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, genau das gleiche bei Tabelle PreOrder und PreOrderProduct.
  + 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. Daraufhin wird Mail gesendet (in MailController realisiert).
  + showDescription(): Liefert Details zum ausgewählten Produkt und sendet diese an den Client
  + logout(): Invalidiert Session des Users und speichert dessen Warenkorb in Datenbank, falls vorhanden
  + confirm(): Nachdem auf den generierten Link in der Registrierungsmail des Users geklickt wurde, wird der User auf „active“ bzw. „authorized“ gesetzt -> nun kann er Käufe tätigen
  + deleteUser(): Falls User Account löschen wird, löscht diese Funktion dessen Einträge in der Datenbanktabelle
  + setPassword(): Falls keine Session vorhanden, wird ein generiertes Passwort an die angegebene E-Mail-Adresse versendet, falls ein Session-Objekt existiert bei der Anfrage, wird das aktuelle Passwort des Users durch das von ihm neu angegebene ersetzt
  + registerOrder(): wird aufgerufen, wenn Produkt in Warenkorb gelegt wird, sendet Status 200 und leeren String zurück, um Client zu signalisieren, dass Aktion realisiert wurde. Wird benötigt, damit das Session-Objekt im SessionHanlder alle aktuellen Produkte enthält, da die neu hinzugefügten Produkte beim Auslaufen der Session sonst nicht in die Datenbank mit übertragen werde
* sessionHanlder.js
  + generateSessionObject(): Generiert Session-ID, setzt Daten und User-ID. Gibt Session-Objekt zurück
  + getSession(): sucht in Hash-Tabelle nach der Session, die im Argument mitgegeben wird
  + invaldiateSession(): Löscht Session-ID aus der Session-Hash-Tabelle
  + cleanSessions(): Funktion, die regelmäßig aufgerufen wird und alte Session aus der Hashtabelle löscht
* orderController.js
  + processOrder(): Updated Tabelleneintrag für Produkt in Product-Tabelle (amount) und erstellt ein neuen Order-Eintrag in OrderProduct-Tabelle. Diese Funktion ist eine Hilfsfunktion und wird nicht im Modul exportiert.
  + insertOrder(): sucht einen existierenden Order-Eintrag in Order-Tabelle, und updated diesen mittels processOrder(). Falls kein Order-Eintrag vorhanden ist, wird ein neuer erstellt. Die Suche zuerst ist nötig, da eine Liste an Produkt-Objekten beim Server ankommt, die alle nacheinander in die Datenbank geschrieben werden. Wir diese Suche nicht durchgeführt, würde für jedes Objekt eine eigene Bestellung angelegt werden, wodurch eine eindeutige Zuordnung nicht mehr möglich wäre.
* preOrderController.js
  + processPreOrder():Updated Tabelleneintrag für Produkt in Product-Tabelle (amount) und erstellt ein neuen Order-Eintrag in PreOrderProduct-Tabelle. Diese Funktion ist eine Hilfsfunktion und wird nicht im Modul exportiert.
  + insertPreOrder(): sucht einen existierenden Order-Eintrag in PreOrder-Tabelle, und updated diesen mittels processPreOrder(). Falls kein PreOrder-Eintrag vorhanden ist, wird ein neuer erstellt. Die Suche zuerst ist nötig, da eine Liste an Produkt-Objekten beim Server ankommt, die alle nacheinander in die Datenbank geschrieben werden. Wir diese Suche nicht durchgeführt, würde für jedes Objekt eine eigene Bestellung angelegt werden, wodurch eine eindeutige Zuordnung nicht mehr möglich wäre.
* mailController.js
  + sendOrderConfirmation(): sucht User mittels User-ID und sendet eine generierte Bestätigungsmail für eine Bestellung.
  + sendRegConfirmation(): Sendet Bestätigungsmail nach der Registrierung. Ein generierter Link ist in der Mail und beim Klicken dieses Links wird der User in der User-Tabelle auf „authorized“ gesetzt, womit er Bestellungen aufgeben kann. Einloggen kann er sich jedoch auch ohne das Klicken des Links.
  + sendGeneratedPassword(): Sendet im Falle, das ein neues Passwort generiert werden muss, das neue Passwort an die Mailadresse des Users.
  + sendChangedPasswortConfirm(): Sendet Infomail an den User, der sein Passwort geändert hat. Dies geschieht, da es sein kann, dass der Account des Users missbraucht wird und er gar nicht selbst das Passwort geändert hat.
  + sendMail(): Registriert in einer Hashmap die Anzahl der Produkte, die der User bestellt hat. Mit dieser Information kann im Nachhinein für die Bestellung des Users eine Mail generiert werden, die alle Produkte mit der bestellten Anzahl auflistet
  + registerProductForMail(): Bei jedem Einfügen eines Produkts in Order/PreOrder, wird diese Funktion aufgerufen. Sie bekommt den Produktnamen, die orderID sowie die Anzahl der bestellten Exemplare. Sie dekrementiert die durch sendMail() registrierte Anzahl der Produkte in der Hashtabelle. Ist der Wert in der Hashtabelle 0, so sind alle Produkte abgehandelt und die Mail wird gesendet.
  + sendRegConfrimation(): Wrapper-Funktion für sendRegConfirmation()
  + sendGeneratedPassword(): Wrapper-Funltion für sendGeneratedPassword()
  + sendChangedPasswordConfirm() Wrapper-Funktion für sendChangePasswordConfirm()

