Dokumentation – Kommunikation zwischen Frontend und Backend

Um den Austausch von Daten zwischen Frontend und Backend zu ermöglichen, mussten sich zunächst auf einheitliche Befehle in Form von URI's (Uniform Ressource Identifier) und Attribute (z.B. E-Mail und Passwort beim Login) geeinigt werden. Die folgende Tabelle zeigt alle Kommunikationen, welche zwischen Backend und Frontend implementiert wurden. Außerdem kann man sehen, welche Anfrage über JavaScript gesendet wird und welche Antwort der Server an das Frontend sendet.

GET / POST / DELETE	BENÖTIGTE URI	ANFRAGE: BODY (im JSON oder String-Format)	Antwort: Body (JSON-Format)
POST	http://localhost:8080/testdaten/alle (alle Testdaten werden in der Datenbank automatisiert angelegt)	-	-
POST	http://localhost:8080/vorstellungen/filmbekommen (wenn ein Film ausgewählt wurde, werden die dazugehörigen Vorstellungen angefragt und im Frontend angezeigt)	Terminator	[{ "vorstellungsid": 1, "filmName": "Transformers", "kinosaalNummer": 1, "startuhrzeit": "12:00", "laengeDerVorstellungInMinuten": "175" }, { … }] (verkürzt)
POST	http://localhost:8080/vorstellungen/sitzplaetze (wenn daraufhin eine Vorstellung ausgewählt wurde, werden die Sitzplätze und deren Status angefragt, um diesen im Frontend anzuzeigen)	{ "filmName":"Transformers", "startuhrzeit":"12:00" }	[{ "sitzplatzID": 161, "reihe": 1, "spalte": 1, "vorstellungsID": 1, "statusVomSitzplatz": "BESETZT" }, { … }] (verkürzt)
POST	http://localhost:8080/kunden/register (Wenn sich ein Kunde registrieren möchte, werden die Daten, welche im Frontend eingegeben werden an	{ "vorname":"Max", "nachname": "Mustermann", "email":"max@gmail.com", "alterInJahren": 20,	True oder False

	den Server gesendet und in der Datenbank gespeichert – Wenn dieser Vorgang gelingt -> Antwort = True)	"strasse":"Musterstrasse", "hausnummer":"7a", "postleitzahl":"50667", "ort":"Musterstadt", "passwort":"TestPasswort" }	

POST	http://localhost:8080/kunden/login (Bei einem Login wird nur die E-Mail (eindeutiger Identifikator) und das Passwort eingegeben. Falls diese stimmen, erhält man das jeweilige Kundenobjekt aus der Datenbank)	{
		"email":"max@gmail.com",
		"passwort": "TestPasswort"
		}
			{
			"id": 25,
			"vorname":"Max",
			"nachname": "Mustermann",
			"email":"max@gmail.com",
			"alterInJahren": 20,
			"strasse":"Musterstrasse",
			"hausnummer":"7a",
			"postleitzahl":"50667",
			"ort":"Musterstadt",
			"passwort":"TestPasswort"
			"verifiziert": **false**
			}

POST	http://localhost:8080/bestellungen (Hier wird ein eindeutiger Identifikator für die jew. Bestellung angefragt)	{
		"email":"max@gmail.com"
		}
			1 -> entspricht der BestellID

POST	http://localhost:8080/tickets (Das jeweilige Ticket wird an den Server gesendet und für 15 Minuten auf den Status “reserviert” gesetzt)	{
		"startuhrzeit": "15:00",
		"kinosaalNummer": 4,
		
		"filmName":"Transformers",
		"preis": "20",
		"sitzplatzreihe": 1,
		"sitzplatzspalte": 1,
		"bestellungID": 1
		}
			True oder False

POST	http://localhost:8080/bestellungen/emailsenden (wenn die Bestellung abgeschlossen wurde wird die BestellID an den Server gesendet)	1 -> entspricht BestellID	True oder False
POST	http://localhost:8080/kunden/passwortvergessen (wenn ein Kunde sein Passwort vergessen hat)	max@gmail.com	True oder False

Beispiel: Eine Anfrage an den Server senden

Für alle oben aufgelisteten Anfragen an den Server haben wir jeweils eine Asynchrone Funktion in JavaScript erstellt.

Zunächst wird eine neue Instanz der Klasse XMLHttpRequests erzeugt. Dann wird eine neue Anfrage von dem angegeben Typ (z.B. POST) mit der festgelegten URI geöffnet. Daraufhin wird die Anfrage mit den notwendigen Daten (JSON oder String-Format) an den Server gesendet. Wenn die Anfrage an den Server erfolgreich war, wird die success-Funktion geöffnet, welche nun mit der Antwort des Servers arbeiten kann. Falls die Anfrage nicht erfolgreich war, wird eine Error-Funktion aufgerufen, welche über das Scheitern der Anfrage informiert.

