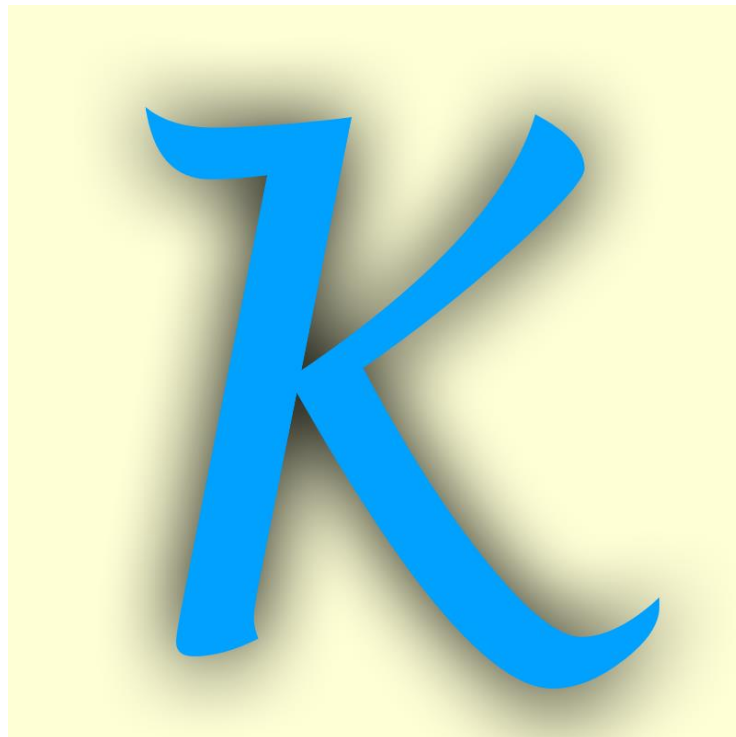


# Programmierung einer Kellner-App



Kantonsschule Im Lee Winterthur

Maturitätsarbeit HS 2017/18

Michael Heider (4a)

Betreuungsperson: Rolf Kleiner

Winterthur, 8. Januar 2018



## Inhaltsverzeichnis

1. Vorwort.....	3
2. Abstract.....	4
3. Einleitung .....	5
4. Entwicklung.....	6
4.1. Web-App.....	6
4.2. Programmiersprachen.....	6
4.3. Programme .....	7
4.3.1. Notepad++ .....	7
4.3.2. FileZilla.....	7
4.4. Webspaces.....	7
4.5. Hilfsmittel .....	8
5. Die App .....	9
5.1. Funktionen .....	9
5.1.1. Übersicht.....	9
5.1.2. Anwendungsbereich und Vorteile .....	9
5.1.3. Satellit .....	11
5.1.4. Basis .....	13
5.1.5. Sprinter .....	14
5.1.6. Betriebsvoraussetzungen.....	15
5.2. Ausblick .....	15
5.3. Bereits erhältliche Alternativen.....	15
6. Programm .....	17
6.1. Datenbankstruktur.....	17
6.2. Bestellen im Detail .....	19
6.2.1. HTML .....	19
6.2.2. JavaScript .....	21
Darstellungsverzeichnis.....	23
Abbildungen .....	23
Listings .....	23
Tabellen .....	23
Quellenverzeichnis .....	24
Anhang.....	25
Programmstrukturpläne .....	25
Quellcode .....	28



## 1. Vorwort

Seit einigen Jahren helfe ich an der alljährlichen Chilbi meiner Wohngemeinde mit. Durch die wachsende Anzahl der Besucher ist das Personal der „Waldbeiz“ zunehmend überlastet. Dank dem allmählichen Aufkommen spezieller Geräte zur Digitalisierung der Bestellabläufe lag die Idee nahe, dass so etwas, in abgespeckter Form, auch für Festwirtschaften zu haben sein müsste. Da die bestehenden Systeme aber deutlich zu umfangreich und kompliziert sind, entschloss ich mich vor gut einem Jahr dazu, im Rahmen eines Projektes im Informatikfreifachunterricht Abhilfe zu schaffen. Schnell zeigte sich jedoch, dass die Idee viel zu umfangreich dafür war, aber in den Rahmen einer Maturitätsarbeit passen würde.

Am Programmieren faszinieren mich schon lange die unglaublichen Möglichkeiten und die schier unbegrenzte Komplexität, die man mit ganz simplen Bausteinen von Grund auf erschaffen kann. Durch das bereits erwähnte Freifach erlangte ich auch das Basiswissen, um ein solches Projekt zu realisieren. Allerdings hatte ich mit den fortgeschrittenen Funktionen der Internetprogrammierwelt dann doch zu kämpfen.

Meiner Betreuungsperson Rolf Kleiner möchte ich meinen Dank ausdrücken. Er konnte mich zwar nicht direkt beim Programmieren unterstützen, worauf er mich im vornherein auch hingewiesen hat, gab mir aber wertvolle Tipps zu den benötigten Funktionen und praktischen Layouts für die App. Ein weiterer Dank geht an Salome Marti und das Team des Restaurants Kafichanne in Tagelswangen, in welchem ich freundlicherweise meine App einen Nachmittag lang testen durfte. An dieser Stelle auch ein grosses Danke an meine Klassenkameradinnen und Klassenkameraden, welche mir ihre Tablets zu diesem Zweck ausgeliehen haben. Des Weiteren möchte ich mich bei Niko Kapelis, Geschäftsführer des Restaurants Gate27 in Winterthur, bedanken, dass er mir das von ihm verwendete System Gastrofix gezeigt hat. Ausserdem möchte ich mich auch bei meiner Familie bedanken. Sie hat mich zu den praktischen Funktionen der App beraten, die schriftliche Arbeit korrigiert, die Karte der Kafichanne digitalisiert und mir den Rücken freigehalten.

## 2. Abstract

In modern ausgerüsteten Restaurants sind heute elektronische, extra für das Aufnehmen von Bestellungen entwickelte Geräte Standard. Die im Rahmen dieser Arbeit programmierte Kellner-App soll die Vorteile eines solchen digitalen Systems nun auch für kleine Anlässe wie zum Beispiel Dorffeste zur Verfügung stellen. Die freiwilligen Servicehelfer verwenden heutzutage einen Bestellblock mit Stift, was einerseits zeitaufwändig ist, da die Bestellungen jeweils zur Küche und zur Getränkeausgabe zurück getragen werden müssen, und andererseits fehleranfällig beim Berechnen des Preises, da dies im Kopf geschieht und mit der Zeit anstrengend wird.

Die Kellner-App, welche auf sämtlichen modernen Touchgeräten läuft, ermöglicht also das digitale Erfassen von Bestellungen, sendet diese automatisch an Küche und Getränkebuffet und zeigt auf Knopfdruck den Gesamtpreis an. Selbstverständlich ist auch geteiltes Bezahlen kein Problem.

### 3. Einleitung

Bei kleineren Veranstaltungen mit Restaurant, wie beispielsweise Dorf- oder Turnfeste, werden Freiwillige als Kellnerinnen und Kellner eingesetzt. Um die Bestellungen aufzunehmen, dienen ihnen bisher der klassische Bestell-Block mit Stift. Diese Variante hat jedoch viele Nachteile. So müssen zum Beispiel beim Bezahlen die Preise zuerst auf einer Liste gesucht und dann im Kopf zusammengezählt werden, was natürlich viel Zeit beansprucht und extrem fehleranfällig ist. Dies steht im Gegensatz zu den elektronischen, extra dafür entwickelten Geräten, welche in grösseren, vernünftig ausgerüsteten Restaurants Standard sind. Diese sind für kleinere und vor allem temporäre Anlässe jedoch zu teuer und zu aufwändig betreffend Inbetriebsetzung und Instruktion der Helfer.

Das Ziel dieser Arbeit ist nun in Form einer App eine bessere Alternative zur Bestellblock-Methode zu schaffen, welche angelehnt ist an die elektronischen Geräte. Smartphones eignen sich hervorragend für diese Aufgabe, da in der heutigen Zeit fast jeder eines mitträgt. Anschaffungskosten allfälliger Geräte entfallen somit komplett. Am Getränkebuffet und in der Küche, wo der Übersicht halber grössere Bildschirme gebraucht werden, können Tablets, welche heute auch verbreitet sind, eingesetzt werden. Kompatibilität für sämtliche Betriebssysteme wie zum Beispiel Android oder iOS kann erreicht werden, indem die App nicht als konventionelle, sogenannte native App, sondern als Web-App umgesetzt wird.

Eine der wichtigsten Anforderungen an eine solche Software ist die Benutzerfreundlichkeit. Die Kellner, das Buffetpersonal sowie die Köche müssen innerhalb kürzester Zeit eingewiesen werden können. Da die freiwilligen Kellner an Festveranstaltungen üblicherweise unterbesetzt sind, müssen alle Vorgänge so effizient, also mit so wenigen Bildschirmberührungen wie möglich, ausgeführt werden können. Des Weiteren ist es mit diesem System möglich, das Servicepersonal aufzuteilen in solche, die Bestellungen aufnehmen sowie einkassieren und andere, die Bestellungen ausliefern. So müssen nicht alle Geld mittragen, was das Risiko von Fehlern beim Rechnen und Einkassieren verringert, aber auch Betrug vorbeugt.

Die vorliegende Arbeit befasst sich vor allem mit der programmierten App, ihren Anforderungen und Funktionen. Des Weiteren finden sich einige Angaben zur Programmierung wie die benötigten Programmiersprachen und die verwendeten Programme. Der genaue Programmcode sowie zugehörige Programmstrukturpläne liegen im Anhang vor.

## 4. Entwicklung

### 4.1. Web-App

Web-Apps sind zwar erst vor wenigen Jahren erfunden worden, sind aber extrem praktisch. Der Unterschied zu konventionellen Apps, auch native Apps genannt, ist für den Benutzer sehr klein. Lediglich der Installationsprozess ist unterschiedlich. So sind Web-Apps nicht in den App Stores zu finden, sondern werden als Internetseite aufgerufen. Die Installation beschränkt sich darauf, die Seite als Lesezeichen auf dem Homescreen abzulegen, worauf das App Symbol, wie bei einer nativen App, erscheint.

Für den Programmierer bedeuten Web-Apps eine enorme Arbeitsverringering. Native Apps sind plattformspezifisch. Das heisst, dass die Apps für jedes Betriebssystem, zum Beispiel Android oder iOS, separat und in unterschiedlichen Programmiersprachen programmiert werden müssen. Web-Apps hingegen müssen nur einmal programmiert werden und funktionieren auf sämtlichen Betriebssystemen. Dies liegt daran, dass eine Web-App eigentlich nur eine Internetseite ist, welche aber mit wenigen Zeilen Code die Adresszeile und sonstige Browselemente ausblendet und auch sonst für den Benutzer wie eine native App dargestellt wird. Sollte ein Betriebssystem oder ein Browser Web-Apps noch nicht unterstützen, so kann immer noch die Internetseite direkt im Browser verwendet werden. Da es sich eigentlich um eine Internetseite handelt, kann sie bei Bedarf sogar am Computer geöffnet werden.

Mittlerweile gibt es sogar Software, um einfache Web-Apps in native Apps umzuwandeln, so dass sie auch in den verschiedenen App Stores angeboten werden können.<sup>1</sup>

### 4.2. Programmiersprachen

HTML (HyperText Markup Language), CSS (Cascading Style Sheets) und JavaScript sind die drei Programmiersprachen, welche für das Aussehen und die Funktionalität einer Webseite verantwortlich sind. Alle drei sind heute absolute Standards.

HTML beschreibt den Inhalt und Aufbau einer Webseite. CSS bestimmt das Aussehen und ermöglicht komplexe Layouts. JavaScript kommt zum Einsatz, sobald etwas berechnet werden muss. Die Programmierschnittstelle jQuery für JavaScript vereinfacht viele Prozesse, die in irgendeiner Form HTML oder CSS beinhalten. Die intuitive Schreibweise macht die komplizierten und vor allem langen und damit unübersichtlichen Standard-HTML-Funktionen überflüssig. Die Entwicklung wird einfacher und schneller und der Code lesbarer.

---

<sup>1</sup> URL: <http://www.selfhtml5.org/apps-programmieren/> (27.12.2017).

