	30
4.14.2	Objekt vom Typ Einheit.....	30
4.15	Details zu Filmgenre	31
4.15.1	HTTP Requests für Filmgenre	31
4.15.2	Objekt vom Typ Filmgenre.....	31
4.16	Details zu Firma	32
4.16.1	HTTP Requests für Firma	32
4.16.2	Objekt vom Typ Firma	32
4.17	Details zu Forumsbenutzer	34
4.17.1	HTTP Requests für Forumsbenutzer	34
4.17.2	Objekt vom Typ Forumsbenutzer.....	34
4.17.3	Abfrageobjekt vom Typ Forumsbenutzereindeutig	35
4.18	Details zu Forumsbereich.....	36
4.18.1	HTTP Requests für Forumsbereich	36
4.18.2	Objekt vom Typ Forumsbereich.....	36
4.19	Details zu Forumseintrag	38
4.19.1	HTTP Requests für Forumseintrag.....	38
4.19.2	Objekt vom Typ Forumseintrag	38
4.20	Details zu Gericht	40
4.20.1	HTTP Requests für Gericht	40
4.20.2	Objekt vom Typ Gericht	40
4.21	Details zu Kinosaal	42
4.21.1	HTTP Requests für Kinosaal	42
4.21.2	Objekt vom Typ Kinosaal.....	42
4.22	Details zu Land	44
4.22.1	HTTP Requests für Land	44
4.22.2	Objekt vom Typ Land.....	44
4.23	Details zu Mehrwertsteuer	45
4.23.1	HTTP Requests für Mehrwertsteuer	45
4.23.2	Objekt vom Typ Mehrwertsteuer	45
4.24	Details zu Person.....	46
4.24.1	HTTP Requests für Person.....	46
4.24.2	Objekt vom Typ Person	46

4.25	Details zu Produkt	48
4.25.1	HTTP Requests für Produkt	48
4.25.2	Objekt vom Typ Produkt.....	48
4.26	Details zu Produktbild	50
4.26.1	HTTP Requests für Produktbild	50
4.26.2	Objekt vom Typ Produktbild.....	50
4.27	Details zu Produktkategorie	51
4.27.1	HTTP Requests für Produktkategorie	51
4.27.2	Objekt vom Typ Produktkategorie.....	51
4.28	Details zu Reservierer	52
4.28.1	HTTP Requests für Reservierer	52
4.28.2	Objekt vom Typ Reservierer	52
4.29	Details zu ReservierterSitz.....	53
4.29.1	HTTP Requests für ReservierterSitz.....	53
4.29.2	Objekt vom Typ ReservierterSitz	53
4.30	Details zu Reservierung.....	54
4.30.1	HTTP Requests für Reservierung.....	54
4.30.2	Objekt vom Typ Reservierung	54
4.31	Details zu Speisenart	56
4.31.1	HTTP Requests für Speisenart	56
4.31.2	Objekt vom Typ Speisenart.....	56
4.32	Details zu Termin	57
4.32.1	HTTP Requests für Termin	57
4.32.2	Objekt vom Typ Termin.....	57
4.32.3	Abfrageobjekt vom Typ Terminbereich	58
4.33	Details zu Vorstellung	59
4.33.1	HTTP Requests für Vorstellung	59
4.33.2	Objekt vom Typ Vorstellung.....	60
4.33.3	Abfrageobjekt vom Typ Sitzfrei	61
4.34	Details zu Zahlungsart	62
4.34.1	HTTP Requests für Zahlungsart	62
4.34.2	Objekt vom Typ Zahlungsart.....	62
4.35	Details zu Zutat.....	63
4.35.1	HTTP Requests für Zutat	63
4.35.2	Objekt vom Typ Zutat.....	63
4.36	Details zu Zutatenliste	64
4.36.1	HTTP Requests für Zutatenliste	64
4.36.2	Objekt vom Typ Zutatenliste.....	64
5	Dateiuploads	65
5.1	Node.js Middleware express-fileupload	65
5.2	HTML Formulare für Dateiuploads	65
5.3	Zugriff auf Dateiuploads auf der Serverseite	66
5.3.1	Aufbau eines original Datei – JSON Objektes.....	66
5.3.2	Hilfsbibliothek fileHelper.js	66
5.4	Weitere Details zu Fileuploads mit express-fileupload	67
5.5	Dateien und Datenbanken	67

6	Anhang	68
6.1	Datenbankmodell	68
6.2	Beispiel eines SQL Create Table Statements.....	69
6.3	Beispiel eines SQL Insert Statements	69
6.4	Beispiele zu Node.js Codes	69
6.4.1	Datenbankverbindung	69
6.4.2	Webserver erstellen und starten.....	69
6.4.3	Service Klassen / Handler in den Server einbinden	70
6.4.4	Service Methode innerhalb einer Service – Klasse.....	70
6.4.5	Beispielhafte DAO Klasse.....	71
6.4.6	Postman.....	72
6.4.7	Beispielhafter webbasierter Client.....	72
6.5	Links und weitere Hilfen	73

1 Einführung

Dieses Dokument soll Ihnen zeigen, wie Sie mit dem **Backend/api im Praktikum WebAnwendungen 2**, arbeiten können.

Das Backend wurde in Node.JS erstellt und arbeitet mit einer SQLite Datenbankdatei. Es implementiert mehrere **CRUD bzw. RESTful WebServices**, basierend auf den einzelnen Datenbanktabellen, welche in der SQLite Datenbankdateien gegeben sind.

Das Backend / die API ist also eine servseitige Anwendungen, welche als Schnittstelle zwischen dem Frontend / Client und einer Datenbank (SQLite) dient.

1.1 Node.js

Das Backend wurde komplett mit Node.JS umgesetzt. Es wurde unter Windows compiliert unter der Version **19.6.1 (x64)**

Bitte stellen Sie sicher, dass Sie die gleiche Version auf Ihrem System verwenden.

1.1.1 Node.js Module

Als Erweiterungen zum Node.JS wurden Projektseitig noch folgende Module per NPM mit installiert:

Better-SQLite, Express, Express Fileupload, Body-Parser, Cors, Lodash, Luxon, MD5, NodeMon und Morgan

Sollten Sie bei der Ausführung Probleme erhalten, prüfen Sie bitte ob Sie die richtige Node.js Version verwenden. Sollte diese nicht passen kann es zu Problemen kommen!

1.2 SQLite

Als Datenbanksystem wird SQLite verwendet. Ein einfaches und Dateibasiertes Datenbanksystem.

1.2.1 Datenbank GUI / DBeaver / JAR Treiber

Um die Datenbankdatei einfach administrieren zu können, empfehlen wir Ihnen das Programm „DBeaver Community Edition“.

Weiterhin benötigen Sie für den Zugriff auf SQLite Dateien aus DBeaver heraus eine Java Treiber / JAR – Datei.

Den Link zu DBeaver und die JAR Datei finden Sie bei den Praktikums – Unterlagen auf Ilias.

1.2.2 Datentypen

SQLite arbeitet nur mit „einfachen“ Datentypen. Heißt bei den einzelnen Tabellen wurden nur die Datentypen INT, TEXT und REAL verwendet.

Selbst ein Datum oder ein Zeitstempel wird datenbankseitig als Text gespeichert. Und zwar in der SQL – Notation, also „jjjj-mm-dd hh:mm:ss“.

1.3 Verzeichnisstruktur des Backend

Das Backend ist wie folgt aufgebaut:

- /
 - Im Hauptordner „WebAnw2Backend“ befindet sich die Server – Anwendung, die NPM Konfiguration und eine Hilfsbibliothek mit oft genutzten Funktionen
- /dao
 - Im dao – Unterordner sind die einzelnen DAO Dateien für den Zugriff auf die Datenbanktabellen
- /db
 - In diesem Ordner liegt die SQLite Datenbankdatei
- /node_module
 - Hier befinden sich die installierten Node.js Module
- /services
 - In diesem Verzeichnis befinden sich die einzelnen Services – Dateien, auch Endpoints genannt, welche die jeweiligen Methoden / Funktionen enthalten

1.4 Verwendeter Editor / IDE

Das Backend wurde mit Visual Studio Code umgesetzt. Sie können aber jeden anderen beliebigen Text-Editor für die Arbeit verwenden.

1.5 Sprachen und Standards

Um die Sache etwas interessanter zu gestalten wurden folgende Standards festgelegt

- Die Datenbanktabellen und Spalten sind in Deutsch und groß- und klein Geschrieben
- Jede Datenbanktabelle hat als Primärschlüssel die Spalte „id“
- Fremdschlüssel werden mit xxxId bezeichnet
- Bei Fremdschlüsseln ist keine Kaskadierung hinterlegt
- Manche Spalten werden mit Standardwerten belegt, manche können NULL sein, manche nicht.
- Die selbst erstellten Dateien in den einzelnen Ordnern sind ebenfalls kleingeschrieben und in Deutsch
- Für die meisten Datenbanktabellen gibt es eine dazu passende Service – Datei sowie eine DAO Datei
- Die Kodierung innerhalb der JS Dateien ist auf englisch
- Ebenfalls alle Konsolen – Ausgaben sind auf englisch
- Der Datentransport zwischen Client und Server geht entweder über URL Parameter bzw. JSON Objekte
- Hier ist auch alles in deutsch und klein geschrieben
- Die Eigenschaftennamen der JSON Objekte entsprechen im Normalfall den Spaltennamen in der jeweiligen Tabelle
- Aufrufe von Serviceklassen und Servicemethoden sind ebenfalls in deutsch gehalten

2 Datenbankstruktur

Den Aufbau der einzelnen Tabellen, also die Struktur, kann für verschiedene Themengebiete verwendet werden. Sie müssen nicht ALLES mit Ihrem Projekt abdecken, Sie können sich das passendste herauspicken und verwenden. Wenn nun aber die bestehenden Tabellen Ihre Anforderungen nicht erfüllen können Sie diese problemlos erweitern, ersetzen oder anpassen. Bitte denken Sie aber daran, dass Sie dann auch wieder die Node.js Dateien entsprechend anpassen müssen!

Einen kompetten Überblick über die einzelnen Tabellen finden Sie in Ilias bzw. im Anhang.

Ebenfalls in Ilias gibt es eine SQL Datei, mit allen Create Table Statements, um die Tabellen anzulegen.

Sowie eine SQL Datei mit Insert – Statements um die einzelnen Tabellen rudimentär mit Daten zu bestücken. Hinweis: diese ist nicht vollständig!!!

Themengebiet	Tabellen
Produktdarstellung	Produktkategorie Produkt Produktbild ggf. Download
Shop	Produktkategorie Produkt Produktbild ggf. Download Mehrwertsteuer Bestellung Bestellposition ggf. Zahlungsart und Person
Personen / Adressverwaltung	Person Adresse Land
Downloads	Download
Terminverwaltung	Termin
Dienstleister – Buchungssystem	Firma Branche Termin
Rezepte	Speisenart Gericht Zutat Einheit Zutatenliste Bewertung
Login	Benutzerrolle Benutzer
Forum	Benutzerrolle Forumsbenutzer Forumsbereich Forumseintrag
Kino / Sitzreservierung	Kinosaal Filmgenre Film Vorstellung Reservierung Reservierer ReservierterSitz

2.1 Dateiuploads und Datenbank

Das Backend demonstriert wie ein Dateiupload verarbeitet werden kann. Egal ob nun eine einzelne Datei oder auch mehrere.

Mit den aufgeladenen Dateien wird aber prinzipiell nichts gemacht.

Es wäre möglich die Dateien direkt auf der Festplatte zu speichern in einem gesonderten Ordner und einem eindeutigen Namen und dann diese Information, also den Pfad zur Datei in der Datenbank zu speichern.

Alternativ können auch die Binärdaten, der Dateiname, Mimetype und die Dateigröße direkt in der Datenbank gespeichert werden. Die Binärdaten z.B. in einer BLOB Spalte.

Beide Möglichkeiten wurden aber hier NICHT umgesetzt!