**Logger**

* logger.js
  + log(): Nimmt eine Log-Message und ruft msgHanlder.addMessage auf
  + deleteLog(): ruft msgHandler.removeMessage auf, nimmt Log-Message
* messageHandler.js
  + constructor(): Im Konstruktor wird eine Hashtabelle, die die Messages beinhaltet, angelegt. Außerdem wird ein Dateideskriptor für die Log-Datei geöffnet
  + addMessage(): hängt an das Logfile die übergebene Nachricht an, fügt diese in der Hashtabelle ein
  + removeMessage(): sucht Zeile der Log-Message anhand des Indexes in Hashtabelle und löscht die Zeile, Eintrag wird auch in Hashtabelle gelöscht

**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
  + definiert Tabelle für Warenkorb, der beim Logout gesichert wird, falls vorhanden

**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
* saveShoppingCart():
  + wird nicht exportiert
  + Sichert den aktuellen Warenkorb des Nutzers beim Auslaufen der Session

**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
* confirmation.ts
  + sendOrder(): ruft RequestProvider auf, um Bestellung zu senden
    - führt Fehlerbehandlung nach Request durch
    - setzt Warenkorb auf „null“
  + removeFromCart(): Löscht Element von Warenkorb
* contact.ts
* home.ts:
  + goToXXX(): leiten auf die gewünschte Seite weiter
  + logout(): Anfrage an RequestProvider, um Logout durchzuführen, Fehlerbehandlung
* information.ts
* login.ts:
  + login(): ruft RequestProvider auf, um Login mit Daten der Eingabefelder zu machen, Fehlerbehandlung
  + register(): leitet zur Registrierungsseite weiter
  + hideButton(): setzt „Anmelden“-Button auf aktiv, wenn Eingaben korrekte Form haben
  + forgotPassword(): leitet auf NewPasswordPage weiter
* logout.ts:
  + goToLogin(): Leitet auf LoginPage weiter
* order.ts
  + addOrder(): fügt Produkt zu productArr in Session-Objekt hinzu
* register.ts
  + register(): nutzt RequestProvider, um Registrierungsdaten an Server zu übermitteln, Fehlerbehandlung
  + hideButton(): setzt „Registrieren“-Button aktiv, falls Eingabedaten die korrekte Form haben
* settings.ts
  + deleteUser(): Nutzt RequestProvider, um User zu löschen, Fehlerbehandlung
  + changePasswort(): leitet auf NewPasswordPage weiter
* new-password.ts
  + hideButton(): setzt „Passwort ändern“-Button aktiv, falls Eingabedaten die korrekte Form haben
  + changePasswort(): nutzt RequesstProvider, um Passwort zu ändern, Fehlerbehandlung

**Provider:**

* request.ts
  + getProducts(): Sendet den eigentlichen Request zum Erhalt aller Produkte ab, wird von mindestens einer Page verwendet
  + getDescription(): Sendet Request ab, der die Description des Produktes x anfordert
  + login(): Fordert Login an, sendet E-Mail und Passwort
  + register(): Fordert Registrierung an, sendet E-Mail, Namen, etc.
  + sendOrder(): sendet eine Bestellung. Eine Bestellung wird aus einem Array aus Objekten repräsentiert, eingebettet im Session-Objekt
  + logout(): Fordert logout an, sendet Session-Objekt mit
  + logoutWithoutSession(): Weiterleitung zur Login-Page
  + setPassword(): Sendet Session mit neuem Passwort an Server, ggf. ein Mail-Objekt, das die Mailaddresse des Users beinhaltet, falls der User das Passwort vergessen hat
  + deleteUser(): Löscht den User aus der Datenbank -> Account wird gelöscht
  + registerProduct(): Sendet Session-Objekt mit dem aktuellen productArr, damit dieser immer den aktuellen Warenkorb hat. Wichtig beim Auslaufen der Session.
* interfaces.ts
  + Product:
    - name: Name des Produkts
    - price: Preis des Produkts
    - amount: Bestand des Produkts
    - weight: Gewicht des Produkts
    - pid: Produkt-ID (eindeutig)
    - pic: String-Repräsentation eines Bildes
    - description: Beschreibung des Produkts
  + User:
    - userName: Benutzername
    - userID: ID
  + Session:
    - sessionID: Session-ID
    - begin: Startzeitpunkt der Session
    - updatedAt: Letzte Registrierung beim Server
    - userID: Nutzer
    - productArr: Array, das die Bestellungen beinhaltet
  + Order:
    - userID: Nutzer-ID
    - prodID: Produkt-ID
    - amount: Anzahl
    - comment: Kommentar (z.B. Sonderwünsche)