Für Funktionen, die nicht direkt auf der Webseite, sondern auf einem Server ausgeführt werden müssen, wird meistens PHP (Hypertext Preprocessor) verwendet. PHP unterstützt auch viele Datenbanken, so auch die verbreitete SQL (Structured Query Language) Sprache, welche in diesem Projekt verwendet wird.

## 4.3. Programme

### 4.3.1. Notepad++

Notepad++ ist ein gratis Quellcodebearbeitungsprogramm. Es sticht hervor durch seine einfache Bedienbarkeit und unterstützt viele Programmiersprachen. Für jede Sprache stellt Notepad++ eine übersichtliche und gut lesbare Farbkodierung zur Verfügung. In begrenztem Umfang können sogar verschiedene Sprachen innerhalb eines Dokumentes entsprechend unterschiedlich farblich hervorgehoben werden. Da unter anderem HTML, CSS, JavaScript und PHP unterstützt werden, ermöglicht es mir, für das ganze Projekt nur ein einziges Codebearbeitungsprogramm zu verwenden. Des Weiteren erlaubt das Programm, Code inklusive der Farbkodierung zu exportieren, um ihn so vernünftig formatiert in andere Dokumente zu integrieren.

### 4.3.2. FileZilla

FileZilla ist ein kostenloses FTP (File Transfer Protocol) Programm. FTP wird verwendet, um Dateien von der Festplatte auf einen Server hochzuladen. Das Programm überzeugt durch Übersichtlichkeit, einfache Bedienung sowie mit der praktischen Drag-and-Drop-Funktion, um das Hoch- und Runterladen von Dateien zu starten. Alternativ zu FTP Programmen gibt es auch FTP taugliche Webseiten wie zum Beispiel Online FTP.

## 4.4. Webspaces

Der Webspaces für meine Web-App wird zur Verfügung gestellt von bplaced. Dies ist ein österreichischer Freehoster, stellt den Webspaces also gratis und in diesem Falle sogar werbefrei zur Verfügung. Er bietet auch die Möglichkeit an, MySQL- oder PostgreSQL-Datenbanken anzulegen, welche dann auf einem MariaDB Server gespeichert werden<sup>2</sup>. Alle von bplaced verwendeten Server stehen in Europa<sup>3</sup>.

---

<sup>2</sup> URL: <http://www.bplaced.net/?location=home> (29.12.2017).

<sup>3</sup> URL: <http://www.bplaced.net/?location=contact> (29.12.2017).

## 4.5. Hilfsmittel

Bei Fragen zu komplexeren Funktionen und Problemen hilft das Internet meistens weiter. Insbesondere erwähnenswert sind W3schools, das Mozilla Developer Network und die offizielle PHP-Seite php.net für ihre ausführlichen Dokumentationen zu JavaScript, HTML, CSS, PHP, SQL-Datenbanken und anderem. Bei Problemen mit jQuery hält die offizielle Dokumentationsseite die Antwort bereit. Das Forum Stack Overflow, in dem fast alle erdenklichen Fragen bereits gestellt und beantwortet wurden, hilft ebenfalls oft weiter.

## 5. Die App

### 5.1. Funktionen

#### 5.1.1. Übersicht

Das Programm ist unterteilt in drei Sektionen, wovon jede eine eigene Web-App ist. Die Unterteilung widerspiegelt die Arbeitsteilung des Personals. Das Servicepersonal, welches Bestellungen aufnimmt und die Zeche einzieht, verwendet den Satellit-Teil. Dieser stellt die nötigen Funktionen bereit. Jede Kellnerin und jeder Kellner verwendet dabei die App auf einem eigenen Smartphone und hat Zugriff auf sämtliche Tische und sämtliche offenen Bestellungen. Es können beliebig viele Kellnerinnen und Kellner gleichzeitig eingesetzt werden.

Daneben gibt es Zentralstationen. Eine Zentralstation ist überall dort vonnöten, wo Produkte produziert oder bereitgestellt werden. So in der Küche, wo Essen zubereitet wird, oder am Getränkebuffet, wo die Getränke bereitgestellt werden. Es ist möglich, mehrere Zentralstationen einzurichten, welche jeweils nur bestimmte bestellte Produkte anzeigen. Beispielsweise werden in der Küche nur die zu kochenden Esswaren angezeigt, während Getränke und Desserts an einer anderen Zentralstation angezeigt werden. Auch hier gibt es keine Begrenzung, wie viele verschiedene Zentralstationen eingesetzt werden. Jede Zentralstation besteht aus zwei Tablets oder sonstigen Touchgeräten mit grösserem Bildschirm. Dabei steht eines, genannt Basis, direkt in der Küche oder an der Getränkeausgabe und zeigt an, welche Produkte in welcher Reihenfolge produziert werden müssen. Das andere, genannt Sprinter, steht an der Warenausgabe und ist sichtbar für diejenigen Angestellten, welche die Waren an den entsprechenden Tisch tragen. Das Sprinter-Gerät zeigt dementsprechend an, welche bereitgestellten Produkte zu welchem Tisch müssen.

#### 5.1.2. Anwendungsbereich und Vorteile

Das System ist ausgelegt auf Festanlässe. Deshalb sind alle Vorgänge so gestaltet, dass sie so schnell wie möglich, also mit wenigen Bildschirmberührungen, ablaufen. Die Bedienung ist selbsterklärend, so dass auch ungeübte Helfer sich ohne Probleme zurechtfinden und keine Zeit für die Instruktion der Helfer verloren geht, zumal diese ja sowieso nicht gleichzeitig eintreffen und alle einzeln instruiert werden müssten.

Die verschiedenen Teile der App ermöglichen zudem eine für Festwirtschaften optimale Arbeitsteilung. Es kann gesondertes Personal zum Austragen der Waren eingeteilt werden, welches kein Geld und kein Gerät benötigt.

Das Servicepersonal, das zuständig für das Aufnehmen und Einkassieren von Bestellungen ist, muss dank der automatischen Übertragung der Bestellungen an die Zentralstationen nicht wie bei der konventionellen Block-und-Stift-Methode zurück zur Küche gehen, sondern kann sich direkt auf die nächste Bestellung fokussieren. Dies bedeutet eine enorme Zeiteinsparung und trägt erheblich zur Entlastung des meist unterbesetzten und überlasteten freiwilligen Servicepersonals bei. Das normalerweise nötige Kopfrechnen für den Gesamtpreis wird mit der Zeit sehr anstrengend und entsprechend fehleranfällig. Erhebliche Einnahmeverluste sind die Folge. Das Wegfallen davon trägt weiter zur Entlastung des Personals bei.

Selbstverständlich ist es möglich, Bestellungen geteilt zu bezahlen. Da gedruckte Quittungen an Festen nicht üblich sind und sowieso keine Drucker vorhanden sind, entfällt dieses Problem komplett. Allerdings gibt es eine Quittungsansicht, welche einer gedruckten Quittung ähnelt und den Gästen gezeigt werden kann. Das Vertrauen des Gastes ist dank dieser Kontrolle erheblich höher, als wenn der Gesamtpreis wie aus dem Nichts angekündigt wird.

Am Ende des Tages kann zudem automatisch eine Übersicht über die verkauften Artikel erstellt werden. Diese zeigt auf, wie viel von jedem Artikel bestellt wurde, aber auch wie viel davon tatsächlich bezahlt beziehungsweise storniert wurde.

Festwirtschaften haben kein grosses Budget zur Verfügung und bestehen nur wenige Tage im Jahr. Entsprechend können sie nicht mehrere Tausend Franken in Spezialgeräte investieren. Da die Web-Apps aber geräteunabhängig funktionieren, kann die Software flexibel für kure Zeitperioden günstig vermietet werden.

Das gesamte System kam im Restaurant Kafichanne in Tagelswangen einen Nachmittag lang testweise zum Einsatz. Die Funktionalität des Systems konnte überprüft und bestätigt werden.

### 5.1.3. Satellit

Mit den Satellit-Geräten können alle offenen Bestellungen eingesehen und bezahlt sowie neue Bestellungen aufgenommen werden. Nachdem aus der Übersicht ein Tisch gewählt worden ist, erscheinen nach kurzer Ladezeit alle noch offenen auf diesen Tisch laufenden Bestellungen in chronologischer Reihengfolge (Abbildung 1). Als offene Bestellung gelten alle Bestellungen, welche noch nicht bezahlt oder noch nicht ausgeliefert worden sind.

Um die genauen Artikel zu sehen, können die Bestellungen aufgeklappt werden, wie dies in der Abbildung 1 bei der obersten Bestellung der Fall ist. Die Zahlen jeweils am rechten Rand geben Auskunft über die bestellte Menge des Artikels und wie viel davon bereits bezahlt worden ist.

Bestellungen, welche bereits komplett bezahlt worden sind, haben anstelle einer grauen Box einen schwarzen Haken. So ist auf den ersten Blick ersichtlich, ob eine Bestellung bereits bezahlt worden ist.

Um Bestellungen oder Teile davon zu bezahlen, werden sie in der rechten, grauen Box angewählt, wodurch darin ein Haken erscheint. So können auch mehrere Bestellungen kombiniert bezahlt werden. Nach einem Click auf den grünen Knopf unten rechts erscheint die Bezahl-Ansicht.

In der Bezahl-Ansicht (Abbildung 2) können die Produkte, welche bezahlt werden sollen, durch Berührung auf die rechte Seite gebracht werden. Getrennte Bezahlung ist also möglich. Der Knopf „Quittung“ oben rechts listet die ausgewählten Artikel im Stil einer klassischen Quittung auf und berechnet den Gesamtpreis. Diese Übersicht kann dann auch dem Gast gezeigt werden, wodurch dieser mehr Kontrolle über den fälligen Betrag erhält.

Nicht einkassierbare Artikel werden mit dem Storno-Knopf abgearbeitet. So markierte Produkte werden einfachheitshalber in den verschiedenen Ansichten als bezahlt angezeigt. In der Tagesübersicht werden sie aber gesondert aufgeführt.



Abbildung 1: Tischübersicht



Abbildung 2: Bezahl-Ansicht

Von der Tischübersicht aus kann auch eine neue Bestellung auf den angezeigten Tisch aufgenommen werden. Damit erscheinen alle Kategorien. Wenn eine geöffnet wird, erscheinen die darin enthaltenen Produkte wie in **Abbildung 3** gezeigt. In jeder Kachel sind der Name und der Preis des Produkts angegeben und zusätzlich wie viele von diesem Artikel bereits bestellt worden sind. Unten links wird ausserdem das Zwischentotal angezeigt. Durch den Löschen-Knopf unten rechts ist es ausserdem möglich, Produkte wieder zu entfernen. Der Weiter-Knopf führt zu einer Liste, in welcher die Bestellung noch einmal übersichtlich dargestellt wird, bevor sie definitiv abgeschlossen wird.

Sämtliche Ansichten des Satelliten sind auf die Bildschirmgrösse von Smartphones optimiert. Smartphones eignen sich hervorragend, da sie leicht und handlich sind.

Kategorien	Getränke: Wein, Bier, Most	Weiter
Allaman rouge Fr. 15.00	Waldbeizwein Fr. 15.00 x1	Goldtröpfli Fr. 15.00 x0
Nürensdorfer Fr. 15.00 x0	Quöllfrisch Fr. 4.50 x2	Holzfassbier Fr. 4.00 x0
Feldschlösschen o.A. Fr. 4.00 x1	Saurer Most Fr. 4.50 x2	Saurer Most o.A. Fr. 4.50 x0
Eve Lichi Fr. 4.00 x0		
Fr. 37.00		löschen

Abbildung 3: Neue-Bestellungs-Ansicht

### 5.1.4. Basis

Sämtliche neuen Bestellungen werden auf den Basis-Geräten dargestellt. Die Bestellungen können vorgefiltert werden, so dass beispielsweise wie in Abbildung 4 in der Küche nur die Esswaren angezeigt werden.