3 Installation und Inbetriebnahme

3.1 Vorbedingung

Stellen Sie sicher, dass auf Ihrem System „Node.js“ installiert ist und die **PATH – Umgebungsvariable** auf den Installationsordner verweist

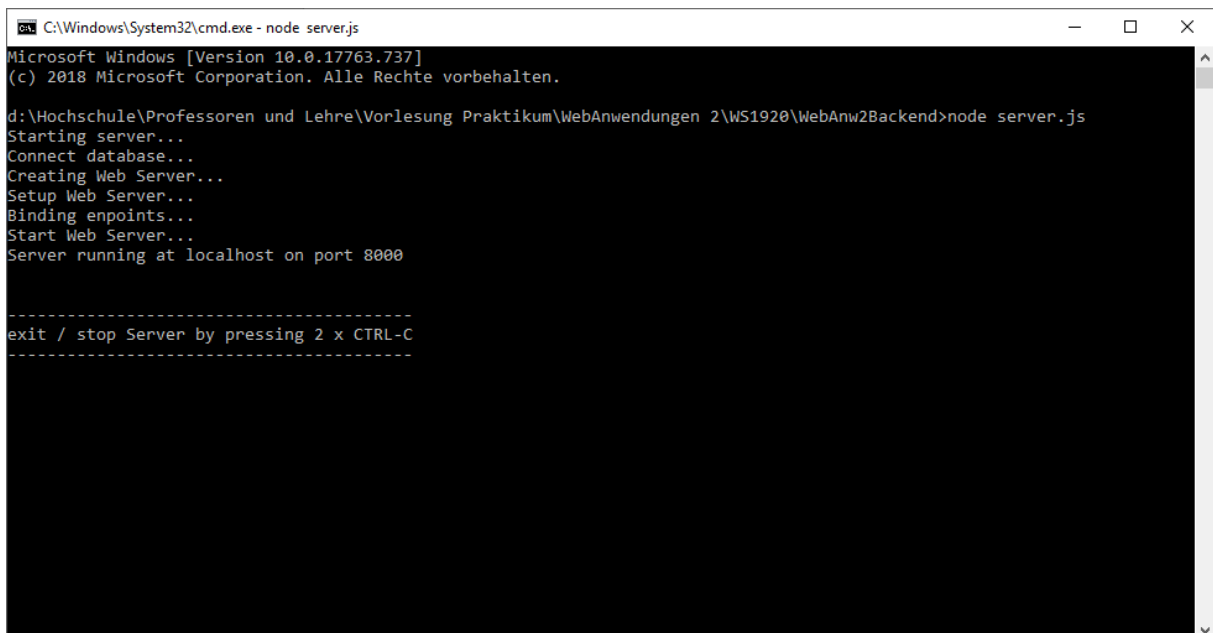
Ebenfalls achten Sie darauf, dass Sie die gleiche Version verwenden, in welcher das Backend erstellt wurde.

Sollten Sie eine andere Konfiguration haben passen Sie diese bitte an.

3.2 Installation und Betrieb des Backends

- Laden Sie die ZIP Datei „WebAnw2Backend.zip“ aus Ilias auf Ihren Rechner
- Entpacken Sie die Datei in einen Ordner Ihrer Wahl
- Öffnen Sie ein Kommandozeilen – Fenster **in** dem Ordner, wo sich die Datei server.js befindet
- Starten Sie den Server durch eingabe des Befehles „**npm start**“

Sie sollten dann folgendes Bild erhalten



```

C:\Windows\System32\cmd.exe - node server.js
Microsoft Windows [Version 10.0.17763.737]
(c) 2018 Microsoft Corporation. Alle Rechte vorbehalten.

d:\Hochschule\Professoren und Lehre\Vorlesung Praktikum\WebAnwendungen 2\WS1920\WebAnw2Backend>node server.js
Starting server...
Connect database...
Creating Web Server...
Setup Web Server...
Binding endpoints...
Start Web Server...
Server running at localhost on port 8000

-----
exit / stop Server by pressing 2 x CTRL-C
-----

```

Tip: Sollten Sie eine Umgebungsvariable zu Node.js NICHT gesetzt haben können Sie alternativ zu einer Standard Windows Kommandozeile auch direkt ein Node Kommandozeilenfenster öffnen. Gehen Sie hierzu auf das Startmenü und öffnen Sie „Node.js Command Prompt“

Hinweis: Sollten Sie hier andere Meldungen oder sogar Fehler erhalten ist Ihre Umgebung inkompatibel zum Projekt. Wenden Sie sich in diesem Fall an Ihren Betreuer.

In diesem Fenster erscheinen auch alle Ausgaben, welche die serverseitige Applikation von sich gibt.

3.2.1 Hinweise zur Arbeit am Backend (Developer – Mode)

Während der Entwicklungszeit, also der Zeit an dem Sie am Backend / der API arbeiten ist es unumgänglich nach jeder Änderung den Server **neu zu starten**, damit die Änderungen auch übernommen werden.

Damit Sie dies nicht immer wieder manuell durchführen müssen, wurde ein weiteres Modul mit installiert, welches sich „nodemon“ nennt, und Ihnen diese Arbeit abnimmt.

Mit folgendem Befehl starten Sie das Backend im Entwickler - Modus (Developer - Mode):

```
npm run dev
```

Nun erkennt nodemon die Änderungen im Code und startet den Server immer wieder neu. Sollte die Automatik versagen können Sie einen Neustart erzwingen durch eingabe von „rs“ und Drücken der Enter Taste.

3.2.2 Hinweis zur normalen Ausführung des Servers (Production – Mode)

Wenn Sie die Entwicklung am Backend abgeschlossen haben muss der Server nicht mehr ständig neu gestartet werden und kann daher ganz normal ausgeführt werden.

Bitte geben Sie hierzu in der Node.js Kommandozeile folgenden Befehl ein:

```
npm start
```

3.3 Aufruf des Backends / der API mit dem Browser

Das Backend ist ein rudimentärer Web Server, welcher nun auf Ihrem Rechner läuft. Er horcht auf dem Port 8000.

Sie können diesen nun einfach mit einem Browser aufrufen.
Verwenden Sie hierzu die Adresse

<http://localhost:8000>

Sollte alles klappen, dann erhalten Sie im Browser ein JSON Objekt zurück, welches angibt, dass der Server läuft.

Sollten Sie einen Timeout erhalten oder sonst eine Fehlermeldung haben Sie sich entweder in der Adresse vertippt oder der Server an sich arbeitet nicht auf ihrem Rechner.

ACHTUNG: SOLLTE EINE FIREWALL AUF IHREM RECHNER DIE KOMMUNIKATION BZW. DEN SERVER BLOCKIEREN, FÜGEN SIE ENTSPRECHENDE AUSNAHMEN AUF IHRER FIREWALL HINZU.

3.4 Server beenden

Um den Server zu beenden genügt es, dass Sie im Konsolenfenster, wo die Serveranwendung läuft zweimal STRG – C drücken.

Nach dem Serverende können Sie das Konsolenfenster mit dem Befehl exit schließen

3.5 Änderungen am Backend / Code

Sie können jederzeit den Code des Backends Ihren Bedürfnissen anpassen und ändern. Denken Sie aber daran, dass die Änderungen erst wirksam werden, wenn Sie den Server beenden und neu starten (Außer Sie verwenden nodemon, also den Developer – Modus (siehe Kapitel 3.2.1).

3.6 Änderungen an der Datenbank

Da die SQLite Datenbank eine Datei ist, sind Änderungen an dieser zur Laufzeit möglich. Ein Serverstart ist nicht unbedingt notwendig

4 API

4.1 Aufruf der einzelnen Services

Die jeweiligen Services werden mit einem HTTP Request aufgerufen. Dieser ist wie folgt aufgebaut

http://[serveradresse]:[port]/[backendname]/[servicename]/[servicemethode]/[opt. Werte]

Wenn Sie also z.B. den Service „Land“ aufrufen wollen um Objekte aller Länder zu erhalten verwenden Sie

http://localhost:8000/api/land/alle

4.2 Übersicht über die einzelnen Serviceklassen

Folgende Serviceklassen sind implementiert und werden als sogenannte Endpoints eingebunden:

- Adresse
- Benutzer
- Benutzerrolle
- Bestellung
- Bewertung
- Branche
- Download
- Einheit
- Film
- Filmgenre
- Firma
- Forumsbenutzer
- Forumsbereich
- Gericht
- Kinosaal
- Land
- Mehrwertsteuer
- Person
- Produkt
- Produktbild
- Produktkategorie
- Reservierer
- Reservierung
- Speisenart
- Termin
- Zahlungsart
- Zutat
- Vorstellung

4.3 Verwendete HTTP Methoden

Folgende HTTP Methoden sind derzeit (im Normalfall) bei allen Services implementiert

GET	Daten aus Datenbank abrufen
POST	Neuen Eintrag in Datenbank erzeugen
PUT	Bestehenden Eintrag in Datenbank ändern
DELETE	Bestehenden Eintrag in Datenbank löschen

Andere HTTP Methoden könnten natürlich auch verwendet werden, für die Umsetzung eines einfachen Web – Projektes müssten diese aber vollauf genügen.

4.4 Namensschema der HTTP Methoden

Nochmals zur Erinnerung. Alle HTTP Aufrufe sind deutsch und kleingeschrieben!

Daten abrufen vom Backend (GET)

Objekt vom Typ Land mit der ID 3 holen
<http://localhost:8000/api/land/gib/3>

Alle Land – Objekte holen
<http://localhost:8000/api/land/alle>

Prüfen ob ein Eintrag mit der ID 7 existiert
<http://localhost:8000/api/land/existiert/7>

Neuen Eintrag erstellen (POST)

<http://localhost:8000/api/land>
 Wobei die Daten hier als JSON Objekt vom jeweiligen Typ geliefert werden müssen, ohne ID

Bestehenden Eintrag ändern (PUT)

<http://localhost:8000/api/land>
 Wobei die Daten hier als JSON Objekt vom jeweiligen Typ geliefert werden müssen, mit ID

Bestehenden Eintrag löschen (DELETE)

<http://localhost:8000/api/land/4>

4.5 Daten an Servicemethoden senden

Generell werden Daten auf zwei Arten an den jeweiligen Service geschickt

Entweder als Teil des RESTFul Aufrufs (z.b. in Form einer ID)

http://localhost:8000/api/land/gib/2

oder als

JSON Objekt, welches im HTTP Body übertragen wird

Bei den JSON Objekten sind alle Eigenschaftsnamen ebenfalls kleingeschrieben.

Manche Eigenschaften müssen Daten enthalten (sind also Pflicht), manche können leer bleiben und müssen auch nicht geschickt werden

Manche sind nullbar.

4.5.1 Beispiel JSON Objekt zum Hinzufügen

```
{
  "strasse": "Lange Straße",
  "hausnummer": "17",
  "plz": "72336",
  "ort": "Balingen",
  "land": {
    "id": 44
  }
}
```

4.5.2 Beispiel JSON Objekt zum Ändern eines Eintrages

```
{
  "id": 5,
  "strasse": "Kurze Straße",
  "hausnummer": "5",
  "plz": "72458",
  "ort": "Ebingen",
  "land": {
    "id": 44
  }
}
```

Die Angabe der ID ist hier wichtig!!!

4.6 Daten vom Service zurückerhalten

Alles was vom Backend zurück geliefert wird, ist immer ein JSON Objekt.

Es gibt prinzipiell zwei Typen von Objekten. Im Erfolgsfall, wenn die Verarebeitung geklappt hat, ein Objekt mit den angeforderten Daten. Oder im Fehlerfall ein Objekt mit einer Fehlermeldung

4.6.1 Beispiel im Erfolgsfall

```
{
  "nachricht": "OK",
  "fehler": false,
  "daten": {
    "strasse": "Lange Straße",
    "hausnummer": "17",
    "plz": "72336",
    "ort": "Balingen",
    "land": {
      "id": 44
    }
  }
}
```

4.6.2 Beispiel im Fehlerfall

```
{
  "nachricht": "Fehler: vorname fehlt",
  "fehler": true,
  "daten": null
}
```

4.7 Details zu Adresse

4.7.1 HTTP Requests für Adresse

Methode	Request
GET	<code>.../api/adresse/gib/[id]</code> Liefert ein JSON Objekt vom Typ Adresse für die angegebene [id]
GET	<code>.../api/adresse/alle</code> Liefert alle JSON Objekte vom Typ Adresse
GET	<code>../api/adresse/existiert/[id]</code> Prüft nach, ob ein Objekt vom Typ Adresse unter dieser [id] existiert Liefert true oder false zurück
POST	<code>../api/adresse</code> Fügt die als JSON gesendeten Daten als neues Objekt hinzu Liefert das neu erstellte JSON Objekt vom Typ Adresse zurück
PUT	<code>../api/adresse</code> Ändert einen bestehenden JSON Objekt anhand der gesendeten Daten Liefert das geänderte JSON Objekt vom Typ Adresse zurück
DELETE	<code>../api/adresse/[id]</code> Löscht das durch die [id] spezifizierte JSON Objekt vom Typ Adresse und liefert true zurück

4.7.2 Objekt vom Typ Adresse

Beispiel

```
{
  "id": 3,
  "strasse": "Lange Straße",
  "hausnummer": "5",
  "adresszusatz": "rechter Eingang",
  "plz": "72336",
  "ort": "Balingen",
  "land": {
    // Objekt vom Typ Land
  }
}
```

Abhängigkeiten

Objekt vom Typ Land

Attribute

id

Eindeutiger Primärschlüssel des Objektes

Typ: Integer, Pflichtfeld bei: PUT, Nullbar: nein, Defaultwert: keiner

strasse

Angabe der Straße

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

hausnummer

Hausnummer innerhalb der Straße

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

adresszusatz

Ein beliebiger Text

Typ: Text, kein Pflichtfeld, Nullbar: nein, Defaultwert: leerer String

plz

die Postleitzahl des Ortes

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

ort

Der Name des Ortes

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

land

Objekt vom Typ Land

Typ: Objekt, Pflichtfeld bei: POST und PUT (min. die id), Nullbar: nein, Defaultwert: keiner