Anzeigen der Daten im Frontend

Die Daten, welche wir durch die Antwort vom Server auf eine unserer Anfragen erhalten, soll nun teilweise im Frontend angezeigt werden. In den meisten Fällen wie z.B. bei der Vorstellung wird nur auf ein bestimmtes HMTL-Element über dessen ID zugegriffen und der jeweilige Text angepasst. Bei den Sitzplätzen jedoch erhalten wir ein Array mit allen Sitzplätzen für einen ausgewählten Film und eine ausgewählte Vorstellung. Jedes Element dieses Arrays stellt einen Sitzplatz da. Durch eine for-Schleife werden alle Elemente durchlaufen. Zunächst wird die ID des jeweiligen Sitzplatzes durch die Zusammensetzung verschiedener Strings bestehend aus der Reihe und Spalte des jeweiligen Sitzplatzes ermittelt. So kann identifiziert werden welcher Sitzplatz in dem Array zu welcher ID im Frontend gehört. Dann wird der Status des aktuellen Elements geprüft. Wenn dieser ungleich “FREI” ist, wird die Klasse des Elements mit der zuvor ermittelten ID auf “occupied” gesetzt. Da die verschiedenen Klassen in der CSS-Datei unterschiedliche Farben zugeordnet bekommen haben bewirkt eine Änderung der Klasse schlussendlich eine Veränderung der Farbe im Frontend.

Zwischenspeichern von Daten im SessionStorage

Da nicht alle Daten sofort an den Server gesendet und einige Daten erst zu einem späteren Zeitpunkt benötigt werden haben wir uns über die Möglichkeiten informiert Daten im Browser des Benutzers zu speichern. Dabei sind wir auf den SessionStorage und den Localstorage von JavaScript gestoßen. Der Unterschied liegt lediglich bei der Haltbarkeit und dem Geltungsbereich. Die Daten des Sessionstorage sind begrenzt auf eine Session, werden also gelöscht, sobald man das Browserfenster schließt. Der SessionStorage ist außerdem auf ein Individuelles Fenster beschränkt, was z.B. der Falle entgegentritt, dass während eines Einkaufs bei mehreren geöffneten Tabs oder Browserfenstern der Kauf aus Versehen zweimal ausgelöst wird. Aus diesem Grund haben wir uns für den Sessionstorage entschieden.

Unser Bestellprozess sieht es vor, dass der Benutzer ohne Gastkonto, Login oder Registrierung den Film, die Vorstellung und seine Sitzplätze auswählen kann. Dies ist notwendig, falls ein Benutzer sich nur über Vorstellungszeiten oder die aktuelle Auslastung einer Vorstellung informieren möchte. Zunächst wird also ein Film ausgewählt, dann eine Vorstellung und daraufhin die Sitzplätze und (optional) das alter der Personen (für z.B. Rabatte). Dann folgt das Eintragen der E-Mail in Form von einem Login, einer Anmeldung oder einem temporären Gastkonto (an Session gekoppelt). An diesem Punkt wird über das Senden der E-Mail eine eindeutige BestellID für die aktuelle Bestellung bei dem Server angefragt und in den Sessionstorage zwischengespeichert. Über diese BestellID können nun die Zwischengespeicherten Informationen an den Server gesendet werden. In diesem Zuge werden die mitgesendeten Sitzplätze für 15 Minuten auf den Status “reserviert” gesetzt. Die Sitzplätze sollen erst auf “reserviert” gesetzt werden können, wenn der Benutzer seine E-Mail angegeben hat, da ab diesem Moment die Bestellabsicht klar ist. Dann kann nur noch die Bezahlart ausgewählt werden. Zum Abschließen der Bestellung wird nur die BestellID an den Server gesendet, welcher daraufhin den Status aller Sitzplätze mit dieser BestellID auf “besetzt” schaltet, sofern die 15 Minuten noch nicht abgelaufen sind.

Sitzplätze zwischenspeichern

Das Speichern der Sitzplätze im SessionStorage wurde durch ein Array realisiert. Da im SessionStorage nur Strings gespeichert werden können, musste das Array also über die Methode JSON.stringify() in einen String umgewandelt werden. Wenn dieses wieder benötig, wurde, konnte es durch die Methode JSON.parse() wieder in ein Array umgewandelt werden.