Die neuen Bestellungen kommen dabei in der rechten und linken Hälfte unten an die Bestehenden. Die Idee ist, dass die Bestellungen, sobald sie in Arbeit sind, mit einem Click auf „gesehen“ von der Neu-Seite (rechts) gelöscht werden. Sobald die nötigen Artikel bereit sind und parat stehen zum Austragen, kann die Bestellung auch aus der „in Produktion“-Seite

in Produktion (alt)	neu
Id: 19, Zeit: 14:49 (Tisch: Inn 1, Von: Alice) <span style="background-color: red; color: white; padding: 2px;">verarbeitet</span>	Id: 20, Zeit: 14:49 (Tisch: Aus 5, Von: Alice) <span style="background-color: red; color: white; padding: 2px;">gesehen</span>
Essen	Essen
Servelat mit Brot 2	Pommes 1
Bratwurst mit Brot 2	
Id: 20, Zeit: 14:49 (Tisch: Aus 5, Von: Alice) <span style="background-color: red; color: white; padding: 2px;">verarbeitet</span>	Id: 25, Zeit: 14:50 (Tisch: Aus 2, Von: Bob) <span style="background-color: red; color: white; padding: 2px;">gesehen</span>
Essen	Essen
Pommes 1	Servelat mit Brot 2
	Bratwurst mit Brot 2
	Waldbeizwurst mit Brot 1
Id: 23, Zeit: 14:50 (Tisch: Inn 5, Von: Bob) <span style="background-color: red; color: white; padding: 2px;">verarbeitet</span>	Pommes 2
Essen	
Servelat mit Brot 1	
Bratwurst mit Brot 1	
Jägerschnitzel mit Brot 2	
Id: 25, Zeit: 14:50 (Tisch: Aus 2, Von: Bob) <span style="background-color: red; color: white; padding: 2px;">verarbeitet</span>	
Essen	
Servelat mit Brot 2	
Bratwurst mit Brot 2	
Waldbeizwurst mit Brot 1	
Pommes 2	

Abbildung 4: Basis

(links) gelöscht werden. Dadurch wird die Bestellung intern als verarbeitet markiert und erscheint auf dem entsprechenden Sprinter-Gerät.

Zusätzlich ist es möglich, einzelne Produktzeilen in der Neu-Hälfte durch Antippen als gesehen zu markieren, wodurch diese ausgegraut werden. Dies ist hilfreich, falls mehrere Personen direkten Zugriff auf das Gerät haben. Dasselbe ist auch in der „in Produktion“-Hälfte möglich, allerdings können diese nicht separat auf den Sprinterbildschirm geschickt werden. Die Bestellung muss als Ganzes als verarbeitet markiert werden, um auf dem Sprinterbildschirm zu erscheinen.

Im Gegensatz zum Satelliten ist die Basis auf grössere Bildschirme ausgelegt. Der grössere Bildschirm ist nötig, da an Festwirtschaften nicht selten mehrere Dutzend Bestellungen gleichzeitig verarbeitet werden und diese entsprechend viel Platz benötigen, um angezeigt zu werden.

### 5.1.5. Sprinter

Alle von der zugehörigen Basis-Station als verarbeitet markierten Bestellungen werden in der Sprinter-Ansicht (Abbildung 5) aufgeführt. Die zuletzt als verarbeitet markierten erscheinen dabei zuunterst. Das Personal, welches eingeteilt ist um Bestellungen auszutragen, kann aufgrund dieser Ansicht sehen, zu welchem Tisch es die entsprechenden Artikel tragen muss. Sobald klar ist, wohin die Artikel gehören, kann durch Berührung der grauen Fläche die Bestellung als ausgeliefert markiert werden, wodurch sie aus der Ansicht verschwindet.

Bestellungen	
<b>Tisch: Inn 1, Menge: 4, (ID: 8, Zeit: 13:05)</b>	■
Essen	
Servelat mit Brot	2
Bratwurst mit Brot	2
<b>Tisch: Aus 4, Menge: 9, (ID: 16, Zeit: 13:10)</b>	■
Essen	
Servelat mit Brot	4
Bratwurst mit Brot	2
Waldbeizwurst mit Brot	1
Pommes	2
<b>Tisch: Aus 5, Menge: 1, (ID: 10, Zeit: 13:06)</b>	■
Essen	
Pommes	1

Abbildung 5: Sprinter

Wie die Basis sind auch die Sprinter auf grössere Bildschirme ausgelegt. Zusätzlich ist die Schrift einiges grösser, so dass die nötigen Informationen auch von weiter weg zu erkennen sind.

### 5.1.6. Betriebsvoraussetzungen

Für einen erfolgreichen Betrieb des Systems braucht es pro gleichzeitig aktivem Kellner ein Smartphone mit der installierten Satellit-Web-App. Pro Zentralstation braucht es zudem zwei Tablets, eines mit der Basis-App und eines mit der Sprinter-App. Dabei ist sicherzustellen, dass diese Geräte zu jedem Zeitpunkt genügend Akkuladung aufweisen. Es müssen also Ladekabel, externe Akkus oder Ersatzgeräte bereit stehen. Jedes Gerät braucht eine dauerhafte Verbindung zum Internet, um mit der Datenbank kommunizieren zu können. Diese kann über ein eingerichtetes WLAN-Netzwerk erfolgen, aber auch über die mobile Datenverbindung.

Im Vorfeld müssen zudem die gesamte Speisekarte sowie die vorhandenen Tische digitalisiert werden. Da diese direkt in eine Datei in einem speziellen Format geschrieben werden müssen, wird dies direkt vom Entwickler erledigt. Er setzt zudem die Datenbank auf und bereitet die nötigen Unterteilungen der Zentralstationen sowie die gewünschte Anzahl Satellit-Geräte vor.

## 5.2. Ausblick

Das System ist einsatzfähig. Ein Testlauf im Restaurant Kafichanne in Tagelswangen hat dies bestätigt.

Bis jetzt muss das System allerdings noch vom Entwickler aufgesetzt werden, sodass die gewünschte Anzahl Zentralstationen und Satelliten zur Verfügung stehen. Dieser Vorgang sollte wegfallen beziehungsweise automatisiert werden. Zusätzlich sollte es möglich sein, beim Starten oder bei der Installation anzugeben, welchem Event das Gerät angehört, sodass sich verschiedene Feste automatisch nicht in die Quere kommen und auch hier das Eingreifen des Entwicklers wegfällt. Des Weiteren braucht es einen Passwortschutz, sodass nicht jeder darauf zugreifen kann.

Um die App vermarkten zu können, muss der Markt geprüft werden. Danach kann die App mittels einer zu erstellenden Homepage veröffentlicht werden. Ziel ist, die App an der diesjährigen Chilbi meiner Gemeinde einzusetzen.

## 5.3. Bereits erhältliche Alternativen

Bei einem POS-System (Point of Service) haben alle Kellner ein tragbares, elektronisches Gerät, mit dem sie die Bestellung beim Gast digital aufnehmen und direkt in die Küche und zur Getränkeausgabe senden. Zusätzlich werden sämtliche gesetzlich vorgeschriebenen Abrechnungen automatisch erstellt. Solche Systeme werden zunehmend auch in der

Schweiz eingesetzt. Bisher werden speziell dafür entworfene Geräte mit darauf abgestimmter Software verwendet. Als Zentralstationen dienen ebenfalls spezielle Geräte. Deshalb sind diese Systeme auch sehr teuer.

Tatsächlich sind auch bereits erste Systeme verfügbar, welche ohne spezielle Hardware, sondern nur mit erhältlichen mobilen Geräten, wie zum Beispiel iPads, funktionieren. Die bestehenden Varianten haben allerdings den Nachteil, dass sie nur für ein Betriebssystem erhältlich sind. Tatsächlich gibt es bis jetzt kein System, das auf Android und iOS Geräten läuft und beide Betriebssysteme kombinieren kann. Zusätzlich sind sämtliche Systeme für Restaurantbetriebe ausgelegt und damit aufgrund der vielen Funktionen schwierig zu bedienen und teuer.

Momentan ist kein POS-System erhältlich, das auf die Bedürfnisse von Festwirtschaften abgestimmt und für diese erschwinglich ist.

## 6. Programm

Programmstrukturpläne zu den drei Teilen Satellit, Basis und Sprinter finden sich im Anhang.

### 6.1. Datenbankstruktur

Alle Bestellungen werden in einer Datenbank auf einem Server gespeichert. Wer wann genau auf die Datenbank zugreift, kann den Programmstrukturplänen im Anhang entnommen werden. Tabelle 1 zeigt einige mögliche Einträge.

id	zeitstempel	zeit	tisch	verarbeitet	ausgeliefert	bestelltVon
1	2017-12-20 16:55:08	16:44:44	1	1111	1111	Alice
2	2017-12-20 16:59:37	16:52:23	3	0001	0001	Alice
3	2017-12-20 16:55:10	16:55:10	1	0000	0000	Bob
4	2017-12-20 17:05:10	17:00:10	2	1111	0001	Bob

bezahltVon	bestellt	bezahlt	storno
Bob	[0,4,0,0,1,0,3,1,0,0,0,0,...	[0,4,0,0,1,0,3,1,0,0,0,0,...	
Alice	[1,1,0,2,0,0,0,0,0,0,0,1,...	[1,1,0,2,0,0,0,0,0,0,0,1,...	[1,0,0,1,0,0,0,0,0,0,0,1,...
	[4,2,1,0,0,2,0,0,0,0,0,0,...	[0,0,0,0,0,0,0,0,0,0,0,0,...	
Bob	[0,3,0,0,0,0,3,0,0,0,0,0,...	[0,1,0,0,0,0,2,0,0,0,0,0,...	

Tabelle 1: Auszug aus der Datenbank

Jede Bestellung erhält von der Datenbank automatisch eine eindeutige, fortlaufende *ID*. Diese dient der eindeutigen Erkennung einer Bestellung und ermöglicht ein schnelleres Finden einer bestimmten Bestellung in der Datenbank. Der *Zeitstempel* wird auch direkt von der Datenbank erstellt und speichert die zuletzt-geändert-Zeit. Diese wird vom Programm zwar nicht verwendet, ist aber sehr hilfreich beim Überwachen der Datenbank während dem Programmieren. Die *Zeit* ist die Uhrzeit, wann die Bestellung aufgenommen wurde.

Die *Tisch*-Spalte gibt an, auf welchem Tisch die Bestellung läuft. Es wird allerdings nicht die Bezeichnung des Tisches gespeichert, sondern dessen interne ID-Nummer.

*Verarbeitet* und *ausgeliefert* enthalten jeweils eine Zahl im Binärformat. Jedes Bit steht dabei für eine Kategorie von Produkten. Das erste Bit ganz rechts steht für die Kategorie mit ID null, das zweite Bit für die Kategorie eins und so weiter. Die Binärzahl hat also so viele Stellen, wie es Kategorien gibt. *0* bedeutet jeweils noch nicht verarbeitet beziehungsweise noch nicht ausgeliefert, während *1* für verarbeitet oder ausgeliefert steht. Dieses Speicherformat ermöglicht es, Bestellungen als nur teilweise verarbeitet oder ausgeliefert zu speichern. Die Getränke sind ja meistens schneller bereit als Essen, das erst noch vorbereitet werden muss. Die Speicherung im Binärformat ermöglicht zudem das gezielte Wechseln bestimmter Bits unabhängig von den Werten der anderen Bits mittels Binärlogik. Dies ist wichtig, weil die Änderung in einem Schritt vollzogen wird und so allfällige Interferenzen zwischen mehreren Geräten, welche gleichzeitig das gleiche Feld bearbeiten wollen, vermieden werden.

*BestelltVon* und *bezahltVon* enthalten eindeutige Identifizierungen von Satellit-Geräten. *bestelltVon* speichert, wer die Bestellung ursprünglich aufgenommen hat und *bezahltVon* wer zuletzt etwas bezahlt oder storniert hat.

*Bestellt* enthält die Gesamtmenge der bestellten Produkte. *Bezahlt* ist eine Teilmenge davon und *storno* wiederum eine Teilmenge von *bezahlt*. Das heisst, dass alle stornierten Produkte auch als bezahlt markiert sind. Alle drei Spalten enthalten Arrays als Strings im JSON-Format (JavaScript Object Notation). Die Einträge bezeichnen, wie viel von welchem Produkt vorhanden ist. So entspricht die nullte Listenposition dem Produkt mit ID null, die erste Position dem Produkt eins und so weiter.

