WildShop – technische Dokumentation

**Backend**

**HTTPS-Server**

* server.js
  + certificate: Zertifikat, welches für HTTPS-Autorisierung genutzt wird
  + privateKey: privater Schlüssel des Servers für HTTPS
  + authorize(): Prüft, ob User ein Session-Objekt mitsendet, wenn ja, Prüfung, ob es noch gültig ist im sessionHandler
  + responseUnauthorized(): Invalidiert Session (falls vorhanden) und schickt Statuscode 401 zurück
* router.js
  + app.route(„/url“) routet URL, die auf „url“ endet und mappt sie auf eine angegebene Funktion aus dem requestController
* requestController.js
  + showHome(): sendet (eventuelle) benötigte Daten für die Einstiegsseite nach dem LogIn/der Regestrierung
  + showProducts(): sendet alle vorhandenen Produkte zurück; hier muss noch eine Funktionalität zum Filtern her
  + addOrder(): Fügt einen neuen Eintrag in die Order-Tabelle und je nach Menge der Bestellungen n neue Einträge in OrderPruduct ein
  + login(): verifiziert User-Daten und sendet bei Erfolg ein Session-Objekt zurück, bei Fehler ist Statuscode 401 und Session-Objekt leer
  + register(): Erstellt einen neuen Eintrag in der User-Tabelle und sendet bei Erfolg ein Session-Objekt zurück (ACHTUNG: Senden einer Bestätigungsmail steht noch aus)

**Database**

* database.js
  + fügt Abhängigkeiten zwischen den Entitäten ein
  + init(): initialisiert die Datenbank und ihre Tabellen
  + softInit(): siehe init, löscht bestehende Tabellen nicht
  + execInit(): führt die Initialisierung durch
  + exportiert sämtliche Sequelize-Objekte für jede Tabelle
* database.json
  + beinhaltet Zugangsdaten zur Datenbank
* dbconnection.js
  + erstellt Verbindung zur DB
  + exportiert Sequelize-Objekt
* order.js
  + definiert Sequelize-Objekt für Order-Tabelle
* order\_product.js
  + definiert Sequelize-Objekt für Order\_Product-Tabelle
* preorder.js
  + definiert Sequelize-Objekt für PreOrder-Tabelle
* products.js
  + definiert Sequelize-Objekt für Product-Tabelle
* user.js
  + definiert Sequelize-Objekt für User-Tabelle
* shopping\_cart.js
  + nicht von Relevanz, da Warenkorb per Cookies realisiert wird
* shopping\_cart\_products.js
  + siehe shopping\_cart.js

**SessionHandler.js:**

* generateSessionObject():
  + erstellt mittels crypto.randomBytes neie Session-ID und gibt ein Session-Objekt mit userID, Erstellungszeitpunkt sowie Lastupdated zurück. Eintrag in eine Hash-Tabelle, die Sessions verwaltet.
* getSessionObject():
  + Prüfung, ob Session in Hash-Tabelle, wenn ja wird Session zurückgegeben, wenn nein undefined
* InvalidateSession():
  + Löscht Session aus Hash-Tabelle
* cleanSessions():
  + wird in vorgegebenem Zeitintervall aufgerufen und löscht alle Sessions aus Hash-Tabelle, die älter als x Minuten sind

**Frontend**

**Pages:**

* constants.ts
  + beinhaltet Klasse Constants, welche Konstanten wie z.B. die URL definiert, die häufig benötigt werden
* description.ts
  + showDescription(): bekommt via navParams die angeklickte Produkt-ID und sendet einen Request an den Server, der die Details zum Produkt liefert. Diese werden dann im HTML angezeigt
* shop.ts
  + getProducts(): Nutzt den RequestProvider und erhält als Antwort eine Liste aller Produkte. Hier sollte noch gefiltert werden!
  + goToDescription(product: Product): pusht die Seite DescriptionPage und liefert dabei die Produkt-ID mit für das Produkt, das angeklickt wurde

**Provider:**

* request.ts
  + getProducts(): Sendet den eigentlichen Request zum Erhalt aller Produkte ab, wird von mindestens einer Page verwendet
* interfaces.ts
  + definiert das Interface Product, welches immer verwendet werden sollte, wenn Requests ein Product-Objekt oder ein Array von Product-Objekten erhält
  + beinhaltet:
    - name: Name des Produkts
    - price: Preis des Produkts
    - stock: Bestand des Produkts
    - weight: Gewicht des Produkts
    - prID: Produkt-ID (eindeutig)