4.8 Details zu Benutzer

4.8.1 HTTP Requests für Benutzer

Methode	Request
GET	<code>.../api/benutzer/gib/[id]</code> Liefert ein JSON Objekt vom Typ Benutzer für die angegebene [id]
GET	<code>.../api/benutzer/alle</code> Liefert alle JSON Objekte vom Typ Benutzer
GET	<code>../api/benutzer/existiert/[id]</code> Prüft nach, ob ein Objekt vom Typ Benutzer unter dieser [id] existiert Liefert true oder false zurück
GET	<code>.../api/benutzer/eindeutig</code> Prüft nach ob ein gelieferter Benutzername eindeutig ist, heißt nicht bereits in der Datenbank gespeichert ist. Benötigt den zu prüfenden Benutzernamen als Objekt Benutzereindeutig Liefert nach der Prüfung true zurück, falls der Benutzername noch nicht verwendet wird, oder aber false
GET	<code>.../api/benutzer/zugang</code> Prüft nach ob ein gelieferter Benutzername / Passwort Zugang zum System haben, heißt in der Datenbank gespeichert sind. Benötigt den zu prüfenden Benutzernamen und das Passwort als Objekt Benutzerzugang Liefert nach Prüfung true zurück, falls der Benutzer zugang hat, oder aber false
POST	<code>../api/benutzer</code> Fügt die als JSON gesendeten Daten als neues Objekt hinzu Liefert das neu erstellte JSON Objekt vom Typ Benutzer zurück
PUT	<code>../api/benutzer</code> Ändert einen bestehenden JSON Objekt anhand der gesendeten Daten Liefert das geänderte JSON Objekt vom Typ Benutzer zurück
DELETE	<code>../api/benutzer/[id]</code> Löscht das durch die [id] spezifizierte JSON Objekt vom Typ Benutzer und liefert true zurück

4.8.2 Objekt vom Typ Benutzer

Beispiel

```
{
  "id": 3,
  "benutzername": "master@blaster.de",
  "passwort": "FA64BB23EE67....",
  "benutzerrolle": {
    // Objekt vom Typ Benutzerrolle
  },
  "person": {
    // Objekt vom Typ Person
  }
}
```

Abhängigkeiten

Objekt vom Typ Benutzerrolle

Objekt vom Typ Person

Attribute

id

Eindeutiger Primärschlüssel des Objektes

Typ: Integer, Pflichtfeld bei: PUT, Nullbar: nein, Defaultwert: keiner

benutzername

Name des Users, muss eindeutig sein, darf also nicht doppelt in der Datenbank vorkommen

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

passwort / neuespasswort

Das Passwort welches gespeichert werden soll

Die Eigenschaft „passwort“ darf nur bei POST verwendet werden, heißt ein neuer Benutzer wird gespeichert

Die Eigenschaft „neuespasswort“ darf nur bei PUT verwendet werden, heißt wenn ein bestehender Eintrag geändert wird kann hier auch das Passwort aktualisiert werden

Passworte werden in der Datenbank immer verschlüsselt gespeichert. Heißt beim Senden wird es in Klartext übertragen, beim lesen aus der Datenbank sieht man nur das gecryptete.

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

benutzerrolle

Objekt vom Typ Benutzerrolle

Typ: Objekt, Pflichtfeld bei: POST und PUT (min. die id), Nullbar: nein, Defaultwert: keiner

person

Objekt vom Typ Person bzw. null

Typ: Objekt, kein Pflichtfeld, Nullbar: ja, Defaultwert: null

4.8.3 Abfrageobjekt vom Typ Benutzereindeutig**Beispiel**

```
{
    "benutzername": "master@blaster.de"
}
```

4.8.4 Abfrageobjekt vom Typ Benutzerzugang**Beispiel**

```
{
    "benutzername": "master@blaster.de",
    "passwort": "test1234"
}
```

4.9 Details zu Benutzerrolle

4.9.1 HTTP Requests für Benutzerrolle

Methode	Request
GET	<code>.../api/benutzerrolle/gib/[id]</code> Liefert ein JSON Objekt vom Typ Benutzerrolle für die angegebene [id]
GET	<code>.../api/benutzerrolle/alle</code> Liefert alle JSON Objekte vom Typ Benutzerrolle
GET	<code>../api/benutzerrolle/existiert/[id]</code> Prüft nach, ob ein Objekt vom Typ Benutzerrolle unter dieser [id] existiert Liefert true oder false zurück
POST	<code>../api/benutzerrolle</code> Fügt die als JSON gesendeten Daten als neues Objekt hinzu Liefert das neu erstellte JSON Objekt vom Typ Benutzerrolle zurück
PUT	<code>../api/benutzerrolle</code> Ändert einen bestehenden JSON Objekt anhand der gesendeten Daten Liefert das geänderte JSON Objekt vom Typ Benutzerrolle zurück
DELETE	<code>../api/benutzerrolle/[id]</code> Löscht das durch die [id] spezifizierte JSON Objekt vom Typ Benutzerrolle und liefert true zurück

4.9.2 Objekt vom Typ Benutzerrolle

Beispiel

```
{
  "id": 1,
  "bezeichnung": "Administrator"
}
```

Abhängigkeiten

Keine

Attribute

id

Eindeutiger Primärschlüssel des Objektes

Typ: Integer, Pflichtfeld bei: PUT, Nullbar: nein, Defaultwert: keiner

bezeichnung

Bezeichnung der Rolle

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

4.10 Details zu Bestellposition

4.10.1 HTTP Requests für Bestellposition

Zu Bestellpositionen gibt es keinen eigenen Service. Objekte diesen Typs sind Teil von Bestellungen.

4.10.2 Objekt vom Typ Bestellposition

Beispiel

```
{
  "id": 33,
  "bestellung": {
    // Teilobjekt vom Typ Bestellung, enthält nur die Eigenschaft id
  },
  "produkt": {
    // Objekt vom Typ Produkt
  },
  "menge": 5,
  "mehrwertsteuersumme": 17.92,
  "nettosumme": 210.02,
  "bruttosumme": 227.94
}
```

Abhängigkeiten

Teilobjekt vom Typ Bestellung, nur die ID, zwecks Referenzierung

Objekt vom Typ Produkt

Attribute

id

Eindeutiger Primärschlüssel des Objektes

Typ: Integer, Pflichtfeld bei: PUT, Nullbar: nein, Defaultwert: keiner

bestellung

Teilobjekt vom Typ Bestellung. Benötigt nur die Eigenschaft id zwecks der Referenzierung

Typ: Objekt, Pflichtfeld bei: POST und PUT (min. die id), Nullbar: nein, Defaultwert: keiner

produkt

Objekt vom Typ Produkt

Typ: Objekt, Pflichtfeld bei: POST und PUT (min. die id), Nullbar: nein, Defaultwert: keiner

menge

Anzahl der Produkte

Typ: Integer, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: 1

mehrwertsteuersumme

Wird aus dem Mehrwertsteueranteil des Produktes und der Menge errechnet

Wird NICHT in der Datenbank gehalten

Typ: Real, nur Anzeige

nettosumme

Wird aus dem Nettopreis des Produktes und der Menge errechnet

Typ: Real, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

bruttosumme

Wird aus dem Bruttopreis des Produktes und der Menge errechnet

Wird NICHT in der Datenbank gehalten

Typ: Real, nur Anzeige

4.11 Details zu Bestellung

4.11.1 HTTP Requests für Bestellung

Methode	Request
GET	<code>.../api/bestellung/gib/[id]</code> Liefert ein JSON Objekt vom Typ Bestellung für die angegebene [id]
GET	<code>.../api/bestellung/alle</code> Liefert alle JSON Objekte vom Typ Bestellung
GET	<code>../api/bestellung/existiert/[id]</code> Prüft nach, ob ein Objekt vom Typ Bestellung unter dieser [id] existiert Liefert true oder false zurück
POST	<code>../api/bestellung</code> Fügt die als JSON gesendeten Daten als neues Objekt hinzu Liefert das neu erstellte JSON Objekt vom Typ Bestellung zurück
PUT	<code>../api/bestellung</code> Ändert einen bestehenden JSON Objekt anhand der gesendeten Daten Liefert das geänderte JSON Objekt vom Typ Bestellung zurück
DELETE	<code>../api/bestellung/[id]</code> Löscht das durch die [id] spezifizierte JSON Objekt vom Typ Bestellung und liefert true zurück

4.11.2 Objekt vom Typ Bestellung

Beispiel

```
{
  "id": 33,
  "bestellzeitpunkt": "15.10.2019 08:17:44",
  "besteller": {
    // Objekt vom Typ Person
  },
  "zahlungsart": {
    // Objekt vom Typ Zahlungsart
  },
  "bestellpositionen": [
    { // Objekt vom Typ Bestellposition },
    { // Objekt vom Typ Bestellposition }, ...
  ],
  "total": {
    "mehrwertsteuer": 25.0,
    "netto": 200.0,
    "brutto": 225.0
  }
}
```

Abhängigkeiten

Objekt vom Typ Person (falls hinterlegt)

Objekt vom Typ Zahlungsart

Objekt/e vom Typ Bestellposition

Attribute

id

Eindeutiger Primärschlüssel des Objektes

Typ: Integer, Pflichtfeld bei: PUT, Nullbar: nein, Defaultwert: keiner

bestellzeitpunkt

Der Zeitpunkt im Format tt.mm.jjjj hh:mm:ss, an welchem die Bestellung ausgelöst wurde

Wird kein Zeitstempel geliefert, wird automatisch der aktuelle Systemzeitpunkt verwendet

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: Systemzeit

besteller

Objekt vom Typ Person, welcher die Bestellung tätigt, bzw. null

Typ: Objekt, kein Pflichtfeld, Nullbar: ja, Defaultwert: null

zahlungsart

Objekt vom Typ Zahlungsart

Typ: Objekt, Pflichtfeld bei: POST und PUT (min. die id), Nullbar: nein, Defaultwert: keiner

bestellpositionen

Array, mit einem oder mehreren Objekten vom Typ Bestellposition

Typ: Array mit Objekten, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

total

Informatives Unterobjekt, welches die Bestellbruttosumme, Bestellnettosumme und Bestellmehrwertsteuersumme enthält

Wird NICHT in der Datenbank gehalten

Typ: Unterobjekt, nur Anzeige

4.12 Details zu Bewertung

4.12.1 HTTP Requests für Bewertung

Methode	Request
GET	<code>.../api/bewertung/gib/[id]</code> Liefert ein JSON Objekt vom Typ Bewertung für die angegebene [id]
GET	<code>.../api/bewertung/alle</code> Liefert alle JSON Objekte vom Typ Bewertung
GET	<code>../api/bewertung/existiert/[id]</code> Prüft nach, ob ein Objekt vom Typ Bewertung unter dieser [id] existiert Liefert true oder false zurück
GET	<code>../api/bewertung/gericht/[id]</code> Liefert alle JSON Objekte vom Typ Bewertung, welche zu einem bestimmten Gericht (identifiziert anhand der [id]) gehören
POST	<code>../api/bewertung</code> Fügt die als JSON gesendeten Daten als neues Objekt hinzu Liefert das neu erstellte JSON Objekt vom Typ Bewertung zurück
PUT	<code>../api/bewertung</code> Ändert einen bestehenden JSON Objekt anhand der gesendeten Daten Liefert das geänderte JSON Objekt vom Typ Bewertung zurück
DELETE	<code>../api/bewertung/[id]</code> Löscht das durch die [id] spezifizierte JSON Objekt vom Typ Bewertung und liefert true zurück

4.12.2 Objekt vom Typ Bewertung

Beispiel

```
{
  "id": 7,
  "gericht": {
    // Teilobjekt vom Typ Gericht, enthält nur die Eigenschaft id
  },
  "punkte": 5,
  "zeitpunkt": "15.10.2019 08:17:44",
  "bemerkung": "Tolles Gericht",
  "ersteller": "Hans"
}
```

Abhängigkeiten

Teilobjekt vom Typ Gericht, nur die ID, zwecks Referenzierung

Attribute**id**

Eindeutiger Primärschlüssel des Objektes

Typ: Integer, Pflichtfeld bei: PUT, Nullbar: nein, Defaultwert: keiner

gericht

Teilobjekt vom Typ Gericht. Benötigt nur die Eigenschaft id zwecks der Referenzierung

Typ: Objekt, Pflichtfeld bei: POST und PUT (min. die id), Nullbar: nein, Defaultwert: keiner

punkte

Die Anzahl der Punkte, welche bei der Bewertung vergeben wurden

Muss eine Zahl größer 0 sein

Typ: Integer, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

zeitpunkt

Der Zeitpunkt im Format tt.mm.jjjj hh:mm:ss, an welchem die Bewertung erstellt wurde