## 6.2. Bestellen im Detail

In Abbildung 6 dargestellt ist die Bestell-Ansicht (im Code genannt *kacheln*), worauf sich die nachfolgenden Unterkapitel beziehen. Im Bild wurde bereits die Kategorie „Getränke: Wein, Bier, Most“ ausgewählt, so dass nun von diesen Produkten bestellt werden kann. Wie alle Ansichten ist auch diese aufgeteilt in Header, Mittelteil und Footer. Der Header ist jeweils aufgeteilt in drei unterschiedlich farbige Bereiche. Der linke, orange Bereich dient als Zurück- oder Abrechnen-Knopf. In diesem Fall führt er zurück zur Kategorienübersicht. In der gelben Mitte wird angezeigt, wo in der App sich der Nutzer gerade befindet. Der grüne Knopf ganz rechts hat abhängig von der Ansicht eine andere Funktion. Im Beispiel öffnet er eine Liste, welche nochmal die gewählten Produkte auflistet, bevor diese definitiv bestellt werden. Im Footer befinden sich weitere Knöpfe oder Angaben. Bei dieser Ansicht wird das Zwischentotal angezeigt sowie ein Knopf mit welchem Produkte wieder gelöscht werden können. Durch dieses sich durch die verschiedenen Ansichten durchziehende Layout und Farbmuster bietet die App ein zusammenhängendes Nutzungsgefühl und erscheint sehr übersichtlich.

Kategorien	Getränke: Wein, Bier, Most	Weiter
Allaman rouge Fr. 15.00 x0	Waldbeizwein Fr. 15.00 x1	Goldtröpfli Fr. 15.00 x0
Nürensdorfer Fr. 15.00 x0	Quöllfrisch Fr. 4.50 x2	Holzfassbier Fr. 4.00 x0
Feldschlösschen o.A. Fr. 4.00 x1	Saurer Most Fr. 4.50 x2	Saurer Most o.A. Fr. 4.50 x0
Eve Lichi Fr. 4.00 x0		
Fr. 37.00		löschen

Abbildung 6: Bestell-Ansicht des Satelliten

### 6.2.1. HTML

Praktischerweise ist durch das konsistente Layout der HTML-Code der verschiedenen Ansichten sehr ähnlich. Abgedruckt in Listing 1 ist der Code zuständig für die oben beschriebene Bestell-Ansicht (im Code genannt *kacheln*).

Die gesamte Ansicht ist eingefasst in ein grosses Div-Element (kurz für division, auf Deutsch etwa Bereich) mit ID *kacheln*. So kann die Ansicht vom JavaScript-Programm einfach als Ganzes identifiziert und versteckt werden, sowie eine andere Ansicht nach dem gleichen Prinzip angezeigt werden. Im Header Section-Element befinden sich normalerweise drei, hier aber vier Div-Elemente, welche jeweils einen Knopf darstellen. Das vierte kommt daher, dass der Kategorien-Knopf nach Berührung ausgetauscht wird durch einen Abrechnen-Knopf. Es werden hier absichtlich nicht die eigentlich richtigen Button-Elemente verwendet, da diese auf mobilen Geräten fehlerhafte Klickanimationen auslösen. Die verschiedenen Klassen sind für die korrekte Darstellung der Grösse, Farbe und Umrandung zuständig. Die genauen Werte dafür sind in der zugehörigen CSS-Datei gespeichert. Die onclick-Eigenschaften definieren, was passiert, wenn der Knopf geklickt wird. So ist beispielsweise ersichtlich, dass

der Weiter-Knopf die Funktion *ansicht(liste)* aufruft, welche dafür verantwortlich ist, die nötigen Vorbereitungen zu treffen und die Ansicht zu *liste* zu wechseln, so dass eine Liste mit allen gewählten Produkten eingeblendet wird. Der Footer ist ähnlich aufgebaut wie der Header. Im Mittelteil befindet sich nur das Div-Element mit ID *alleKacheln*. Dies liegt daran, dass die einzelnen Kacheln und deren Bestandteile beim Starten der App von einer JavaScript-Funktion dynamisch erzeugt und hier eingefügt werden. Sie werden also nicht im HTML-Code, sondern im JavaScript-Code definiert.

```
1 <!-- ..... KACHELN ..... -->
2 <div id="kacheln" style="display:none;">
3   <!-- HEADER -->
4   <section class="header rahmen">
5     <!--nur eine der beiden Navigationen gleichzeitig sichtbar-->
6     <div id="navigationKategorie"
7       class="kategorieNavigation rahmen schrift"
8       onclick="kategorieOffnen(-1)">Kategorien</div>
9     <div id="navigationAbbrechen"
10      class="kategorieNavigation rahmen schrift"
11      onclick="bestellungAbbrechen()">abbrechen</div>
12     <div id="jetzigeKategorie"
13      class="jetzigeKategorie rahmen grosseschrift">
14       INITIAL_jetztigeKategorie</div>
15     <div class="fertig rahmen schrift"
16      onclick="ansicht('liste')">Weiter</div>
17   </section>
18
19   <!-- MITTE -->
20   <section class="mitte">
21     <!--Produkt- und Kategorie-Kacheln mit JavaScript-->
22     <div id="allekacheln"></div>
23   </section>
24
25   <!-- FOOTER -->
26   <section class="footer rahmen">
27     <p id="subAngaben" class="rahmen">
28       <var id="subTotal">INITIAL_Preis</var>
29     </p>
30     <!--Anfangszustand: class modusBestellen
31      (abbestellen = true/false)-->
32     <div id="abbestellen"
33      class="rechteck rahmen schrift modusBestellen"
34      onclick="modusAbbestellen()">löschen</div>
35   </section>
36 </div>
```

Listing 1: HTML-Code der Bestell-Ansicht des Satelliten

## 6.2.2. JavaScript

In der Bestells-Ansicht (im Code genannt *kacheln*) kann von den gezeigten Produkten durch Antippen bestellt werden. Bei jedem Produkt zeigt zudem die kleine Zahl an, wie viele von diesem Artikel bereits bestellt sind. Diese Zahlen werden im globalen Array *bestellt* gespeichert. Die Position innerhalb des Arrays entspricht dabei der jeweiligen ID des Produktes.

Der Löschen-Knopf unten rechts ändert den Modus so, dass anstatt ein Produkt hinzugefügt eines abgezählt wird. Das Programm setzt hierfür die globale Variable *abbestellen* auf *true*. Durch erneutes Berühren des Löschen-Knopfes, wird die Variable wieder auf *false* gesetzt und der Modus zurückgesetzt. Das unten angezeigte Zwischentotal wird gespeichert in der globalen Variable *totalPreis*.

```
1 //Produkt (ab)bestellen wenn auf Kachel geklickt
2 //Position id des bestellt-Arrays plus/minus 1
3 //id: ID des zu bestellenden Produkts
4 function bestellen(id) {
5     if (abbestellen) {
6         //eins abzählen
7         if (bestellt[id] > 0) { //keines abzählen, falls bereits 0
8             bestellt[id]--;
9             totalPreis -= produkte[id].preis;
10        }
11    } else {
12        //eins hinzufügen
13        bestellt[id]++;
14        totalPreis += produkte[id].preis;
15    }
16
17    //Menge in Produkte Kachel aktualisieren
18    mengeDiv[id].text(mengeAnzeige(bestellt[id], true));
19    //Angaben unten aktualisieren
20    $("#subTotal").text(preisAnzeige(totalPreis, true));
21 }
```

Listing 2: JavaScript-Code der Funktion "bestellen"

Wenn eine Produktschaltfläche angetippt wird, ruft diese die Funktion *bestellen*, von welcher der Code in Listing 2 abgedruckt ist, mit der ID des entsprechenden Produktes als Argument auf. Die Funktion prüft in Zeile 5 zuerst den Wert von *abbestellen*. Ist dieser *true*, der Löschen-Knopf also angewählt, so prüft die Funktion, ob vom betreffenden Produkt noch mindestens eines bestellt ist. Es sollen schliesslich keine negativen Mengen möglich sein. Sollte dies nicht der Fall sein, so geht die Funktion direkt weiter zu Zeile 17. Ist aber noch mindestens eines bestellt, so wird vom entsprechenden Wert im *bestellt*-Array eines abgezogen. In Zeile 9 wird zudem vom Zwischentotal der Preis des abbestellten Produktes abgezählt. Der verwendete globale *produkte*-Array enthält sämtliche Informationen zu den

Produkten, welche jeweils als Objekt vorliegen. Sollte der Wert von *abbestellen false* sein, so geschieht vom Prinzip her das gleiche, nur dass nun hinzu- statt abgezählt wird.

In den Zeilen 17 bis 20 werden die Angaben zur bestellten Menge des Produktes in der betreffenden Kachel sowie das Zwischentotal unten auf dem Bildschirm aktualisiert. Die Funktionen *mengeAnzeige* und *preisAnzeige* sorgen dabei für die korrekte Darstellung der Werte.

# Darstellungsverzeichnis

## Abbildungen

Titelbild: Icon der App (Spezialversion in HD)

Abbildung 1: Tischübersicht .....	11
Abbildung 2: Bezahl-Ansicht.....	11
Abbildung 3: Neue-Bestellungs-Ansicht.....	12
Abbildung 4: Basis.....	13
Abbildung 5: Sprinter .....	14
Abbildung 6: Bestell-Ansicht des Satelliten.....	19

## Listings

Listing 1: HTML-Code der Bestell-Ansicht des Satelliten .....	20
Listing 2: JavaScript-Code der Funktion "bestellen" .....	21

## Tabellen

Tabelle 1: Auszug aus der Datenbank.....	17
--	----

## Quellenverzeichnis

selfHTML5: App Programmierung. URL: <http://www.selfhtml5.org/apps-programmieren/> (27.12.2017).

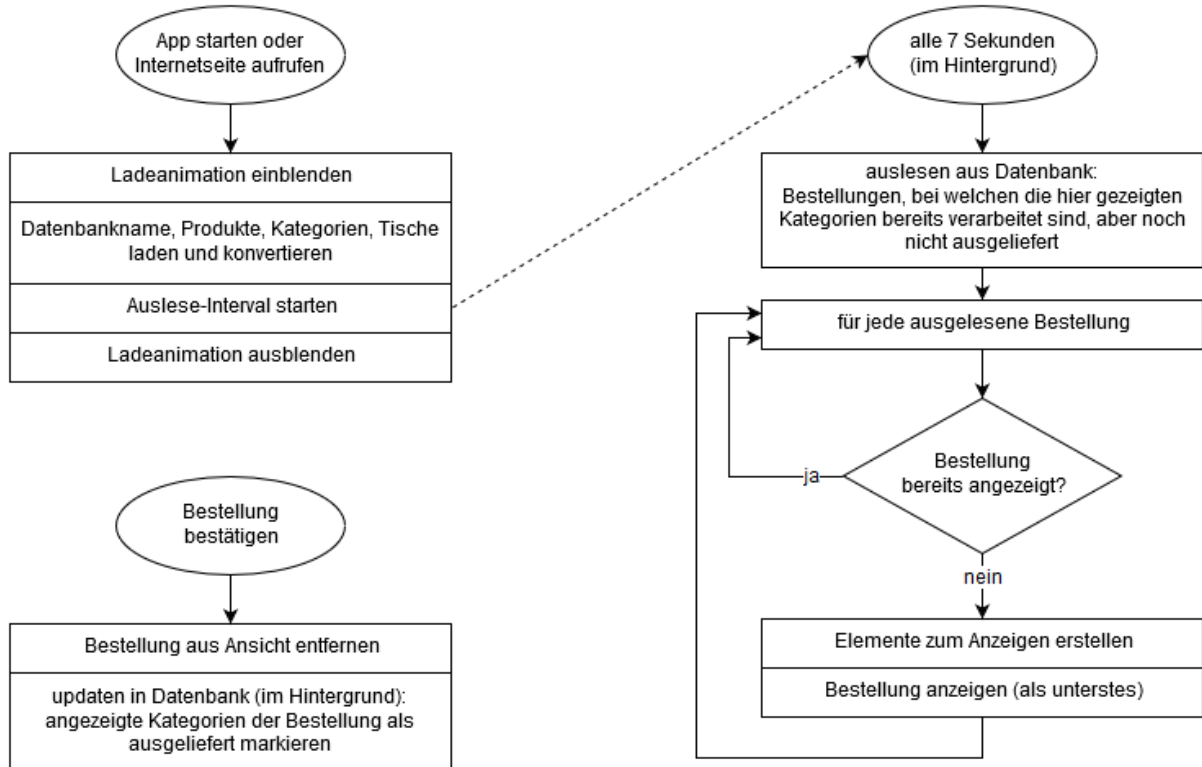
Offizielle bplaced-Homepage. URL: <http://www.bplaced.net/?location=home> (29.12.2017).

