Aufgabe: Fahrzeugmarken- und Modell-API

Gebaut werden soll eine kleine REST-konforme API auf JSON-Basis welche die Verwaltung von Fahrzeugmodellen und Marken ermöglicht, die dazugehörige Datenbank soll ebenfalls modelliert werden.

Für den Endpunkt zum hinzufügen eines Fahrzeugmodells soll ein Integration-Test hinzugefügt werden, für die anderen Endpunkte muss kein Test hinzugefügt werden.

Eine Marke besteht aus: Name (Maximal 100 Zeichen), Preissegment (Auswahl aus: Niedrig, Mittel, Hoch) Ein Modell besteht aus: Name (Maximal 100 Zeichen), Motorleistung in kW (Gleitkommazahlen sind erlaubt).

Restriktion: Ein Modell ist immer mit einer Marke verknüpft, (1 Make : n Modelle), Modelle ohne Marken existieren nicht

Hinweise:

- Jede Datenbanktabelle hat eine ID als Primärschlüssel,der Entwurf der Datenbank soll in 3. Normalform erfolgen
- Fehlerfälle sollen adequat mit dem richtigen HTTP-Code und ebenfalls per JSON beantwortet werden.
- Die Anfragen werden ebenfalls in JSON gestellt. Das JSON Encoding sowie Decoding (serialisieren / deserialisieren) soll, wenn möglich, vom Framework erledigt werden
- Wir gehen davon aus, dass die Anwendung im geschützten Backbone läuft und eine Authentifizierung daher nicht notwendig ist.
- Die Daten der Fahrzeuge und der Marken sollen in PostgreSQL oder MySQL gespeichert werden. Eine Entwicklungsinstanz der Datenbank soll via docker-compose (docker-compose up -d) erzeugt werden können, die docker-compose.yml soll daher ebenfalls im Hauptverzeichnis committed werden.
- Das Datenbankschema soll als SQL-Skript am Ende committed werden, ein Schema-Management-Tool wie z. B. Flyway muss nicht eingebunden werden
- Unnötige Projektdateien sollen per .gitignore ausgeschlossen werden
- Bitte Kleinschrittig committen und auf die commit-Messages achten
- Die Abgabe der fertigen Lösung kann per Github (Link zum Repository) oder als gezipptes Git Repository (per Anhang)

Bei der Bewertung wird auf folgendes geachtet:

- Korrektheit: Die Anwendung erfüllt die Spezifikation, die Umsetzung erfolgte anhand der Hinweise
- Code-Qualität / Lesbarer Code: OOP-Prinzipien wurden richtig angewandt, Bezeichner wurden

sprechend gewählt, der Code wurde durch Kommentare an den richtigen Stellen um Informationen ergänzt, der Code wurde logisch aufgetrennt (z. B. in private-Funktionen, separate Klassen, etc.)