Wird kein Zeitstempel geliefert, wird automatisch der aktuelle Systemzeitpunkt verwendet

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: Systemzeit

bemerkung

Bemerkung zur Bewertung vom Ersteller, kann null sein

Typ: Text, kein Pflichtfeld, Nullbar: ja, Defaultwert: null

ersteller

Name des Erstellers, kann null sein

Typ: Text, kein Pflichtfeld, Nullbar: ja, Defaultwert: null

4.13 Details zu Branche

4.13.1 HTTP Requests für Branche

Methode	Request
GET	<code>.../api/branche/gib/[id]</code> Liefert ein JSON Objekt vom Typ Branche für die angegebene [id]
GET	<code>.../api/branche/alle</code> Liefert alle JSON Objekte vom Typ Branche
GET	<code>../api/branche/existiert/[id]</code> Prüft nach, ob ein Objekt vom Typ Branche unter dieser [id] existiert Liefert true oder false zurück
POST	<code>../api/branche</code> Fügt die als JSON gesendeten Daten als neues Objekt hinzu Liefert das neu erstellte JSON Objekt vom Typ Branche zurück
PUT	<code>../api/branche</code> Ändert einen bestehenden JSON Objekt anhand der gesendeten Daten Liefert das geänderte JSON Objekt vom Typ Branche zurück
DELETE	<code>../api/branche/[id]</code> Löscht das durch die [id] spezifizierte JSON Objekt vom Typ Branche und liefert true zurück

4.13.2 Objekt vom Typ Branche

Beispiel

```
{
  "id": 3,
  "bezeichnung": "Metallbau"
}
```

Abhängigkeiten

Keine

Attribute

id

Eindeutiger Primärschlüssel des Objektes

Typ: Integer, Pflichtfeld bei: PUT, Nullbar: nein, Defaultwert: keiner

bezeichnung

Name der Branche

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

4.14 Details zu Einheit

4.14.1 HTTP Requests für Einheit

Methode	Request
GET	<code>.../api/einheit/gib/[id]</code> Liefert ein JSON Objekt vom Typ Einheit für die angegebene [id]
GET	<code>.../api/einheit/alle</code> Liefert alle JSON Objekte vom Typ Einheit
GET	<code>../api/einheit/existiert/[id]</code> Prüft nach, ob ein Objekt vom Typ Einheit unter dieser [id] existiert Liefert true oder false zurück
POST	<code>../api/einheit</code> Fügt die als JSON gesendeten Daten als neues Objekt hinzu Liefert das neu erstellte JSON Objekt vom Typ Einheit zurück
PUT	<code>../api/einheit</code> Ändert einen bestehenden JSON Objekt anhand der gesendeten Daten Liefert das geänderte JSON Objekt vom Typ Einheit zurück
DELETE	<code>../api/einheit/[id]</code> Löscht das durch die [id] spezifizierte JSON Objekt vom Typ Einheit und liefert true zurück

4.14.2 Objekt vom Typ Einheit

Beispiel

```
{
  "id": 3,
  "bezeichnung": "kg"
}
```

Abhängigkeiten

Keine

Attribute

id

Eindeutiger Primärschlüssel des Objektes

Typ: Integer, Pflichtfeld bei: PUT, Nullbar: nein, Defaultwert: keiner

bezeichnung

Bezeichnung der Einheit

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

4.15 Details zu Filmgenre

4.15.1 HTTP Requests für Filmgenre

Methode	Request
GET	<code>.../api/filmgenre/gib/[id]</code> Liefert ein JSON Objekt vom Typ Filmgenre für die angegebene [id]
GET	<code>.../api/filmgenre/alle</code> Liefert alle JSON Objekte vom Typ Filmgenre
GET	<code>../api/eineit/existiert/[id]</code> Prüft nach, ob ein Objekt vom Typ Filmgenre unter dieser [id] existiert Liefert true oder false zurück
POST	<code>../api/filmgenre</code> Fügt die als JSON gesendeten Daten als neues Objekt hinzu Liefert das neu erstellte JSON Objekt vom Typ Filmgenre zurück
PUT	<code>../api/filmgenre</code> Ändert einen bestehenden JSON Objekt anhand der gesendeten Daten Liefert das geänderte JSON Objekt vom Typ Filmgenre zurück
DELETE	<code>../api/filmgenre/[id]</code> Löscht das durch die [id] spezifizierte JSON Objekt vom Typ Filmgenre und liefert true zurück

4.15.2 Objekt vom Typ Filmgenre

Beispiel

```
{
  "id": 2,
  "bezeichnung": "Komödien"
}
```

Abhängigkeiten

Keine

Attribute

id

Eindeutiger Primärschlüssel des Objektes

Typ: Integer, Pflichtfeld bei: PUT, Nullbar: nein, Defaultwert: keiner

bezeichnung

Bezeichnung des Filmgenres

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

4.16 Details zu Firma

4.16.1 HTTP Requests für Firma

Methode	Request
GET	<code>.../api/firma/gib/[id]</code> Liefert ein JSON Objekt vom Typ Firma für die angegebene [id]
GET	<code>.../api/firma/alle</code> Liefert alle JSON Objekte vom Typ Firma
GET	<code>../api/firma/existiert/[id]</code> Prüft nach, ob ein Objekt vom Typ Firma unter dieser [id] existiert Liefert true oder false zurück
POST	<code>../api/firma</code> Fügt die als JSON gesendeten Daten als neues Objekt hinzu Liefert das neu erstellte JSON Objekt vom Typ Firma zurück
PUT	<code>../api/firma</code> Ändert einen bestehenden JSON Objekt anhand der gesendeten Daten Liefert das geänderte JSON Objekt vom Typ Firma zurück
DELETE	<code>../api/firma/[id]</code> Löscht das durch die [id] spezifizierte JSON Objekt vom Typ Firma und liefert true zurück

4.16.2 Objekt vom Typ Firma

Beispiel

```
{
  "id": 5,
  "name": "HS Alstadt",
  "inhaber": null,
  "beschreibung": "Hier arbeiten wir",
  "adresse": {
    // Objekt vom Typ Adresse
  },
  "ansprechpartner": {
    // Objekt vom Typ Person, oder null
  },
  "branche": {
    // Objekt vom Typ Branche
  }
}
```

Abhängigkeiten

Objekte vom Typ Adresse und Branche
Optional auch vom Typ Person

Attribute

id

Eindeutiger Primärschlüssel des Objektes

Typ: Integer, Pflichtfeld bei: PUT, Nullbar: nein, Defaultwert: keiner

name

Der Name der Firma

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

inhaber

Der Name des Inhabers, oder null

Typ: Text, kein Pflichtfeld, Nullbar: ja, Defaultwert: null

beschreibung

Ein beliebiger Text

Typ: Text, kein Pflichtfeld, Nullbar: nein, Defaultwert: leerer String

adresse

Objekt vom Typ Adresse

Typ: Objekt, Pflichtfeld bei: POST und PUT (min. die id), Nullbar: nein, Defaultwert: keiner

ansprechpartner

Objekt vom Typ Person, oder null

Typ: Objekt, kein Pflichtfeld, Nullbar: ja, Defaultwert: null

branche

Objekt vom Typ Branche

Typ: Objekt, Pflichtfeld bei: POST und PUT (min. die id), Nullbar: nein, Defaultwert: keiner

4.17 Details zu Forumsbenutzer

4.17.1 HTTP Requests für Forumsbenutzer

Methode	Request
GET	<code>.../api/forumsbenutzer/gib/[id]</code> Liefert ein JSON Objekt vom Typ Forumsbenutzer für die angegebene [id]
GET	<code>.../api/forumsbenutzer/alle</code> Liefert alle JSON Objekte vom Typ Forumsbenutzer
GET	<code>../api/forumsbenutzer/existiert/[id]</code> Prüft nach, ob ein Objekt vom Typ Forumsbenutzer unter dieser [id] existiert Liefert true oder false zurück
GET	<code>.../api/forumsbenutzer/eindeutig</code> Prüft nach ob ein gelieferter Benutzername eindeutig ist, heißt nicht bereits in der Datenbank gespeichert ist. Benötigt den zu prüfenden Benutzernamen als Objekt Forumsbenutzereindeutig Liefert nach der Prüfung true zurück, falls der Benutzernamen noch nicht verwendet wird, oder aber false
POST	<code>../api/forumsbenutzer</code> Fügt die als JSON gesendeten Daten als neues Objekt hinzu Liefert das neu erstellte JSON Objekt vom Typ Forumsbenutzer zurück
PUT	<code>../api/forumsbenutzer</code> Ändert einen bestehenden JSON Objekt anhand der gesendeten Daten Liefert das geänderte JSON Objekt vom Typ Forumsbenutzer zurück
DELETE	<code>../api/forumsbenutzer/[id]</code> Löscht das durch die [id] spezifizierte JSON Objekt vom Typ Forumsbenutzer und liefert true zurück

4.17.2 Objekt vom Typ Forumsbenutzer

Beispiel

```
{
  "id": 3,
  "benutzername": "master@blaster.de",
  "geschlecht": "weiblich",
  "geburtstag": "17.11.1987",
  "beitritt": "10.09.2019 07:05:44",
  "rolle": {
    // Objekt vom Typ Benutzerrolle
  }
}
```

Abhängigkeiten

Objekt vom Typ Benutzerrolle

Attribute**id**

Eindeutiger Primärschlüssel des Objektes

Typ: Integer, Pflichtfeld bei: PUT, Nullbar: nein, Defaultwert: keiner

benutzername

Name des Users, muss eindeutig sein, darf also nicht doppelt in der Datenbank vorkommen

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

geschlecht

Gibt das Geschlecht an, entweder „weiblich“ oder „männlich“

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

geburtstag

Das Geburtsdatum des Benutzers im Format tt.mm.jjjj

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

beitritt

Der Zeitpunkt im Format tt.mm.jjjj hh:mm:ss, an welchem der Benutzereintrag erstellt wurde

Wird kein Zeitstempel geliefert, wird automatisch der aktuelle Systemzeitpunkt verwendet

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: Systemzeit

rolle

Objekt vom Typ Benutzerrolle

Typ: Objekt, Pflichtfeld bei: POST und PUT (min. die id), Nullbar: nein, Defaultwert: keiner

4.17.3 Abfrageobjekt vom Typ Forumsbenutzereindeutig

Beispiel

```
{  
    "benutzername": "master@blaster.de"  
}
```

4.18 Details zu Forumsbereich

4.18.1 HTTP Requests für Forumsbereich

Methode	Request
GET	<code>.../api/forumsbereich/gib/[id]</code> Liefert ein JSON Objekt vom Typ Forumsbereich für die angegebene [id]
GET	<code>.../api/forumsbereich/alle</code> Liefert alle JSON Objekte vom Typ Forumsbereich
GET	<code>../api/forumsbereich/existiert/[id]</code> Prüft nach, ob ein Objekt vom Typ Forumsbereich unter dieser [id] existiert Liefert true oder false zurück
POST	<code>../api/forumsbereich</code> Fügt die als JSON gesendeten Daten als neues Objekt hinzu Liefert das neu erstellte JSON Objekt vom Typ Forumsbereich zurück
PUT	<u>ACHTUNG: FORUMSEINTRÄGE WERDEN HIER NICHT GESPEICHERT</u> <code>../api/forumsbereich</code> Ändert einen bestehenden JSON Objekt anhand der gesendeten Daten Liefert das geänderte JSON Objekt vom Typ Forumsbereich zurück
DELETE	<u>ACHTUNG: FORUMSEINTRÄGE WERDEN HIER NICHT GESPEICHERT</u> <code>../api/forumsbereich/[id]</code> Löscht das durch die [id] spezifizierte JSON Objekt vom Typ Forumsbereich und liefert true zurück

4.18.2 Objekt vom Typ Forumsbereich

Beispiel

```
{
  "id": 22,
  "thema": "Alles rund um Bildschirme",
  "beschreibung": "Hier finden Sie alle Details",
  "administrator": {
    // Objekt vom Typ Forumsbenutzer
  },
  "threads": [
    { // Objekt vom Typ Forumseintrag },
    { // Objekt vom Typ Forumseintrag }, ...
  ]
}
```

Abhängigkeiten

Objekt vom Typ Forumsbenutzer
 Objekte vom Typ Forumseintrag

Attribute**id**

Eindeutiger Primärschlüssel des Objektes

Typ: Integer, Pflichtfeld bei: PUT, Nullbar: nein, Defaultwert: keiner

thema

Das Thema, um welches in dem Forumszweig geht

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

beschreibung

Weiterführende Beschreibung des Themas

Typ: Text, kein Pflichtfeld, Nullbar: nein, Defaultwert: leerer String

administrator

Objekt vom Typ Forumsbenutzer

Typ: Objekt, Pflichtfeld bei: POST und PUT (min. die id), Nullbar: nein, Defaultwert: keiner

threads

Array, leer, bzw. mit einem oder mehreren Objekten vom Typ Forumseintrag

Und dies rekursiv

Typ: Array mit Objekten, kein Pflichtfeld, Nullbar: nein, Defaultwert: leeres Array

Diese Einträge werden beim Service nur lesend zugegriffen.