Offizielle bplaced-Homepage. URL: <http://www.bplaced.net/?location=contact> (29.12.2017).

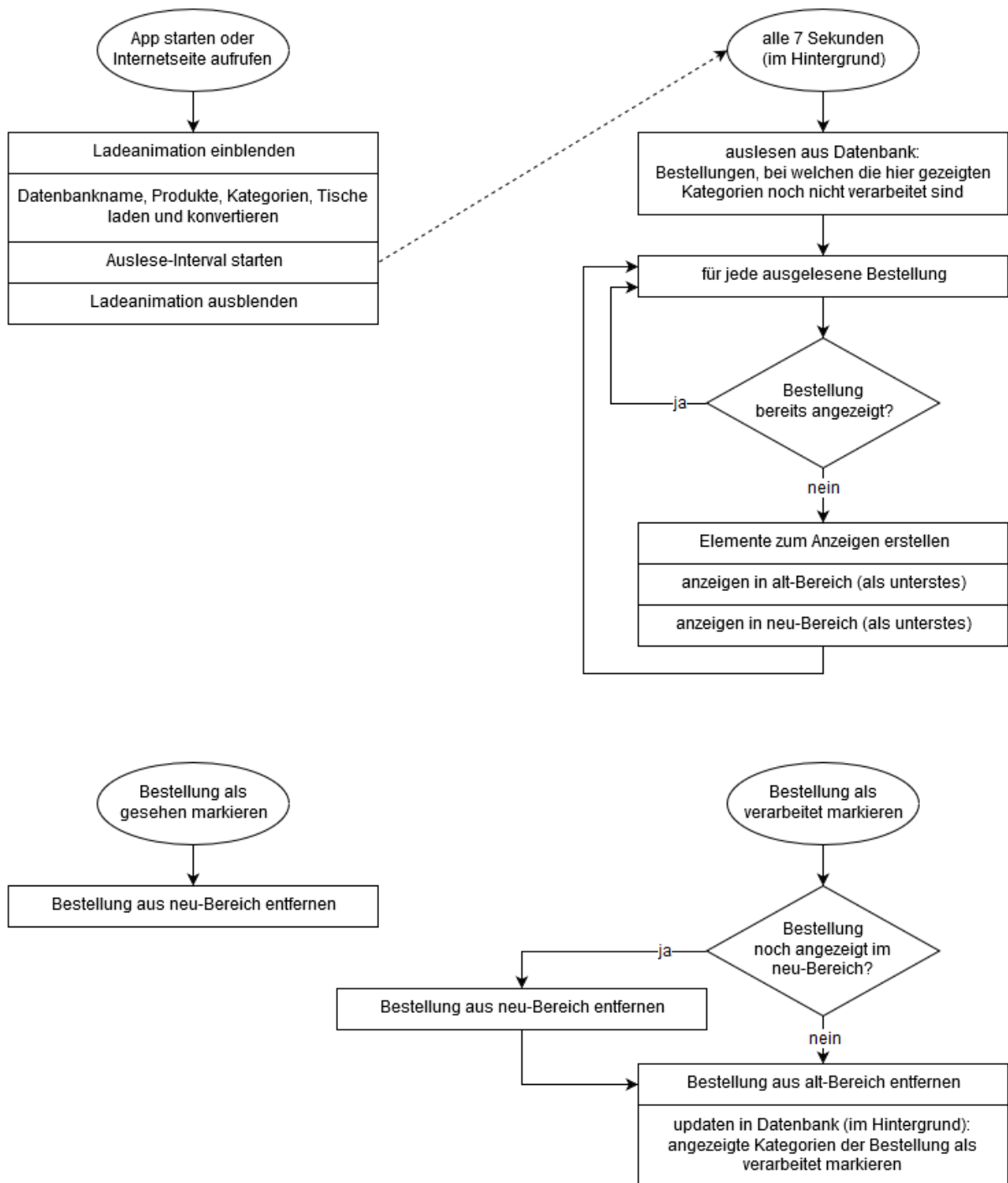
# Anhang

## Programmstrukturpläne

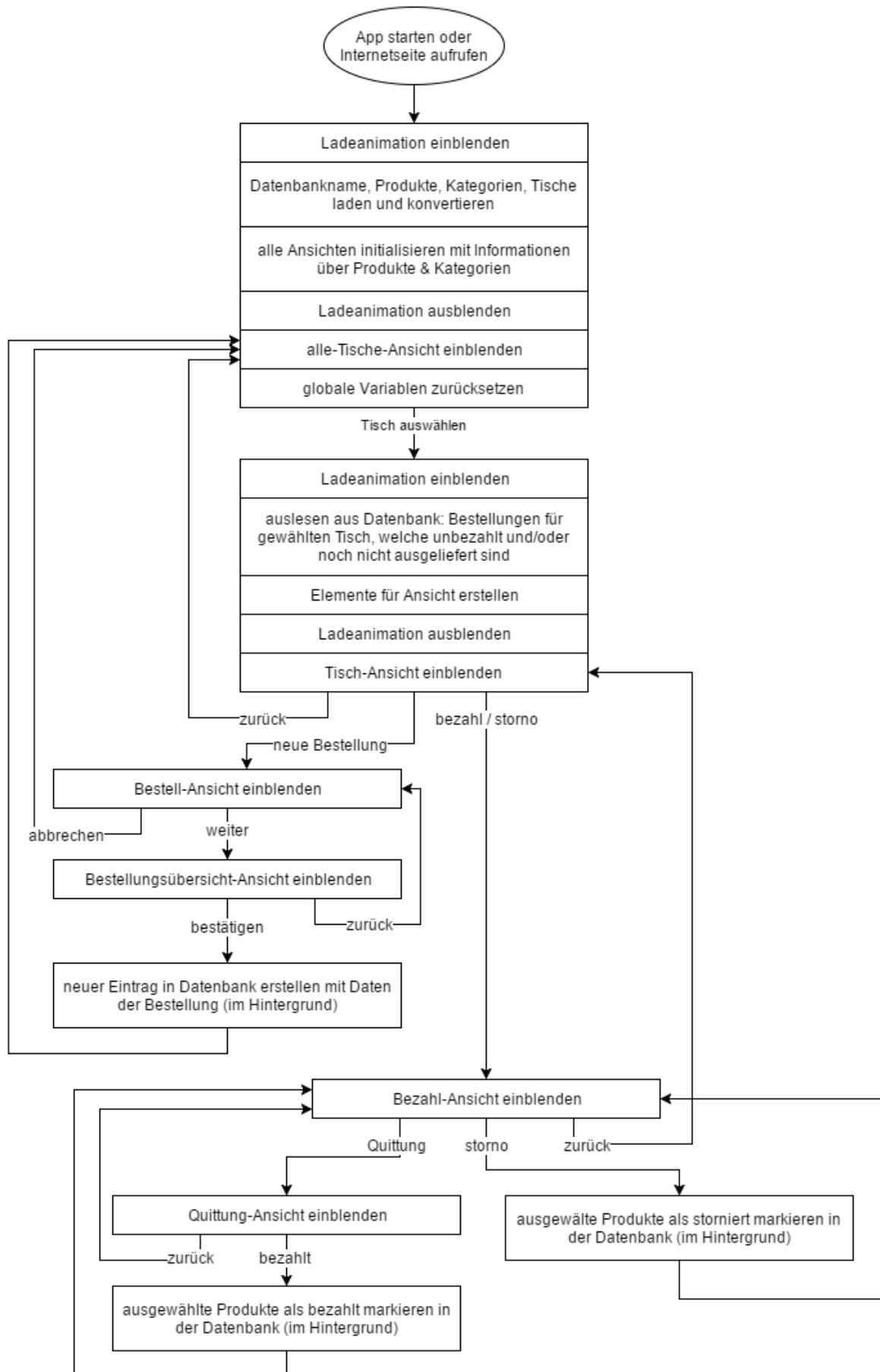
Sprinter:



Basis:



Satellit:



## Quellcode

1.	database .....	29
1.1.	auslesenAusliefern.php .....	29
1.2.	auslesenId.php .....	29
1.3.	auslesenTisch.php .....	30
1.4.	auslesenUnverarbeitet.php .....	30
1.5.	eventErstellen.php.....	31
1.6.	eventExistiert.php.....	31
1.7.	insertBestellung.php.....	32
1.8.	loginDaten.php .....	33
1.9.	SQL.php .....	33
1.10.	SQLprepared.php.....	33
1.11.	tagesUbersicht.php .....	35
1.12.	updaten.php .....	37
2.	eventData.....	39
2.1.	eventData.json .....	39
2.2.	kategorien.json .....	39
2.3.	produkte.json.....	39
2.4.	tische.json .....	40
3.	scripts.....	41
3.1.	allgemeineFunktionen.js .....	41
3.2.	polyfill.js.....	50
4.	stylesheets .....	51
4.1.	gemeinsam.css .....	51
4.2.	reset.css .....	52
5.	Web-App.....	53
5.1.	manifestBasis.json .....	53
5.2.	manifestSatellit.json .....	53
5.3.	manifestSprinter.json .....	53
5.4.	icon180x180.png.....	54
5.5.	startup1900x1000.png .....	54
6.	basis.css .....	55
7.	basis.html.....	56
8.	basis.js .....	57
9.	satellit.css .....	61
10.	satellit.html.....	65
11.	satellit.js.....	67
12.	sprinter.css .....	79
13.	sprinter.html.....	80
14.	sprinter.js .....	81

## 1. database

### 1.1. auslesenAusliefern.php

```

1  <?php
2  /*
3   * Datei: auslesenAusliefern.php
4   * Projekt: Kellner App
5   * Autor: Michael Heider
6   * Datum: HS 2017
7   *
8   * Was:
9   * auslieferbare Bestellungen auslesen
10  * ALLE der gegebenen Kategorien verarbeitet UND noch MIN 1 nicht ausgeliefert
11  * Rückgabe: Array von Arrays (Zeilen) mit Spaltennamen als Indece oder Fehlermeldung
12  */
13
14
15 //SQLprepared.php einbinden
16 require "SQLprepared.php";
17
18 //AUSLESEN
19 $tabelleName = $_REQUEST["eventName"];
20
21 $zeigeKat = (array)$REQUEST["zeigeKat"]; //Array zu prüfender Kategorien-IDs
22 // (genaue Prüfregele siehe oben)
23 // falls leer: gibt nichts zurück
24
25
26
27 //FAKE-AUSLESEN
28 /*echo "ERROR_Fake-Auslesen aktiviert<br>";
29 $tabelleName = "Event05";
30 $zeigeKat = [0]; //Array zu prüfender Kategorien-IDs
31 */
32
33
34 //Arrays erstellen
35 $katBin = 0;
36 foreach ($zeigeKat as $key => $value) {
37     $katBin += 2 ** $value;
38 }
39 unset($value); //letztes $value entfernen
40
41 $where = [$katBin, $katBin, $katBin];
42
43 //SQL vorbereiten
44 $sql = "SELECT id, zeit, tisch, verarbeitet, ausgeliefert, "
45     . "bestelltVon, bezahlt, storno, bezahltVon, bestellt FROM $tabelleName "
46     . "WHERE ((ausgeliefert & ?) <> ?) AND (verarbeitet & ?)";
47 //wird zu: ((ausgeliefert & katBin) <> katBin) AND (ausgeliefert & katBin)
48
49 //ausführen
50 $ausgelesen = select($tabelleName, $sql, $where);
51
52 //ohne JSON wird "Array" zurückgegeben
53 //mit json_encode: Array von JS-Objects
54 print json_encode($ausgelesen);
55
56 ?>