Das Hinzufügen, Ändern oder Löschen übernimmt der Service Forumseintrag

4.19 Details zu Forumseintrag

4.19.1 HTTP Requests für Forumseintrag

Methode	Request
POST	<i>../api/forumseintrag</i> Fügt die als JSON gesendeten Daten als neues Objekt hinzu Liefert das neu erstellte JSON Objekt vom Typ Forumseintrag zurück
PUT	<i>../api/forumseintrag</i> Ändert einen bestehenden JSON Objekt anhand der gesendeten Daten Liefert das geänderte JSON Objekt vom Typ Forumseintrag zurück
DELETE	<i>../api/forumseintrag/[id]</i> Löscht das durch die [id] spezifizierte JSON Objekt vom Typ Forumseintrag und liefert true zurück

HINWEIS: DIESE SERVICES SIND IN DER DATEI „FORUMSBEREICH.JS“ ENTHALTEN

4.19.2 Objekt vom Typ Forumseintrag

Beispiel

```
{
  "id": 17,
  "beitrag": "Ich hab mal ne Frage",
  "ersteller": {
    // Objekt vom Typ Forumsbenutzer
  },
  "erstellzeitpunkt": "15.10.2019 22:10:15",
  "vater": {
    // Teilobjekt vom Typ Forumseintrag, enthält nur die Eigenschaft id
  },
  "antworten": [
    { // Objekt vom Typ Forumseintrag },
    { // Objekt vom Typ Forumseintrag }, ...
  ]
}
```

Abhängigkeiten

Objekt vom Typ Forumsbenutzer
Objekt vom Typ Forumseintrag

Attribute**id**

Eindeutiger Primärschlüssel des Objektes

Typ: Integer, Pflichtfeld bei: PUT, Nullbar: nein, Defaultwert: keiner

beitrag

Der Text, welcher den Eintrag darstellt

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

erstellzeitpunkt

Der Erstellzeitpunkt im Format tt.mm.jjjj hh:mm:ss, an welchem der Eintrag erstellt wurde

Wird kein Zeitstempel geliefert, wird automatisch der aktuelle Systemzeitpunkt verwendet

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: Systemzeit

vater

Teilobjekt vom Typ Forumseintrag. Benötigt nur die Eigenschaft id zwecks der Referenzierung

Damit wird definiert, wer das Vaterelement dieses Eintrages ist. Kann auch null sein bei keinem Vater

Typ: Objekt, Pflichtfeld bei: POST und PUT (min. die id), Nullbar: ja, Defaultwert: null

antworten

Array, entweder leer bzw. eines oder mehrere Objekte vom Typ Forumseintrag

Typ: Array mit Objekten, kein Pflichtfeld, Nullbar: nein, Defaultwert: leeres Array

HINWEIS: WENN EIN VATER NICHT ANGEGEBEN IST, HEIßT DIES, DASS DER EINTRAG EINEM BEREICH ZUGEORDNET WIRD. WENN EIN VATER ANGEGEBEN WIRD DANN GEHÖRT DER EINTRAG ZU EINEM ANDEREN EINTRAG.

LÖSCHEN: BEIM LÖSCHEN IST ANZUMERKEN, DASS EIN WIRKLICHES LÖSCHEN IN DER DATENBANK NICHT DURCHGEFÜHRT WIRD, WEGEN DER REKURSIVEN PROBLEMATIK. STATT DESSEN WIRD EIN SCHLATER BEIM JEWELIGEN EINTRAG GESETZT, WELCHER ANGIBT DASS DIESER EINTRAG (UND DAMIT AUCH ALLE DARUNTER GEHÖRIGE) NICHT MEHR GELADEN WERDEN.

4.20 Details zu Gericht

4.20.1 HTTP Requests für Gericht

Methode	Request
GET	<code>.../api/gericht/gib/[id]</code> Liefert ein JSON Objekt vom Typ Gericht für die angegebene [id]
GET	<code>.../api/gericht/alle</code> Liefert alle JSON Objekte vom Typ Gericht
GET	<code>../api/gericht/existiert/[id]</code> Prüft nach, ob ein Objekt vom Typ Gericht unter dieser [id] existiert Liefert true oder false zurück
POST	<code>../api/gericht</code> Fügt die als JSON gesendeten Daten als neues Objekt hinzu Liefert das neu erstellte JSON Objekt vom Typ Gericht zurück
PUT	<code>../api/gericht</code> Ändert einen bestehenden JSON Objekt anhand der gesendeten Daten Liefert das geänderte JSON Objekt vom Typ Gericht zurück
DELETE	<code>../api/gericht/[id]</code> Löscht das durch die [id] spezifizierte JSON Objekt vom Typ Gericht und liefert true zurück

4.20.2 Objekt vom Typ Gericht

Beispiel

```
{
  "id": 5,
  "bezeichnung": "Kirschtorte",
  "speisenart": {
    // Objekt vom Typ Speisenart
  },
  "zubereitung": "Man nehme dies und das...",
  "bildpfad": null,
  "zutaten": [
    { // Objekt vom Typ Zutatenliste },
    { // Objekt vom Typ Zutatenliste }, ...
  ],
  "bewertungen": [
    { // Objekt vom Typ Bewertung },
    { // Objekt vom Typ Bewertung }, ...
  ],
  "durchschnittsbewertung": 7.5
}
```

Abhängigkeiten

Objekt vom Typ Speisenart
Objekt vom Typ Bewertung
Objekt vom Typ Zutatenliste

Attribute

id

Eindeutiger Primärschlüssel des Objektes

Typ: Integer, Pflichtfeld bei: PUT, Nullbar: nein, Defaultwert: keiner

bezeichnung

Name des Gerichts

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

speisenart

Objekt vom Typ Speisenart

Typ: Objekt, Pflichtfeld bei: POST und PUT (min. die id), Nullbar: nein, Defaultwert: keiner

zubereitung

Zubereitungsanweisung

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

bildpfad

Angabe des Pfades / Dateiname zu einem Bild

Typ: Text, kein Pflichtfeld, Nullbar: ja, Defaultwert: null

zutaten

Array, eines oder mehrere Objekte vom Typ Zutatenliste. Darf nicht leer sein

Typ: Array mit Objekten, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

bewertungen

Array, entweder leer bzw. eines oder mehrere Objekte vom Typ Bewertung

Typ: Array mit Objekten, kein Pflichtfeld, Nullbar: nein, Defaultwert: leeres Array

durchschnittsbewertung

Der Durchschnitt, welcher aus den einzelnen Punkten der Bewertungen ermittelt wird.

Sollten keine Bewertungen gegeben sein ist der Durchschnitt 0

Wird NICHT in der Datenbank gehalten

Typ: Real, nur Anzeige

4.21 Details zu Kinosaal

4.21.1 HTTP Requests für Kinosaal

Methode	Request
GET	<i>.../api/kinosaal/gib/[id]</i> Liefert ein JSON Objekt vom Typ Kinosaal für die angegebene [id]
GET	<i>.../api/kinosaal/alle</i> Liefert alle JSON Objekte vom Typ Kinosaal
GET	<i>../api/kinosaal/existiert/[id]</i> Prüft nach, ob ein Objekt vom Typ Kinosaal unter dieser [id] existiert Liefert true oder false zurück
POST	<i>../api/kinosaal</i> Fügt die als JSON gesendeten Daten als neues Objekt hinzu Liefert das neu erstellte JSON Objekt vom Typ Kinosaal zurück
PUT	<i>../api/kinosaal</i> Ändert einen bestehenden JSON Objekt anhand der gesendeten Daten Liefert das geänderte JSON Objekt vom Typ Kinosaal zurück
DELETE	<i>../api/kinosaal/[id]</i> Löscht das durch die [id] spezifizierte JSON Objekt vom Typ Kinosaal und liefert true zurück

4.21.2 Objekt vom Typ Kinosaal

Beispiel

```
{
  "id": 2,
  "bezeichnung": "Kleiner Saal",
  "leinwand": 120,
  "tonsystem": "Dolby DTS",
  "projektion": "Multiplex Beamer",
  "projektionsart": "2D",
  "geschoss": "EG",
  "sitzreihen": 15,
  "sitzeprereihe": 10,
  "sitzegesamt": 150
}
```

Abhängigkeiten

Keine

Attribute

id

Eindeutiger Primärschlüssel des Objektes

Typ: Integer, Pflichtfeld bei: PUT, Nullbar: nein, Defaultwert: keiner

bezeichnung

Name des Kinosaals

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

leinwand

Größe der Leinwand in Quadratmeter, muss größer 0 sein

Typ: Real, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

tonsystem

Bezeichnung des Tonsystems im Raum

Typ: Text, kein Pflichtfeld, Nullbar: nein, Defaultwert: leerer String

projektion

Bezeichnung der Projektoren im Raum

Typ: Text, kein Pflichtfeld, Nullbar: nein, Defaultwert: leerer String

projektionsart

Angabe ob der Raum 2D oder 3D Filme anzeigen kann. Andere Werte sind nicht erlaubt

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

geschoss

Angabe in welchem Stockwerk sich der Saal befindet

Typ: Text, kein Pflichtfeld, Nullbar: nein, Defaultwert: leerer String

sitzreihen

Anzahl der Reihen von Sitzen im Raum, muss größer 0 sein

Typ: Integer, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

sitzreproreihe

Anzahl der Sitze innerhalb einer Reihe, muss größer 0 sein

Typ: Integer, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

sitzegesamt

Anzahl der Sitze im Raum

Wird NICHT in der Datenbank gehalten

Typ: Integer, nur Anzeige

4.22 Details zu Land

4.22.1 HTTP Requests für Land

Methode	Request
GET	<code>.../api/land/gib/[id]</code> Liefert ein JSON Objekt vom Typ Land für die angegebene [id]
GET	<code>.../api/land/alle</code> Liefert alle JSON Objekte vom Typ Land
GET	<code>../api/land/existiert/[id]</code> Prüft nach, ob ein Objekt vom Typ Land unter dieser [id] existiert Liefert true oder false zurück
POST	<code>../api/land</code> Fügt die als JSON gesendeten Daten als neues Objekt hinzu Liefert das neu erstellte JSON Objekt vom Typ Land zurück
PUT	<code>../api/land</code> Ändert einen bestehenden JSON Objekt anhand der gesendeten Daten Liefert das geänderte JSON Objekt vom Typ Land zurück
DELETE	<code>../api/land/[id]</code> Löscht das durch die [id] spezifizierte JSON Objekt vom Typ Land und liefert true zurück

4.22.2 Objekt vom Typ Land

Beispiel

```
{
  "id": 3,
  "kennzeichnung": "DE",
  "bezeichnung": "Deutsch"
}
```

Abhängigkeiten

Keine

Attribute

id

Eindeutiger Primärschlüssel des Objektes

Typ: Integer, Pflichtfeld bei: PUT, Nullbar: nein, Defaultwert: keiner

kennzeichnung

Kürzel für das Land

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

bezeichnung

Name des Landes

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

4.23 Details zu Mehrwertsteuer

4.23.1 HTTP Requests für Mehrwertsteuer

Methode	Request
GET	<code>.../api/mehrwertsteuer/gib/[id]</code> Liefert ein JSON Objekt vom Typ Mehrwertsteuer für die angegebene [id]
GET	<code>.../api/mehrwertsteuer/alle</code> Liefert alle JSON Objekte vom Typ Mehrwertsteuer
GET	<code>../api/mehrwertsteuer/existiert/[id]</code> Prüft nach, ob ein Objekt vom Typ Mehrwertsteuer unter dieser [id] existiert Liefert true oder false zurück
POST	<code>../api/mehrwertsteuer</code> Fügt die als JSON gesendeten Daten als neues Objekt hinzu Liefert das neu erstellte JSON Objekt vom Typ Mehrwertsteuer zurück
PUT	<code>../api/mehrwertsteuer</code> Ändert einen bestehenden JSON Objekt anhand der gesendeten Daten Liefert das geänderte JSON Objekt vom Typ Mehrwertsteuer zurück
DELETE	<code>../api/mehrwertsteuer/[id]</code> Löscht das durch die [id] spezifizierte JSON Objekt vom Typ Mehrwertsteuer und liefert true zurück

4.23.2 Objekt vom Typ Mehrwertsteuer

Beispiel

```
{
  "id": 1,
  "bezeichnung": "Standardsteuer",
  "steuersatz": 19.0
}
```