```

### 1.2. auslesenId.php

```

1  <?php
2  /*
3   * Datei: auslesenId.php
4   * Projekt: Kellner App
5   * Autor: Michael Heider
6   * Datum: HS 2017
7   *
8   * Was:
9   * Bestellungen mit bestimmten IDs auslesen (via Parameter)
10  * Rückgabe: Array von Arrays (Zeilen) mit Spaltennamen als Indece oder Fehlermeldung
11  */
12
13
14 //SQL_prepared.php einbinden
15 require "SQLprepared.php";
16
17 //AUSLESEN
18 $tabelleName = $_REQUEST["eventName"];
19
20 $id = (array)$REQUEST["id"]; //Array von IDs
21 // falls leer []: alle auslesen
22 // (leerer Array wird nicht automatisch konvertiert)
23
24
25
26 //FAKE-AUSLESEN
27 /*echo "ERROR_Fake-Auslesen aktiviert<br>";
28 $tabelleName = "Event05";
29 $id = []; //Array von IDs
30 */
31
32
33 //SQL vorbereiten
34 $sql = "SELECT id, zeit, tisch, verarbeitet, ausgeliefert, "
35     . "bestelltVon, bezahlt, storno, bezahltVon, bestellt FROM $tabelleName ";
36

```

```

37
38 if (count($id) != 0) { //falls keine Bestellungen-IDs angegeben (==0): alle Auslesen
39     $sql .= "WHERE ";
40
41     //alle IDs aus Array mit OR verknüpfen (im WHERE-Teil des Statements)
42     foreach ($id as $key => $value) {
43         if ($key != 0) { //nicht das erste Mal
44             $sql .= " OR ";
45         }
46         $sql .= "id=?";
47     }
48 }
49
50 // $sql fertig zusammengebastelt
51
52 //ausführen
53 // $id ist bereits ein Array
54 $ausgelesen = select($tabelleName, $sql, $id);
55
56 //ohne JSON wird "Array" zurückgegeben
57 //mit json_encode: Array von JS-Objects
58 print json_encode($ausgelesen);
59
60 ?>

```

### 1.3. auslesenTisch.php

```

1  <?php
2  /*
3   * Datei: auslesenTisch.php
4   * Projekt: Kellner App
5   * Autor: Michael Heider
6   * Datum: HS 2017
7   *
8   * Was:
9   * Bestellungen von einem Tisch auslesen (via Parameter)
10  * Rückgabe: Array von Arrays (Zeilen) mit Spaltennamen als Indece oder Fehlermeldung
11  */
12
13
14 //SQL_prepared.php einbinden
15 require "SQLprepared.php";
16
17 //AUSLESEN
18 $tabelleName = $_REQUEST["eventName"];
19
20 $tisch = $_REQUEST["tisch"];
21
22
23 //FAKE-AUSLESEN
24 /*echo "ERROR_Fake-Auslesen aktiviert<br>";
25 $tabelleName = "Event05";
26 $tisch = 6;
27 */
28
29
30 //Array erstellen
31 $where = array($tisch); //Array mit nur einem Element
32
33 //SQL vorbereiten
34 $sql = "SELECT id, zeit, tisch, verarbeitet, ausgeliefert, "
35     . "bestelltVon, bezahlt, storno, bezahltVon, bestellt "
36     . "FROM $tabelleName WHERE tisch = ?";
37
38 //ausführen
39 $ausgelesen = select($tabelleName, $sql, $where);
40
41 //ohne JSON wird "Array" zurückgegeben
42 //mit json_encode: Array von JS-Objects
43 print json_encode($ausgelesen);
44
45 ?>

```

### 1.4. auslesenUnverarbeitet.php

```

1  <?php
2  /*
3   * Datei: auslesenUnverarbeitet.php
4   * Projekt: Kellner App
5   * Autor: Michael Heider
6   * Datum: HS 2017
7   *
8   * Was:
9   * Bestellungen auslesen, bei denen miin 1 bestimmte Kategorie ($zeigeKat)
10  * noch nicht verarbeitet ist (via Parameter)
11  * Rückgabe: Array von Arrays (Zeilen) mit Spaltennamen als Indece oder Fehlermeldung
12  */
13
14
15 //SQL_prepared.php einbinden
16 require "SQLprepared.php";
17
18 //AUSLESEN
19 $tabelleName = $_REQUEST["eventName"];
20
21 $zeigeKat = (array)$_REQUEST["zeigeKat"]; //Array zu prüfender Kategorien-IDs
22 //prüft, ob ALLE gegebenen unverarbeitet sind

```

```

23 //falls leer: alle Kategorien unverarbeitet
24
25
26
27 //FAKE-AUSLESEN
28 /*echo "ERROR_Fake-Auslesen aktiviert<br>";
29 $tabelleName = "Event05";
30 $zeigeKat = [0]; //Array zu prüfender Kategorien-IDs
31 */
32
33
34 //Arrays erstellen
35 $katBin = 0;
36 foreach ($zeigeKat as $key => $value) {
37     $katBin += 2 ** $value;
38 }
39 unset($value); //letztes $value entfernen
40
41 $where = [$katBin, $katBin];
42 //$where = [$katBin]; //Alternative
43
44 //SQL vorbereiten
45 $sql = "SELECT id, zeit, tisch, verarbeitet, ausgeliefert, "
46     . "bestelltVon, bezahlt, storno, bezahltVon, bestellt FROM $tabelleName "
47     . "WHERE ((verarbeitet & ?) <> ?)";
48     //. "WHERE ((verarbeitet | ?) - verarbeitet) <> 0)"; //Alternative
49
50 //ausführen
51 $ausgelesen = select($tabelleName, $sql, $where);
52
53 //ohne JSON wird "Array" zurückgegeben
54 //mit json_encode: Array von JS-Objects
55 print json_encode($ausgelesen);
56
57 ?>
    
```

## 1.5. eventErstellen.php

```

1 <?php
2 /*
3  * Datei: eventErstellen.php
4  * Projekt: Kellner App
5  * Autor: Michael Heider
6  * Datum: HS 2017
7  *
8  * Was:
9  * neue Tabelle erstellen in DB
10 * Hier eingeben: Tabellennamen, Anz. Kategorien
11 * Rückgabe: "OK" oder Fehlermeldung
12 */
13
14
15 //SQL.php einbinden
16 require "SQL.php";
17
18 //SETUP
19 $tabelleName = "Event09"; //Name der Tabelle in der DB
20 $anzKategorien = 4; //wie viele Bits 'verarbeitet' und 'ausgeliefert' haben müssen
21
22 //SQL
23 $sql = "CREATE TABLE $tabelleName (
24 id MEDIUMINT UNSIGNED UNIQUE AUTO_INCREMENT PRIMARY KEY,
25 zeitstempel TIMESTAMP,
26 zeit TIME NOT NULL,
27 tisch SMALLINT UNSIGNED NOT NULL,
28 verarbeitet BIT($anzKategorien) NOT NULL DEFAULT 0,
29 ausgeliefert BIT($anzKategorien) NOT NULL DEFAULT 0,
30 bestelltVon VARCHAR(15) NOT NULL,
31 bezahltVon VARCHAR(15) DEFAULT null,
32 bestellt TEXT NOT NULL,
33 bezahlt TEXT NOT NULL,
34 storno TEXT NOT NULL
35 )
36 CHARACTER SET 'utf8';
37 //bestelltVon, bezahltVon: gegebene Maxlänge
38 //zeitstempel: zuletzt geändert (yyyy:mm:dd hh:mm:ss)
39
40 //SQL ausführen & Status zurückgeben
41 $status = sql($sql);
42 echo $status;
43
44 ?>
    
```

## 1.6. eventExistiert.php

```

1 <?php
2 /*
3  * Datei: eventExistiert.php
4  * Projekt: Kellner App
5  * Autor: Michael Heider
6  * Datum: HS 2017
7  *
8  * Was:
9  * prüfen, ob eine Tabelle in der verwendeten Datenbank existiert
10 * Rückgabe: true oder false oder Fehler (in SQL)
11 */
12
    
```

```

13
14 //SQL_prepared.php einbinden
15 require "SQLprepared.php";
16
17 //AUSLESEN
18 $tabelleName = $_REQUEST["eventName"];
19
20
21
22 //FAKE-AUSLESEN
23 /*echo "ERROR_Fake-Auslesen aktiviert";
24 $tabelleName = "Event04";
25 */
26
27
28 //Array erstellen
29 $where = array($tabelleName);
30
31 //SQL vorbereiten
32 //prüft ob Tabelle mit Name $tabelleName in verwendeten Datenbank existiert
33 $sql = "SELECT *
34 FROM information_schema.tables
35 WHERE table_schema = DATABASE()
36 AND table_name = ?";
37
38 //ausführen
39 //ACHTUNG: hier kann ein Fehler produziert werden
40 $ausgelesen = select($tabelleName, $sql, $where);
41
42 //falls min 1 Zeile ausgelesen: Tabelle existiert
43 if (gettype($ausgelesen) == "array" && count($ausgelesen) > 0) {
44     echo "true";
45 } else {
46     echo "false";
47 }
48
49 ?>
    
```

## 1.7. insertBestellung.php

```

1 <?php
2 /*
3  * Datei: insertBestellung.php
4  * Projekt: Kellner App
5  * Autor: Michael Heider
6  * Datum: HS 2017
7  *
8  * Was:
9  * neue Bestellung erstellen (via Parameter)
10 * Rückgabe: "OK" oder Fehlermeldung
11 *
12 * ANMERKUNG: verwendet Länge des bestellt-Array zur Initialisierung des
13 * bezahlt-Arrays, da TEXT kein Default haben kann in DB
14 */
15
16
17 //SQL_prepared.php einbinden
18 require "SQLprepared.php";
19
20 //AUSLESEN
21 $tabelleName = $_REQUEST["eventName"];
22
23 $zeit = $_REQUEST["zeit"];
24 $tisch = $_REQUEST["tisch"];
25 $bestellt = $_REQUEST["bestellt"];
26 $bestelltVon = $_REQUEST["bestelltVon"]; //max 10 Zeichen, Text
27 //ANMERKUNG: verarbeitet wird von Basen angepasst
28
29
30
31 //FAKE-AUSLESEN
32 /*echo "ERROR_Fake-Auslesen aktiviert";
33 $tabelleName = "Event07";
34 $zeit = "01:46";
35 $tisch = 12;
36 $bestellt = "[1,15,7,0,6,3,12]";
37 $bestelltVon = "abc012"; //max 10 Zeichen, Text
38 */
39
40
41 //bezahlt Array initialisieren aufgrund von Länge von bestellt-Array
42 $bezahlt = array_fill(0, count(json_decode($bestellt)), 0); //zur Initialisierung
43 // (Text Data-Type kann kein Default haben in DB)
44 $bezahlt = json_encode($bezahlt);
45
46 //zu bindende Variablen festlegen
47 $variablen = array($zeit, $tisch, $bestellt, $bestelltVon, $bezahlt);
48
49 //SQL vorbereiten
50 $sql = "INSERT INTO $tabelleName
51 (zeit, tisch, bestellt, bestelltVon, bezahlt)
52 VALUES (?, ?, ?, ?, ?)";
53
54 //ausführen
55 $status = insert($tabelleName, $sql, $variablen);
56
57 echo $status;
58
59 ?>
    
```

## 1.8. loginDaten.php

```

1  <?php
2  /*
3   * Datei: loginDaten.php
4   * Projekt: Kellner App
5   * Autor: Michael Heider
6   * Datum: HS 2017
7   *
8   * Was:
9   * Zugangsdaten Server & Datenbank:
10  * Servername, Benutzername, Passwort, Name der DB: Variablenamen siehe Code
11  */
12
13
14  //ZUGANGSDATEN SERVER & DATENBANK
15  $dbname = "kellnerapp";
16
17  $servername = "localhost";
18  $username = "kellnerapp";
19  $password = "kellnerappDB-8-1";
20  ?>