Abhängigkeiten

Keine

Attribute

id

Eindeutiger Primärschlüssel des Objektes

Typ: Integer, Pflichtfeld bei: PUT, Nullbar: nein, Defaultwert: keiner

bezeichnung

Name der Mehrwertsteuer

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

steuersatz

Der Steuersatz als Dezimalzahl, muss größer 0 sein

Typ: Real, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

4.24 Details zu Person

4.24.1 HTTP Requests für Person

Methode	Request
GET	<code>.../api/person/gib/[id]</code> Liefert ein JSON Objekt vom Typ Person für die angegebene [id]
GET	<code>.../api/person/alle</code> Liefert alle JSON Objekte vom Typ Person
GET	<code>../api/person/existiert/[id]</code> Prüft nach, ob ein Objekt vom Typ Person unter dieser [id] existiert Liefert true oder false zurück
POST	<code>../api/person</code> Fügt die als JSON gesendeten Daten als neues Objekt hinzu Liefert das neu erstellte JSON Objekt vom Typ Person zurück
PUT	<code>../api/person</code> Ändert einen bestehenden JSON Objekt anhand der gesendeten Daten Liefert das geänderte JSON Objekt vom Typ Person zurück
DELETE	<code>../api/person/[id]</code> Löscht das durch die [id] spezifizierte JSON Objekt vom Typ Person und liefert true zurück

4.24.2 Objekt vom Typ Person

Beispiel

```
{
  "id": 3,
  "anrede": "Frau",
  "vorname": "Petra",
  "nachname": "Müller",
  "adresse": {
    // Objekt vom Typ Adresse
  },
  "telefonnummer": "012356840",
  "email": "petra.m@web.de",
  "geburtstag": "15.10.1980"
}
```

Abhängigkeiten

Objekt vom Typ Adresse

Attribute**id**

Eindeutiger Primärschlüssel des Objektes

Typ: Integer, Pflichtfeld bei: PUT, Nullbar: nein, Defaultwert: keiner

anrede

Angabe der Anrede, kann „Herr“ oder „Frau“ sein

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

vorname

Der Vorname der Person

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

nachname

Der Nachname der Person

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

adresse

Objekt vom Typ Adresse

Typ: Objekt, Pflichtfeld bei: POST und PUT (min. die id), Nullbar: nein, Defaultwert: keiner

telefonnummer

Die Telefonnummer der Person

Typ: Text, kein Pflichtfeld, Nullbar: nein, Defaultwert: leerer String

email

Die Email Adresse der Person

Muss das Email Format erfüllen

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

geburtstag

Der Geburtstag der Person

Entweder angegeben in deutschem Format tt.mm.jjjj oder null

Typ: Text, kein Pflichtfeld, Nullbar: ja, Defaultwert: keiner

4.25 Details zu Produkt

4.25.1 HTTP Requests für Produkt

Methode	Request
GET	<code>.../api/produkt/gib/[id]</code> Liefert ein JSON Objekt vom Typ Produkt für die angegebene [id]
GET	<code>.../api/produkt/alle</code> Liefert alle JSON Objekte vom Typ Produkt
GET	<code>../api/produkt/existiert/[id]</code> Prüft nach, ob ein Objekt vom Typ Produkt unter dieser [id] existiert Liefert true oder false zurück
POST	<code>../api/produkt</code> Fügt die als JSON gesendeten Daten als neues Objekt hinzu Liefert das neu erstellte JSON Objekt vom Typ Produkt zurück
PUT	<code>../api/produkt</code> Ändert einen bestehenden JSON Objekt anhand der gesendeten Daten Liefert das geänderte JSON Objekt vom Typ Produkt zurück
DELETE	<code>../api/produkt/[id]</code> Löscht das durch die [id] spezifizierte JSON Objekt vom Typ Produkt und liefert true zurück

4.25.2 Objekt vom Typ Produkt

Beispiel

```
{
  "id": 9,
  "kategorie": {
    // Objekt vom Typ Produktkategorie
  },
  "bezeichnung": "Produkt ABC",
  "beschreibung": "Eines unserer besten Produkte",
  "mehrwertsteuer": {
    // Objekt vom Typ Mehrwertsteuer
  },
  "details": "Weitere Infos und Beschreibungen",
  "nettopreis": 19.99,
  "mehrwertsteueranteil": 3.80,
  "bruttopreis": 23.79,
  "datenblatt": {
    // Objekt vom Typ Download, oder null
  },
  "bilder": [
    { // Objekt vom Typ Produktbild },
    { // Objekt vom Typ Produktbild }, ...
  ]
}
```


Abhängigkeiten

Objekt vom Typ Produktkategorie
Objekt vom Typ Produktbild (falls hinterlegt)
Objekt vom Typ Download (falls hinterlegt)
Objekt vom Typ Mehrwertsteuer

Attribute

id

Eindeutiger Primärschlüssel des Objektes

Typ: Integer, Pflichtfeld bei: PUT, Nullbar: nein, Defaultwert: keiner

kategorie

Objekt vom Typ Produktkategorie

Typ: Objekt, Pflichtfeld bei: POST und PUT (min. die id), Nullbar: nein, Defaultwert: keiner

bezeichnung

Name des Produktes

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

beschreibung

Beschreibung zum Produkt

Typ: Text, kein Pflichtfeld, Nullbar: nein, Defaultwert: leerer String

mehrwertsteuer

Objekt vom Typ Mehrwertsteuer

Typ: Objekt, Pflichtfeld bei: POST und PUT (min. die id), Nullbar: nein, Defaultwert: keiner

details

Weiterführende Beschreibung zum Produkt

Typ: Text, kein Pflichtfeld, Nullbar: ja, Defaultwert: null

nettopreis

Nettopreis des Produktes

Typ: Real, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

mehrwertsteueranteil

Anteil, welcher sich aus Nettopreis und Mehrwertsteuersatz ergibt

Wird NICHT in der Datenbank gehalten

Typ: Real, nur Anzeige

bruttopreis

Die Summe von Nettopreis und Mehrwertsteueranteil

Wird NICHT in der Datenbank gehalten

Typ: Real, nur Anzeige

datenblatt

Objekt vom Typ Download

Typ: Objekt, kein Pflichtfeld, Nullbar: ja, Defaultwert: null

bilder

Array, entweder leer bzw. eines oder mehrere Objekte vom Typ Produktbild

Typ: Array mit Objekten, kein Pflichtfeld, Nullbar: nein, Defaultwert: leeres Array

4.26 Details zu Produktbild

4.26.1 HTTP Requests für Produktbild

Methode	Request
GET	<code>.../api/produktbild/gib/[id]</code> Liefert ein JSON Objekt vom Typ Produktbild für die angegebene [id]
GET	<code>.../api/produktbild/alle</code> Liefert alle JSON Objekte vom Typ Produktbild
GET	<code>../api/produktbild/existiert/[id]</code> Prüft nach, ob ein Objekt vom Typ Produktbild unter dieser [id] existiert Liefert true oder false zurück
POST	<code>../api/produktbild</code> Fügt die als JSON gesendeten Daten als neues Objekt hinzu Liefert das neu erstellte JSON Objekt vom Typ Produktbild zurück
PUT	<code>../api/produktbild</code> Ändert einen bestehenden JSON Objekt anhand der gesendeten Daten Liefert das geänderte JSON Objekt vom Typ Produktbild zurück
DELETE	<code>../api/produktbild/[id]</code> Löscht das durch die [id] spezifizierte JSON Objekt vom Typ Produktbild und liefert true zurück

4.26.2 Objekt vom Typ Produktbild

Beispiel

```
{
  "id": 4,
  "bezeichnung": "Hosen"
}
```

Abhängigkeiten

Keine

Attribute

id

Eindeutiger Primärschlüssel des Objektes

Typ: Integer, Pflichtfeld bei: PUT, Nullbar: nein, Defaultwert: keiner

bildpfad

Pfad zur Datei auf dem Dateisystem

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

4.27 Details zu Produktkategorie

4.27.1 HTTP Requests für Produktkategorie

Methode	Request
GET	<code>.../api/produktkategorie/gib/[id]</code> Liefert ein JSON Objekt vom Typ Produktkategorie für die angegebene [id]
GET	<code>.../api/produktkategorie/alle</code> Liefert alle JSON Objekte vom Typ Produktkategorie
GET	<code>../api/produktkategorie/existiert/[id]</code> Prüft nach, ob ein Objekt vom Typ Produktkategorie unter dieser [id] existiert Liefert true oder false zurück
POST	<code>../api/produktkategorie</code> Fügt die als JSON gesendeten Daten als neues Objekt hinzu Liefert das neu erstellte JSON Objekt vom Typ Produktkategorie zurück
PUT	<code>../api/produktkategorie</code> Ändert einen bestehenden JSON Objekt anhand der gesendeten Daten Liefert das geänderte JSON Objekt vom Typ Produktkategorie zurück
DELETE	<code>../api/produktkategorie/[id]</code> Löscht das durch die [id] spezifizierte JSON Objekt vom Typ Produktkategorie und liefert true zurück

4.27.2 Objekt vom Typ Produktkategorie

Beispiel

```
{
  "id": 4,
  "bezeichnung": "Hosen"
}
```

Abhängigkeiten

Keine

Attribute

id

Eindeutiger Primärschlüssel des Objektes

Typ: Integer, Pflichtfeld bei: PUT, Nullbar: nein, Defaultwert: keiner

bezeichnung

Name der Produktkategorie

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

4.28 Details zu Reservierer

4.28.1 HTTP Requests für Reservierer

Methode	Request
GET	<i>.../api/reservierer/gib/[id]</i> Liefert ein JSON Objekt vom Typ Reservierer für die angegebene [id]
GET	<i>.../api/reservierer/alle</i> Liefert alle JSON Objekte vom Typ Reservierer
GET	<i>../api/eineit/existiert/[id]</i> Prüft nach, ob ein Objekt vom Typ Reservierer unter dieser [id] existiert Liefert true oder false zurück
POST	<i>../api/reservierer</i> Fügt die als JSON gesendeten Daten als neues Objekt hinzu Liefert das neu erstellte JSON Objekt vom Typ Reservierer zurück
PUT	<i>../api/reservierer</i> Ändert einen bestehenden JSON Objekt anhand der gesendeten Daten Liefert das geänderte JSON Objekt vom Typ Reservierer zurück
DELETE	<i>../api/reservierer/[id]</i> Löscht das durch die [id] spezifizierte JSON Objekt vom Typ Reservierer und liefert true zurück

4.28.2 Objekt vom Typ Reservierer

Beispiel

```
{
  "id": 5,
  "vorname": "Michaela",
  "nachname": "Müller",
  "email": "m.m@hotmail.com"
}
```

Abhängigkeiten

Keine

Attribute

id

Eindeutiger Primärschlüssel des Objektes

Typ: Integer, Pflichtfeld bei: PUT, Nullbar: nein, Defaultwert: keiner

vorname

Vorname der Person

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

nachname

Nachname der Person

Typ: Text, kein Pflichtfeld, Nullbar: nein, Defaultwert: leerer String

email

Die Email Adresse der Person, muss das Email Format erfüllen

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

4.29 Details zu ReservierterSitz

4.29.1 HTTP Requests für ReservierterSitz

Zu ReservierterSitz gibt es keinen eigenen Service. Objekte diesen Typs sind Teil von Reservierung.

4.29.2 Objekt vom Typ ReservierterSitz

Beispiel

```
{
  "id": 33,
  "reservierung": {
    // Teilobjekt vom Typ Reservierung, enthält nur die Eigenschaft id
  },
  "reihe": 5,
  "sitzplatz": 10
}
```

Abhängigkeiten

Teilobjekt vom Typ Reservierung, nur die ID, zwecks Referenzierung

Attribute

id

Eindeutiger Primärschlüssel des Objektes

Typ: Integer, Pflichtfeld bei: PUT, Nullbar: nein, Defaultwert: keiner

reservierung

Teilobjekt vom Typ Reservierung. Benötigt nur die Eigenschaft id zwecks der Referenzierung

Typ: Objekt, Pflichtfeld bei: POST und PUT (min. die id), Nullbar: nein, Defaultwert: keiner

reihe

Nummer der Sitzreihe, muss eine Zahl größer 0 sein

Typ: Integer, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: 1

sitzplatz

Nummer des Sitzes in der Reihe, muss eine Zahl größer 0 sein

Typ: Integer, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: 1

4.30 Details zu Reservierung

4.30.1 HTTP Requests für Reservierung

Methode	Request
GET	<code>.../api/reservierung/gib/[id]</code> Liefert ein JSON Objekt vom Typ Reservierung für die angegebene [id]
GET	<code>.../api/reservierung/alle</code> Liefert alle JSON Objekte vom Typ Reservierung
GET	<code>../api/eineit/existiert/[id]</code> Prüft nach, ob ein Objekt vom Typ Reservierung unter dieser [id] existiert Liefert true oder false zurück
POST	<code>../api/reservierung</code> Fügt die als JSON gesendeten Daten als neues Objekt hinzu Liefert das neu erstellte JSON Objekt vom Typ Reservierung zurück
PUT	<code>../api/reservierung</code> Ändert einen bestehenden JSON Objekt anhand der gesendeten Daten Liefert das geänderte JSON Objekt vom Typ Reservierung zurück
DELETE	<code>../api/reservierung/[id]</code> Löscht das durch die [id] spezifizierte JSON Objekt vom Typ Reservierung und liefert true zurück

4.30.2 Objekt vom Typ Reservierung

Beispiel

```
{
  "id": 2,
  "zeitpunkt": "15.10.2019 12:12:13",
  "vorstellung": {
    // Teilobjekt vom Typ Vorstellung, enthält nur die Eigenschaft id
  },
  "reservierer": {
    // Objekt vom Typ Reservierer
  },
  "zahlungsart": {
    // Objekt vom Typ Zahlungsart
  },
  "reserviertesitze": [
    { // Objekt vom Typ ReservierterSitz },
    { // Objekt vom Typ ReservierterSitz }, ...
  ]
}
```

Abhängigkeiten

Objekt vom Typ Reservierer

Objekt vom Typ Zahlungsart

Teilobjekt vom Typ Vorstellung, nur die ID, zwecks Referenzierung

Objekt vom Typ ReservierterSitz

Attributeeste

id

Eindeutiger Primärschlüssel des Objektes

Typ: Integer, Pflichtfeld bei: PUT, Nullbar: nein, Defaultwert: keiner

zeitpunkt

Der Zeitpunkt im Format tt.mm.jjjj hh:mm:ss, an welchem die Reservierung ausgelöst wurde

Wird kein Zeitstempel geliefert, wird automatisch der aktuelle Systemzeitpunkt verwendet

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: Systemzeit

vorstellung

Teilobjekt vom Typ Vorstellung. Benötigt nur die Eigenschaft id zwecks der Referenzierung

Typ: Objekt, Pflichtfeld bei: POST und PUT (min. die id), Nullbar: nein, Defaultwert: keiner

reservierer

Objekt vom Typ Reservierer

Typ: Objekt, Pflichtfeld bei: POST und PUT (min. die id), Nullbar: nein, Defaultwert: keiner

zahlungsart

Objekt vom Typ Zahlungsart

Typ: Objekt, Pflichtfeld bei: POST und PUT (min. die id), Nullbar: nein, Defaultwert: keiner

reserviertesitze

Array, mit einem oder mehreren Objekten vom Typ ReservierterSitz

Typ: Array mit Objekten, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

4.31 Details zu Speisenart

4.31.1 HTTP Requests für Speisenart

Methode	Request
GET	<code>.../api/speisenart/gib/[id]</code> Liefert ein JSON Objekt vom Typ Speisenart für die angegebene [id]
GET	<code>.../api/speisenart/alle</code> Liefert alle JSON Objekte vom Typ Speisenart
GET	<code>../api/eineit/existiert/[id]</code> Prüft nach, ob ein Objekt vom Typ Speisenart unter dieser [id] existiert Liefert true oder false zurück
POST	<code>../api/speisenart</code> Fügt die als JSON gesendeten Daten als neues Objekt hinzu Liefert das neu erstellte JSON Objekt vom Typ Speisenart zurück
PUT	<code>../api/speisenart</code> Ändert einen bestehenden JSON Objekt anhand der gesendeten Daten Liefert das geänderte JSON Objekt vom Typ Speisenart zurück
DELETE	<code>../api/speisenart/[id]</code> Löscht das durch die [id] spezifizierte JSON Objekt vom Typ Speisenart und liefert true zurück

4.31.2 Objekt vom Typ Speisenart

Beispiel

```
{
  "id": 1,
  "bezeichnung": "Gerichte aus dem Ofen",
  "beschreibung": "Das ist lecker",
  "bildpfad": "ofen.jpg"
}
```

Abhängigkeiten

Keine

Attribute

id

Eindeutiger Primärschlüssel des Objektes

Typ: Integer, Pflichtfeld bei: PUT, Nullbar: nein, Defaultwert: keiner

bezeichnung

Bezeichnung der Speisenart

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

beschreibung

Weiterführende Beschreibungen zur Speisenart

Typ: Text, kein Pflichtfeld, Nullbar: nein, Defaultwert: leerer String

bildpfad

Pfad zur Bilddatei

Typ: Text, kein Pflichtfeld, Nullbar: ja, Defaultwert: null

4.32 Details zu Termin

4.32.1 HTTP Requests für Termin

Methode	Request
GET	<code>.../api/termin/gib/[id]</code> Liefert ein JSON Objekt vom Typ Termin für die angegebene [id]
GET	<code>.../api/termin/alle</code> Liefert alle JSON Objekte vom Typ Termin
GET	<code>../api/termin/existiert/[id]</code> Prüft nach, ob ein Objekt vom Typ Termin unter dieser [id] existiert Liefert true oder false zurück
GET	<code>.../api/termin/dienstleister/[id]</code> Liefert alle JSON Objekte vom Typ Termin, welche zu einem bestimmten Dienstleister gehören, definiert durch die angegebene ID
GET	<code>.../api/termin/bereich</code> Liefert alle JSON Objekte vom Typ Termin, zu einem bestimmten zeitlichen Bereich gehören. Details siehe ObjektTerminbereich
POST	<code>../api/termin</code> Fügt die als JSON gesendeten Daten als neues Objekt hinzu Liefert das neu erstellte JSON Objekt vom Typ Termin zurück
PUT	<code>../api/termin</code> Ändert einen bestehenden JSON Objekt anhand der gesendeten Daten Liefert das geänderte JSON Objekt vom Typ Termin zurück
DELETE	<code>../api/termin/[id]</code> Löscht das durch die [id] spezifizierte JSON Objekt vom Typ Termin und liefert true zurück

4.32.2 Objekt vom Typ Termin

Beispiel

```
{
  "id": 1,
  "bezeichnung": "Unterricht",
  "beschreibung": "Ist wichtig",
  "zeitpunkt": "15.10.2019 09:45:00",
  "dauer": 90,
  "dienstleister": {
    // Objekt vom Typ Firma, oder null
  }
}
```

Abhängigkeiten

Optional vom Typ Firma

Attribute**id**

Eindeutiger Primärschlüssel des Objektes

Typ: Integer, Pflichtfeld bei: PUT, Nullbar: nein, Defaultwert: keiner

bezeichnung

Der Name des Termins

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

beschreibung

Ein beliebiger Text

Typ: Text, kein Pflichtfeld, Nullbar: nein, Defaultwert: leerer String

zeitpunkt

Der Zeitpunkt im Format tt.mm.jjjj hh:mm:ss, an welchem der Termin statt findet

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

dauer

Die Dauer des Termins in Minuten

Typ: Integer, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: 60

dienstleister

Objekt vom Typ Firma, oder null

Typ: Objekt, kein Pflichtfeld, Nullbar: ja, Defaultwert: null

4.32.3 Abfrageobjekt vom Typ Terminbereich

Beispiel

```
{  
  "id": 1,  
  "von": "10.10.2019 00:00:00",  
  "bis": "12.10.2019 12:30:00"  
}
```

4.33 Details zu Vorstellung

4.33.1 HTTP Requests für Vorstellung

Methode	Request
GET	<i>.../api/vorstellung/gib/[id]</i> Liefert ein JSON Objekt vom Typ Vorstellung für die angegebene [id]
GET	<i>.../api/vorstellung/alle</i> Liefert alle JSON Objekte vom Typ Vorstellung
GET	<i>../api/vorstellung/existiert/[id]</i> Prüft nach, ob ein Objekt vom Typ Vorstellung unter dieser [id] existiert Liefert true oder false zurück
GET	<i>.../api/vorstellung/film/[id]</i> Liefert alle JSON Objekte vom Typ Vorstellung zu einem bestimmten Film, identifiziert durch die [id]
GET	<i>.../api/vorstellung/kinosaal/[id]</i> Liefert alle JSON Objekte vom Typ Vorstellung zu einem bestimmten Kinosaal, identifiziert durch die [id]
GET	<i>.../api/vorstellung/sitzfrei</i> Prüft nach, ob ein bestimmter Sitz bei einer Vorstellung frei ist (returniert true) oder nicht (false). Benötigt ein Abfrageobjekt vom Typ Sitzfrei
POST	<i>../api/vorstellung</i> Fügt die als JSON gesendeten Daten als neues Objekt hinzu Liefert das neu erstellte JSON Objekt vom Typ Vorstellung zurück
PUT	<i>../api/vorstellung</i> Ändert einen bestehenden JSON Objekt anhand der gesendeten Daten Liefert das geänderte JSON Objekt vom Typ Vorstellung zurück
DELETE	<i>../api/vorstellung/[id]</i> Löscht das durch die [id] spezifizierte JSON Objekt vom Typ Vorstellung und liefert true zurück

4.33.2 Objekt vom Typ Vorstellung

Beispiel

```
{
  "id": 33,
  "film": {
    // Objekt vom Typ Film
  },
  "kinosaal": {
    // Objekt vom Typ Kinosaal
  },
  "zeitpunkt": "15.10.2019 09:45:00",
  "reservierungen": [
    { // Objekt vom Typ Reservierung },
    { // Objekt vom Typ Reservierung }, ...
  ],
  "sitze": {
    "gesamt": 120,
    "reserviert": 15,
    "frei": 105
  }
}
```

Abhängigkeiten

Objekt vom Typ Film

Objekt vom Typ Kinosaal

Objekt/e vom Typ Reservierung (falls hinterlegt)

Attribute

id

Eindeutiger Primärschlüssel des Objektes

Typ: Integer, Pflichtfeld bei: PUT, Nullbar: nein, Defaultwert: keiner

film

Objekt vom Typ Film, zu welchem die Vorstellung ist

Typ: Objekt, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

kinosaal

Objekt vom Typ Kinosaal, in welchem die Vorstellung stattfindet

Typ: Objekt, Pflichtfeld bei: POST und PUT (min. die id), Nullbar: nein, Defaultwert: keiner

zeitpunkt

Der Zeitpunkt im Format tt.mm.jjjj hh:mm:ss, zu welchem die Vorstellung stattfindet

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: Systemzeit

reservierungen

Array, welches leer ist bzw. mit einem oder mehreren Objekten vom Typ Reservierung

Typ: Array mit Objekten, kein Pflichtfeld, Nullbar: nein, Defaultwert: leeres Array

sitze

Informatives Unterobjekt, welches die Anzahl der Sitze, die reservierten und freien Sitze angibt

Wird NICHT in der Datenbank gehalten

Typ: Unterobjekt, nur Anzeige

4.33.3 Abfrageobjekt vom Typ Sitzfrei

Beispiel

```
{  
    "id": 1,  
    "reihe": 5,  
    "sitzplatz": 7  
}
```

4.34 Details zu Zahlungsart

4.34.1 HTTP Requests für Zahlungsart

Methode	Request
GET	<code>.../api/zahlungsart/gib/[id]</code> Liefert ein JSON Objekt vom Typ Zahlungsart für die angegebene [id]
GET	<code>.../api/zahlungsart/alle</code> Liefert alle JSON Objekte vom Typ Zahlungsart
GET	<code>../api/zahlungsart/existiert/[id]</code> Prüft nach, ob ein Objekt vom Typ Zahlungsart unter dieser [id] existiert Liefert true oder false zurück
POST	<code>../api/zahlungsart</code> Fügt die als JSON gesendeten Daten als neues Objekt hinzu Liefert das neu erstellte JSON Objekt vom Typ Zahlungsart zurück
PUT	<code>../api/zahlungsart</code> Ändert einen bestehenden JSON Objekt anhand der gesendeten Daten Liefert das geänderte JSON Objekt vom Typ Zahlungsart zurück
DELETE	<code>../api/zahlungsart/[id]</code> Löscht das durch die [id] spezifizierte JSON Objekt vom Typ Zahlungsart und liefert true zurück

4.34.2 Objekt vom Typ Zahlungsart

Beispiel

```
{
  "id": 2,
  "bezeichnung": "PayPal"
}
```

Abhängigkeiten

Keine

Attribute

id

Eindeutiger Primärschlüssel des Objektes

Typ: Integer, Pflichtfeld bei: PUT, Nullbar: nein, Defaultwert: keiner

bezeichnung

Name der Zahlungsart

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

4.35 Details zu Zutat

4.35.1 HTTP Requests für Zutat

Methode	Request
GET	<code>.../api/zutat/gib/[id]</code> Liefert ein JSON Objekt vom Typ Zutat für die angegebene [id]
GET	<code>.../api/zutat/alle</code> Liefert alle JSON Objekte vom Typ Zutat
GET	<code>../api/eineit/existiert/[id]</code> Prüft nach, ob ein Objekt vom Typ Zutat unter dieser [id] existiert Liefert true oder false zurück
POST	<code>../api/zutat</code> Fügt die als JSON gesendeten Daten als neues Objekt hinzu Liefert das neu erstellte JSON Objekt vom Typ Zutat zurück
PUT	<code>../api/zutat</code> Ändert einen bestehenden JSON Objekt anhand der gesendeten Daten Liefert das geänderte JSON Objekt vom Typ Zutat zurück
DELETE	<code>../api/zutat/[id]</code> Löscht das durch die [id] spezifizierte JSON Objekt vom Typ Zutat und liefert true zurück

4.35.2 Objekt vom Typ Zutat

Beispiel

```
{
  "id": 5,
  "bezeichnung": "Milch",
  "beschreibung": "Die Gute aus dem Allgäu"
}
```

Abhängigkeiten

Keine

Attribute

id

Eindeutiger Primärschlüssel des Objektes

Typ: Integer, Pflichtfeld bei: PUT, Nullbar: nein, Defaultwert: keiner

bezeichnung

Bezeichnung der Zutat

Typ: Text, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: keiner

beschreibung

Weiterführende Beschreibungen zur Zutat

Typ: Text, kein Pflichtfeld, Nullbar: nein, Defaultwert: leerer String

4.36 Details zu Zutatenliste

4.36.1 HTTP Requests für Zutatenliste

Zu Zutatenliste gibt es keinen eigenen Service. Objekte diesen Typs sind Teil von Gericht.