```

## 1.9. SQL.php

```

1  <?php
2  /*
3   * Datei: SQL.php
4   * Projekt: Kellner App
5   * Autor: Michael Heider
6   * Datum: HS 2017
7   *
8   * Was:
9   * Funktion um SQL auszuführen.
10  * Erstellt und schliesst Verbindung.
11  */
12
13
14  //sql: SQL-query, $multi: ob mehrere queries auf einmal übertragen werden können
15  function sql($sql, $multi = false) {
16      //ZUGANGSDATEN SERVER & DATENBANK: //dbLogin.php einbinden
17      require "loginDaten.php";
18
19      //Verbindung herstellen und überprüfen
20      $verbindung = new mysqli($servername, $username, $password, $dbname);
21      if (!$verbindung) {
22          return "ERROR_Verbindung fehlgeschlagen_" . $verbindung->connect_errno .
23              "_" . $verbindung->connect_error;
24      }
25
26      //SQL ausführen
27      if ($multi) {
28          if (!$verbindung->multi_query($sql)) {
29              return "ERROR_" . $verbindung->error . "_" . $sql;
30          }
31      } else {
32          if (!$verbindung->query($sql)) {
33              return "ERROR_" . $verbindung->error . "_" . $sql;
34          }
35      }
36
37
38
39      //Verbindung schliessen
40      $verbindung->close(); //Verbindung schliessen
41
42      return "OK";
43  }
44
45  ?>

```

## 1.10. SQLprepared.php

```

1  <?php
2  /*
3   * Datei: SQLprepared.php
4   * Projekt: Kellner App
5   * Autor: Michael Heider
6   * Datum: HS 2017
7   *
8   * Was:
9   * SQL via Parameter
10  * dadurch SQL-Injection sicher
11  * enthält Funktionen für: select, update, insert
12  */
13
14
15  //SELECT, Eintrag/Einträge auslesen
16  //$tabelleName: Name der DB-Tabelle, $sql: SQL-Query,
17  //$where: array mit zu suchenden Spaltenwerten für WHERE (kann leer sein),
18  //return: Array von Arrays (Zeilen) mit Spaltennamen als Indec oder Fehlermeldung
19  function select($tabelleName, $sql, $where = array()) {
20      //ZUGANGSDATEN SERVER & DATENBANK: //dbLogin.php einbinden
21      require "loginDaten.php";
22
23      //Typen der $where ermitteln

```

```

24     $typen = typen($where);
25
26     //Verbindung herstellen und überprüfen
27     $verbindung = new mysqli($servername, $username, $password, $dbname);
28     if (!$verbindung) {
29         return "ERROR_Verbindung fehlgeschlagen" . $verbindung->connect_errno .
30             " " . $verbindung->connect_error;
31     }
32
33     //Anweisung erstellen
34     $statement = $verbindung->prepare($sql);
35     if (!$statement) {
36         return "ERROR_" . $verbindung->error . " " . $sql; //braucht $verbindung
37     }
38
39     if (count($where) != 0) { //keine Parameter binden, wenn $where leer ist
40         //Statement und Typen vorhängen
41         array_unshift($where, $statement, $typen);
42
43         //referenzen() verwenden, da mysqli_stmt_bind_param keine values will
44         if (!call_user_func_array("mysqli_stmt_bind_param", referenzen($where))) {
45             return "ERROR_" . $statement->error . " " . print_r($where, true);
46         }
47     }
48
49     if (!$statement->execute()) { //Anweisung ausführen
50         return "ERROR_" . $statement->error;
51     }
52
53     //Ausgelesenes bekommen
54     //AlleResultate: Array von Arrays mit Spaltennamen als Indice
55     $alleResultate = array();
56     $resultat = $statement->get_result();
57     //Zeile: Array mit Spaltennamen als Index (eine ausgelesene Zeile)
58     while ($zeile = $resultat->fetch_assoc()) { //für alle Zeilen
59         array_push($alleResultate, $zeile);
60     }
61
62     $statement->close(); //Anweisung schliessen
63     $verbindung->close(); //Verbindung schliessen
64
65     //Resultat zurückgeben
66     return $alleResultate;
67 }
68
69
70 //INSERT, neuer Eintrag
71 //StabelleName: Name der DB-Tabelle, $sql: SQL-Query,
72 //Variablen: array mit zu übertragenden Variablen,
73 //return: "OK" oder Fehlermeldung
74 function insert($StabelleName, $sql, $werte) {
75     //ZUGANGSDATEN SERVER & DATENBANK: //dbLogin.php einbinden
76     require "loginDaten.php";
77
78     //Typen der $werte ermitteln
79     $types = typen($werte);
80
81     //Verbindung herstellen und überprüfen
82     $verbindung = new mysqli($servername, $username, $password, $dbname);
83     if (!$verbindung) {
84         return "ERROR_Verbindung fehlgeschlagen" . $verbindung->connect_errno .
85             " " . $verbindung->connect_error;
86     }
87
88     //Anweisung erstellen
89     $statement = $verbindung->prepare($sql);
90     if (!$statement) {
91         return "ERROR_" . $verbindung->error . " " . $sql; //braucht $verbindung
92     }
93
94     //Variablen zuweisen
95     //dem $werte auch das $statement und die $types vorhängen
96     array_unshift($werte, $statement, $types);
97
98     //referenzen() verwenden, da mysqli_stmt_bind_param keine values will
99     if (!call_user_func_array("mysqli_stmt_bind_param", referenzen($werte))) {
100         return "ERROR_" . $statement->error . " " . print_r($werte, true);
101     }
102
103     if (!$statement->execute()) { //Anweisung ausführen
104         return "ERROR_" . $statement->error;
105     }
106
107     $statement->close(); //Anweisung schliessen
108     $verbindung->close(); //Verbindung schliessen
109     return "OK";
110 }
111
112 //UPDATE, Eintrag verändern
113 //falls kein Eintrag zu $where passt, passiert nichts
114 //falls kein Eintrag geändert wird, wird der TIMESTAMT nicht aktualisiert (pro Zeile)
115 //StabelleName: Name der DB-Tabelle, $sql: SQL-Query,
116 //StabelleName: Name der DB-Tabelle, $sql: SQL-Query,
117 //StabelleName: Name der DB-Tabelle, $sql: SQL-Query,
118 //StabelleName: Name der DB-Tabelle, $sql: SQL-Query,
119 //StabelleName: Name der DB-Tabelle, $sql: SQL-Query,
120 //StabelleName: Name der DB-Tabelle, $sql: SQL-Query,
121 //StabelleName: Name der DB-Tabelle, $sql: SQL-Query,
122 //StabelleName: Name der DB-Tabelle, $sql: SQL-Query,
123 //StabelleName: Name der DB-Tabelle, $sql: SQL-Query,
124 //StabelleName: Name der DB-Tabelle, $sql: SQL-Query,
125 //StabelleName: Name der DB-Tabelle, $sql: SQL-Query,
126 //StabelleName: Name der DB-Tabelle, $sql: SQL-Query,
127 //StabelleName: Name der DB-Tabelle, $sql: SQL-Query,
128 //StabelleName: Name der DB-Tabelle, $sql: SQL-Query,

```

```

129     if (!$verbindung) {
130         return "ERROR_Verbindung_fehlgeschlagen_" . $verbindung->connect_errno .
131             "_" . $verbindung->connect_error;
132     }
133
134     //Anweisung erstellen
135     $statement = $verbindung->prepare($sql);
136     if(!$statement) {
137         return "ERROR_" . $verbindung->error . "_" . $sql; //braucht $verbindung
138     }
139
140     //werte und where zusammenhängen (zuerst $werte, dann $where)
141     $parameter = array_merge($werte, $where); //$where kann auch leer sein
142     $stypenAlle = $stypenWerte . $stypenWhere;
143     //Statement und Typen vorhängen
144     array_unshift($parameter, $statement, $stypenAlle);
145
146     //referenzen() verwenden, da mysqli_stmt_bind_param keine values will
147     if(!call_user_func_array("mysqli_stmt_bind_param", referenzen($parameter))) {
148         return "ERROR_" . $statement->error . "_" . print_r($parameter, true);
149     }
150
151     if(!$statement->execute()) { //Anweisung ausführen
152         return "ERROR_" . $statement->error;
153     }
154
155     //Anz. Zeilen, in denen tatsächlich etwas verändert wurde
156     $anzZeilen = $statement->affected_rows;
157
158     $statement->close(); //Anweisung schliessen
159     $verbindung->close(); //Verbindung schliessen
160     return $anzZeilen;
161 }
162
163
164 //Typen der Werte in Array als String ermitteln
165 function typen($array) {
166     $stypen = "";
167     foreach ($array as $key => $value) {
168         switch (gettype($value)) {
169             case "integer":
170                 $stypen .= "i";
171                 break;
172
173             case "double": //double oder float
174                 $stypen .= "d";
175                 break;
176
177             default:
178                 $stypen .= "s";
179                 break;
180         }
181     }
182     unset($value); //letztes $value entfernen
183     return $stypen;
184 }
185
186 //gibt einen Array mit Referenzen zu ursprünglichen Elementen zurück
187 //für mysqli_stmt_bind_param
188 function referenzen($array) {
189     $referenzen = array();
190     foreach ($array as $key => $value) {
191         $referenzen[$key] = &$array[$key];
192     }
193     unset($value); //letztes $value entfernen
194     return $referenzen;
195 }
196
197 ?>

```

## 1.11. tagesUbersicht.php

```

1  <?php
2  /*
3   * Datei: tagesUbersicht.php
4   * Projekt: Kellner App
5   * Autor: Michael Heider
6   * Datum: HS 2017
7   *
8   * Was:
9   * alle Bestellungen auslesen (via Parameter)
10  * zurückgeben in HTML-Format und Datei erstellen in definiertem Format
11  * Datei wird in selben Verzeichnis wie diese php-Datei erstellt
12  *
13  * ACHTUNG: falls Datei schon vorhanden, wird sie überschrieben
14  * ACHTUNG: liest Produkte Datei aus, URL gegeben relativ zu dieser php-Datei
15  */
16
17
18 //SQL_prepared.php einbinden
19 require "SQLprepared.php";
20
21
22 // ===== SETUP =====
23 //auszulesendes Event: Name
24 $tabelleName = "Event08";
25
26 //Separatoren (Zeichen zwischen zwei Spalten bzw. Zeilen)
27 //für html: $col = ", "; $row = "<br>";
28 //für txt: $col = ", "; $row = "\n";
29 //für MS Excel CSV: $col = ","; $row = "\n";

```

```

30 $col = ",";
31 $row = "\n";
32
33
34 // ===== AUSLESEN =====
35 //$tabelleName wird gesetzt unter Setup
36
37 //SQL vorbereiten
38 $sql = "SELECT id, zeit, tisch, verarbeitet, ausgeliefert, "
39       . "bestelltVon, bezahlt, storno, bezahltVon, bestellt FROM $tabelleName";
40 //kein WHERE Teil, da alle ausgelesen werden sollen
41
42 //SQL ausführen
43 $ausgelesen = select($tabelleName, $sql);
44
45
46 // ===== AUSWERTEN =====
47 //Arrays: $alleBestellt, $alleBezahlt, $alleStorno
48 //abbrechen, falls keine Bestellungen
49 if (count($ausgelesen) < 1) {
50     print "keine Bestellungen";
51     die();
52 }
53
54 //Produkte-Datei auslesen und konvertieren
55 //--> $produktum $anzProdukte
56 $produkteStr = file_get_contents("../eventData/produkte.json");
57 $produkte = json_decode($produkteStr);
58 $anzProdukte = count($produkte);
59 //innere Objekte zu assoziativen Arrays konvertieren
60 for ($i = 0; $i < $anzProdukte; $i++) {
61     $produkte[$i] = (array)$produkte[$i];
62 }
63
64 //Total-Arrays initialisieren
65 $alleBestellt = array_fill(0, $anzProdukte, 0);
66 $alleBezahlt = array_fill(0, $anzProdukte, 0); //unten wird storno-Array abgezogen
67 $alleStorno = array_fill(0, $anzProdukte, 0);
68
69 //für jede Bestellung
70 for ($i = 0, $len = count($ausgelesen); $i < $len; $i++) {
71     $bestellung = $ausgelesen[$i];
72
73     //JSON-Arrays in php-Arrays umwandeln
74     $bestellung["bestellt"] = json_decode($bestellung["bestellt"]);
75     $bestellung["bezahlt"] = json_decode($bestellung["bezahlt"]);
76     $bestellung["storno"] = json_decode($bestellung["storno"]);
77
78     //bestellt, bezahlt, storno jeweils zu Total hinzuzählen
79     for ($k = 0; $k < $anzProdukte; $k++) {
80         $alleBestellt[$k] += $bestellung["bestellt"][$k];
81         $alleBezahlt[$k] += $bestellung["bezahlt"][$k];
82         $alleStorno[$k] += $bestellung["storno"][$k]; //merkt automatisch, wenn leer
83     }
84 }
85 //storno-Array von bezahlt-Array abziehen
86 for ($i = 0; $i < $anzProdukte; $i++) {
87     $alleBezahlt[$i] -= $alleStorno[$i];
88 }
89
90
91 // ===== AUSGEBEN =====
92 //Separatoren (Zeichen zwischen zwei Spalten bzw. Zeilen) wurden am Anfang definiert
93 //$col und $row
94
95 $ausgabe = "";
96
97 //Titelzeile ausgeben
98 //ACHTUNG: KANN NICHT MIT "ID" (GROSSSCHREIBUNG) ANFANGEN FÜR MS-EXCEL-CSV
99 $ausgabe = $ausgabe . "id{$col}Name{$col}Stueckpreis{$col}bestellt{$col}bezahlt{$col}storniert";
100 $ausgabe = $ausgabe . $row;
101
102 print "id, Name, Stueckpreis, bestellt, bezahlt, storniert";
103 print "<br>";
104
105 //für jedes Produkt eine Zeile ausgeben
106 for ($i = 0; $i < $anzProdukte; $i++) {
107     $ausgabe = $ausgabe . $produkte[$i]["id"] . $col . $produkte[$i]["name"] . $col .
108               preisAnzeige($produkte[$i]["preis"]) . $col .
109               $alleBestellt[$i] . $col . $alleBezahlt[$i] . $col . $alleStorno[$i];
110     $ausgabe = $ausgabe . $row;
111
112     print $produkte[$i]["id"] . ", " . $produkte[$i]["name"] . ", " .
113           preisAnzeige($produkte[$i]["preis"]) . ", " .
114           $alleBestellt[$i] . ", " . $alleBezahlt[$i] . ", " . $alleStorno[$i];
115     print "<br>";
116 }
117
118 print "<br>";
119 print "ausgabe: " . $ausgabe;
120
121 //In Datei speichern
122 //ACHTUNG: ÜBERSCHREIBT URSPRÜNGLICHE DATEI
123 file_put_contents("tagesÜbersicht.csv", $ausgabe);
124
125
126 // ===== FUNKTIONEN =====
127 // =====
128 // =====
129
130 //Preis mit 2 Nachkommastellen, '.' als Dezimalzeichen, '\\' als Tausenderzeichen
131 function preisAnzeige($preis) {
132     return number_format($preis, 2, '.', '\\');
133 }
134