4.36.2 Objekt vom Typ Zutatenliste

Beispiel

```
{
  "id": 3,
  "gericht": {
    // Teilobjekt vom Typ Gericht, enthält nur die Eigenschaft id
  },
  "zutat": {
    // Objekt vom Typ Zutat
  },
  "menge": 5,
  "einheit": {
    // Objekt vom Typ Einheit
  }
}
```

Abhängigkeiten

Teilobjekt vom Typ Gericht, nur die ID, zwecks Referenzierung

Objekt vom Typ Zutat

Objekt vom Typ Einheit

Attribute

id

Eindeutiger Primärschlüssel des Objektes

Typ: Integer, Pflichtfeld bei: PUT, Nullbar: nein, Defaultwert: keiner

gericht

Teilobjekt vom Typ Gericht. Benötigt nur die Eigenschaft id zwecks der Referenzierung

Typ: Objekt, Pflichtfeld bei: POST und PUT (min. die id), Nullbar: nein, Defaultwert: keiner

zutat

Objekt vom Typ Zutat

Typ: Objekt, Pflichtfeld bei: POST und PUT (min. die id), Nullbar: nein, Defaultwert: keiner

menge

Angabe der Anzahl, muss größer 0 sein

Typ: Integer, Pflichtfeld bei: POST und PUT, Nullbar: nein, Defaultwert: 1

einheit

Objekt vom Typ Einheit

Typ: Objekt, Pflichtfeld bei: POST und PUT (min. die id), Nullbar: nein, Defaultwert: keiner

5 Dateiuploads

Neben dem reinen Transfer von Text können auch binäre Daten zum Server gesendet und dort verarbeitet werden, heißt also auch Dateien.

5.1 Node.js Middleware express-fileupload

Damit Uploads generell von einem Node.JS Express Webserver verarbeitet werden können, wird die Dependency „express-fileupload“ benötigt.

Dem Express Server muss mitgeteilt werden, dass er das Modul verwenden soll. Siehe Datei „server.js“, Zeilen 34 bis 39.

```
app.use(fileUpload({
  createParentPath: true,
  limits: {
    fileSize: 2 * 1024 * 1024 * 1024    // limit to 2MB
  }
}));
```

Hier kann dann auch definiert werden, wie groß die Dateien maximal sein dürfen

5.2 HTML Formulare für Dateiuploads

Ein normales HTML Formular ist nicht in der Lage, Binärdaten zu übertragen. Daher ist es wichtig den Formtag so zu erweitern, dass die Dateien richtig geladen und übertragen werden. Hier ein Beispiel über die benötigten Angaben:

```
<form id="uploadFormlar" enctype="multipart/form-data" method="post">
```

Zusätzlich wird dann noch ein Datei - HTML Formularfeld benötigt, wie z.B.

```
<input type="file" name="picture">
```

Wenn mehrere Dateien gleichzeitig aufgeladen werden sollen, kann noch das Attribut „multiple“ angehängt werden.

```
<input type="file" name="picture" multiple>
```

5.3 Zugriff auf Dateiuploads auf der Serverseite

Serverseitig werden Dateiuploads im „request“ – Objekt als Attribut „**files**“ zur Verfügung gestellt. Die Verbindung zur HTML Seite ist das dort verwendete Attribut „name“.

Wird also in HTML ... name="picture" ... angegeben, kann in Node.JS per

request.files.picture

auf die Dateiinformationen zugegriffen werden, welche als JSON Objekt zur Verfügung gestellt werden.

Wird HTML seitig das Attribut „multiple“ angegeben, ist unter diesem Zugriffsnamen gleich ein Array gegeben.

5.3.1 Aufbau eines original Datei – JSON Objektes

Die erzeugten JSON Objekte für Dateien haben folgenden Aufbau:

```
{
  name: 'info.txt',      // Dateiname mit Dateiendung
  data: <Buffer 68 74 74 70 73...>, // Hexdaten / der Inhalt der Datei selbst
  size: 59,              // Dateigröße in byte
  encoding: '7bit',      // encoding der Datei
  tempFilePath: "",      // temporärer Dateipfad
  truncated: false,      // wurde Datei nur teilweise hochgeladen
  mimetype: 'text/plain', // typ der Datei
  md5: '3cf22e21f62230adf603a7033db4cfd0', // md5 Checksumme der Datei
  mv: [Function: mv]    // Funktion zum bewegen der Datei
}
```

5.3.2 Hilfsbibliothek fileHelper.js

Die Hilfsbibliothek **fileHelper.js**, welche Serverseitig zur Verfügung steht soll Ihnen den Umgang mit Dateiuploads vereinfachen. In der Bibliothek finden Sie Funktionen um die Anzahl der aufgeladenen Dateien zu erfahren, die Datei JSON Objekte zu erhalten, diese auch zu erweitern mit weiteren Informationen oder aber z.B. zufallsgesteuerte Dateinamen zu erzeugen.

Die jeweiligen Funktionalitäten sind in der Bibliotheksdatei direkt per Kommentar beschrieben.

Wenn Sie den Parameter „expanded“ bei den Funktionen „getUploadedFilesAsArray()“ bzw. „getAllUplaodedFilesAsArray()“ auf true setzen, werden die original Datei – JSON Objekte um folgende Eigenschaften erweitert:

```
nameOnly: 'info',      // Dateiname ohne Dateiendung
extension: 'txt',      // die Dateiendung
isPicture: false,      // Flag ob die Datei ein Bild ist
isWebPicture: false,   // Flag ob die Datei ein Web Bild ist (jpg, png, gif)
```

5.4 Weitere Details zu Fileuploads mit express-fileupload

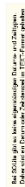
Weitere Details zu Fileuploads finden Sie auf der npm Seite

<https://www.npmjs.com/package/express-fileupload>

5.5 Dateien und Datenbanken

Wie bereits in Kapitel 2.1 erwähnt, ist es möglich Dateien in Datenbanken zu speichern. Hier werden neben dem Dateinamen, dem Mimetype und der Dateigröße die reinen Binärdaten in der Datenbank gespeichert, meist in einer BLOB Spalte. Mit diesen Informationen können dann später Dateien wiederhergestellt und zum Client zurück geliefert werden.

Im aktuellen Backend werden Dateien aber nur aufgeladen und nicht mehr verarbeitet. Heißt die Dateien gehen nach dem Upload verloren. Das Beispiel in der Demo soll nur veranschaulichen wie ein Upload generell funktionieren kann.



6.2 Beispiel eines SQL Create Table Statements

```
CREATE TABLE Land (
    ID INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
    Kennzeichnung TEXT NOT NULL,
    Bezeichnung TEXT NOT NULL
);
```

6.3 Beispiel eines SQL Insert Statements

```
INSERT INTO Kinosaal (Bezeichnung, Leinwand, Tonsystem, Projektion, Projektionsart,
Sitzreihen, SitzeProReihe, Geschoss) VALUES ('Saal 1', 67, 'Dolby SRD', 'Digitale
Projektion', 0, 10, 22, 'Erdgeschoss');
```

6.4 Beispiele zu Node.js Codes

6.4.1 Datenbankverbindung

```
const Database = require('better-sqlite3');
const dbOptions = { verbose: console.log };
const dbFile = './db/db.sqlite';
const dbConnection = new Database(dbFile, dbOptions);
```

6.4.2 Webserver erstellen und starten

```
// create Server
const HTTP_PORT = 8000;
var express = require('express');
var app = express();

// setup server for post data
const bodyParser = require("body-parser");
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
...
```

```
// start server
app.listen(HTTP_PORT, () => {
  console.log('Start Web Server...');
  console.log('Server running at localhost on port ' + HTTP_PORT);
});
```

6.4.3 Service Klassen / Handler in den Server einbinden

```
const TOPLEVELPATH = '/api';
var serviceRouter = require('./services/land.js');
app.use(TOPLEVELPATH, serviceRouter);

serviceRouter = require('./services/adresse.js');
app.use(TOPLEVELPATH, serviceRouter);

// ...
```

6.4.4 Service Methode innerhalb einer Service – Klasse

```
const helper = require('../helper.js');
const LandDao = require('../dao/landDao.js');
const express = require('express');
var serviceRouter = express.Router();

// ...

serviceRouter.get('/land/gib/:id', function(request, response) {
  helper.log('Service Land: Client requested one record, id=' + request.params.id);

  const landDao = new LandDao(request.app.locals.dbConnection);
  try {
    var result = landDao.loadById(request.params.id);
    helper.log('Service Land: Record loaded');
    response.status(200).json(helper.jsonMsgOK(result));
  } catch (ex) {
    helper.logError('Service Land: Error loading record by id. Exception occurred: ' +
ex.message);
    response.status(400).json(helper.jsonMsgError(ex.message));
  }
});

// ...

module.exports = serviceRouter;
```

6.4.5 Beispielhafte DAO Klasse

```

const helper = require('../helper.js');

class LandDao {

  constructor(dbConnection) {
    this._conn = dbConnection;
  }

  getConnection() {
    return this._conn;
  }

  loadById(id) {
    var sql = 'SELECT * FROM Land WHERE ID=?';
    var statement = this._conn.prepare(sql);
    var result = statement.get(id);

    if (helper.isUndefined(result))
      throw new Error('No Record found by id=' + id);

    return helper.objectKeysToLower(result);
  }

  // ...

  create(kennzeichnung = "", bezeichnung = "") {
    var sql = 'INSERT INTO Land (Kennzeichnung,Bezeichnung) VALUES (?,?)';
    var statement = this._conn.prepare(sql);
    var params = [kennzeichnung, bezeichnung];
    var result = statement.run(params);

    if (result.changes !== 1)
      throw new Error('Could not insert new Record. Data: ' + params);

    var newObj = this.loadById(result.lastInsertRowid);
    return newObj;
  }

  // ...

  toString() {
    helper.log('LandDao [_conn=' + this._conn + ']');
  }
}

module.exports = LandDao;

```

6.4.6 Postman

Als Tool, um bei Änderungen am Backend diese zu prüfen, heißt also unterschiedliche HTTP Requests an das Backend zu senden empfehlen wir Ihnen das Programm „Postman“.

<https://www.getpostman.com/>

Dieses ist kostenlos erhältlich, kann nach einer Registrierung ohne weiteres von Ihnen verwendet werden und bietet die Möglichkeit ohne große Programmierung die unterschiedlichsten HTTP Requests abzusetzen.

6.4.7 Beispielhafter webbasierter Client

Passend zum Backend wurde von uns ein minimaler Web – Client implementiert, welcher Ihnen im Browser zeigen soll wie so ein HTTP Request funktioniert.

Diesen Client finden Sie ebenfalls in Ilias zusammen mit dem Backend. Der Client befindet sich im Ordner Frontend.

Hinweis: Aktivieren Sie auch die Konsole im Browser, um die Antworten der Serverseite zu sehen.

6.5 Links und weitere Hilfen

HochschulIlias

<https://elearning.hs-albsig.de>

Datenbankmodell als PDF

Siehe den Praktikumsordner in Ilias

SQLite Datei zum Erstellen der Datenbanktabellen

Siehe den Praktikumsordner in Ilias

SQLite Datei mit rudimentären Insert-Statements

Siehe den Praktikumsordner in Ilias

Beispielhafter HTTP Web Client

Siehe den Praktikumsordner in Ilias

Node.js

<https://nodejs.org/en/>

<https://nodejs.org/en/download/>

Node.js Express Server mit Dateiupload

<https://attacomsian.com/blog/uploading-files-nodejs-express>

SQLite

<https://www.sqlite.org/index.html>

<https://www.sqlite.org/download.html>

<https://www.sqlite.org/docs.html>

DBeaver Community Edition

<https://dbeaver.io/>

<https://dbeaver.io/download/>

CRUD sowie RESTFul Services

Siehe Vorlesungunterlagen

<https://glossar.hs-augsburg.de/CRUD>

https://de.wikipedia.org/wiki/Representational_State_Transfer

JSON

https://de.wikipedia.org/wiki/JavaScript_Object_Notation

Postman

<https://www.getpostman.com/>

<https://www.getpostman.com/downloads/>

Visual Studio Code

<https://code.visualstudio.com/>