```

135 ?&gt;

## 1.12. updaten.php

```

1  <?php
2  /*
3   * Datei: updaten.php
4   * Projekt: Kellner App
5   * Autor: Michael Heider
6   * Datum: HS 2017
7   *
8   * Was:
9   * Bestellung-Daten updaten (via Parameter)
10  * Rückgabe: Anz. Zeilen in denen tatsächlich etwas geändert wurde oder Fehlermeldung
11  */
12
13
14  //SQL_prepared.php einbinden
15  require "SQLprepared.php";
16
17  // AUSLESEN
18  $tabelleName = $_REQUEST["eventName"];
19
20  //WHERE
21  $id = (array)$_REQUEST["id"]; //Array von IDs
22
23  //WERTE
24  $bezahlt = $_REQUEST["bezahlt"];
25  $storno = $_REQUEST["storno"];
26  $bezahltVon = $_REQUEST["bezahltVon"]; //max 10 Zeichen, Text
27  $verarbeitet = (array)$_REQUEST["verarbeitet"]; //Array von zu setzenden Kategorien
28  $ausgeliefert = (array)$_REQUEST["ausgeliefert"]; //Array von zu setzenden Kategorien
29  $tisch = $_REQUEST["tisch"]; //nur falls spätere Korrektur
30  $bestellt = $_REQUEST["bestellt"]; //nur falls spätere Korrektur
31
32  //Anmerkungen:
33  //id: Array von IDs (wird automatisch zu php-Array konvertiert)
34  //falls id nicht mehr existiert oder null: nichts passiert, auch keinen Fehler
35  //$verarbeitet ist Array von zu setzenden Kategorien
36  //$ausgeliefert: analog zu verarbeitet
37  //$tisch und $bestellt sollten eigentlich nicht mehr geändert werden
38  //uhrzeit und bestelltVon können nicht geändert werden
39
40
41
42  //FAKE-AUSLESEN
43  /*echo "ERROR Fake-Auslesen aktiviert";
44  $tabelleName = "Event05";
45  $id = [4]; //Array von IDs
46  //$bezahlt = 1;
47  //$bezahltVon = "xyz789"; //max 10 Zeichen, Text
48  //$verarbeitet = [0];
49  //$ausgeliefert = [0];
50  //$tisch = 7;
51  //$bestellt = "[1, 2, 3, 65, 4, 39]";
52  */
53
54
55  //Arrays erstellen
56  $variablen = array();
57  $where = $id; //ist bereits ein Array
58  if (count($id) == 0) { //nichts zum updaten (wäre unnötig und gäbe SQL Parse-Error)
59      echo "0"; //Null Zeilen bearbeitet zurückgeben
60      die(); //ABBRECHEN: SQL-Query nicht ausführen
61  }
62
63
64  //SQL vorbereiten und $variablen-Array füllen
65  //$sql fertig zusammengestellt: (verarbeitet und ausgeliefert Teile speziell
66  //Stil von: "UPDATE $tabelleName SET bezahlt=?, bezahltVon=? ... WHERE id=? OR id=? ..."
67  $sql = "UPDATE $tabelleName SET "; //Anfang
68
69  $erstes = true; //erste Spalte hat noch kein vorgestelltes Komma
70
71  /*Prinzip:
72  if ($name !== null) {
73      if (!$erstes) { //Komma beim ersten weglassen
74          $sql .= ",";
75      }
76      $erstes = false; //nächstes ist nicht mehr erstes
77      $sql .= " name=?"; //Spaltenname (name) an $sql anhängen
78      $variablen[] = $name; //name an $variablen-Array anhängen
79  }*/
80  if ($bezahlt !== null) {
81      if (!$erstes) {
82          $sql .= ",";
83      }
84      $erstes = false;
85      $sql .= " bezahlt=?";
86      $variablen[] = $bezahlt;
87  }
88  if ($storno !== null) {
89      if (!$erstes) {
90          $sql .= ",";
91      }
92      $erstes = false;
93      $sql .= " storno=?";
94      $variablen[] = $storno;
95  }
96  if ($bezahltVon !== null) {
97      if (!$erstes) {

```

```

98     $sql .= ",";
99 }
100 $erstes = false;
101 $sql .= " bezahltVon?";
102 $variablen[] = $bezahltVon;
103 }
104 if ($tisch !== null) {
105     if (!$erstes) {
106         $sql .= ",";
107     }
108     $erstes = false;
109     $sql .= " tisch?";
110     $variablen[] = $tisch;
111 }
112 if ($bestellt !== null) {
113     if (!$erstes) {
114         $sql .= ",";
115     }
116     $erstes = false;
117     $sql .= " bestellt?";
118     $variablen[] = $bestellt;
119 }
120 /*Prinzip:
121 if ($verarbeitet !== null) {
122     if (count($verarbeitet) > 0) { //nur falls überhaupt etwas in Array
123         if (!$erstes) { //Komma beim ersten weglassen
124             $sql .= ",";
125         }
126         $erstes = false;
127         $katBin = 0; //binär-Zahl der Kategorien berechnen
128         foreach ($verarbeitet as $key => $value) {
129             $katBin += 2 ** $value;
130         }
131         unset($value); //letztes $value entfernen
132         $sql .= " verarbeitet=(verarbeitet | ?)"; //Array in DB updaten
133         //(Binär Logik: gegebene Kategorien auf 1 setzten, andere unverändert)
134         $variablen[] = $katBin; //entsprechender Wert an $variablen anhängen
135     }
136 }*/
137 if ($verarbeitet !== null) {
138     if (count($verarbeitet) > 0) {
139         if (!$erstes) {
140             $sql .= ",";
141         }
142         $erstes = false;
143         $katBin = 0;
144         foreach ($verarbeitet as $key => $value) {
145             $katBin += 2 ** $value;
146         }
147         unset($value);
148         $sql .= " verarbeitet=(verarbeitet | ?)";
149         $variablen[] = $katBin;
150     }
151 }
152 if ($ausgeliefert !== null) {
153     if (count($ausgeliefert) > 0) {
154         if (!$erstes) {
155             $sql .= ",";
156         }
157         $erstes = false;
158         $katBin = 0;
159         foreach ($ausgeliefert as $key => $value) {
160             $katBin += 2 ** $value;
161         }
162         unset($value);
163         $sql .= " ausgeliefert=(ausgeliefert | ?)";
164         $variablen[] = $katBin;
165     }
166 }
167
168 if ($erstes) { //nichts zum updaten (wäre unnötig und gäbe SQL-Query Parse-Error)
169     echo "0"; //Null Zeilen bearbeitet zurückgeben
170     die(); //ABBRECHEN: SQL-Query nicht ausführen
171 }
172
173 $sql .= " WHERE "; //where anfügen
174
175 //alle IDs aus Array mit OR verknüpfen (im WHERE-Teil des Statements)
176 foreach ($id as $key => $value) {
177     if ($key != 0) { //nicht das erste Mal
178         $sql .= " OR ";
179     }
180     $sql .= "id=?";
181 }
182
183 // $sql fertig zusammengestellt (Bsp. siehe oben)
184
185 //ausführen
186 //gibt Anz. Zeilen in denen tatsächlich etwas geändert wurde oder Fehlermeldung
187 $status = update($tabelleName, $sql, $variablen, $where);
188
189 echo $status;
190
191 ?>

```

## 2. eventData

Für jeden Anlass werden die Event Daten individuell angepasst. Hier abgedruckt ist das Beispiel, welches auch für die Abbildungen in dieser Arbeit verwendet worden ist. Die Speisekarte stammt vom Waldbeizverein der Chilbi Lindau.

### 2.1. eventData.json

```
1 {
2   "name": "Waldbeiz"
3 }
```

### 2.2. kategorien.json

```
1 [
2   {
3     "id": 0,
4     "name": "Essen"
5   },
6   {
7     "id": 1,
8     "name": "Dessert"
9   },
10  {
11    "id": 2,
12    "name": "Getränke: Wein, Bier, Most"
13  },
14  {
15    "id": 3,
16    "name": "Getränke: Mineral, Kaffee, Tee"
17  }
18 ]
```

### 2.3. produkte.json

```
1 [
2   {
3     "id": 0,
4     "name": "Servelat mit Brot",
5     "preis": 4,
6     "kategorie": 0
7   },
8   {
9     "id": 1,
10    "name": "Bratwurst mit Brot",
11    "preis": 5,
12    "kategorie": 0
13  },
14  {
15    "id": 2,
16    "name": "Waldbeizwurst mit Brot",
17    "preis": 6,
18    "kategorie": 0
19  },
20  {
21    "id": 3,
22    "name": "Jägerschnitzel mit Brot",
23    "preis": 8.5,
24    "kategorie": 0
25  },
26  {
27    "id": 4,
28    "name": "Poulet mit Brot",
29    "preis": 10,
30    "kategorie": 0
31  },
32  {
33    "id": 5,
34    "name": "Pommes",
35    "preis": 5,
36    "kategorie": 0
37  },
38  {
39    "id": 6,
40    "name": "Crèmeschnitte",
41    "preis": 3.5,
42    "kategorie": 1
43  },
44  {
45    "id": 7,
46    "name": "Mehlsuppe",
47    "preis": 4,
48    "kategorie": 1
49  },
50  {
51    "id": 8,
52    "name": "Allaman rouge",
53    "preis": 15,
54    "kategorie": 2
55  },
56  {
57    "id": 9,
58    "name": "Waldbeizwein",
59    "preis": 15,
60    "kategorie": 2
61  },
62  {
63    "id": 10,
64    "name": "Goldtröpfli",
65    "preis": 15,
66    "kategorie": 2
67  },
68  {
69    "id": 11,
70    "name": "Nürensdorfer",
71    "preis": 15,
72    "kategorie": 2
73  },
74  {
75    "id": 12,
76    "name": "Quöllfrisch",
77    "preis": 4.5,
78    "kategorie": 2
79  },
80  {
81    "id": 13,
82    "name": "Holzfassbier",
83    "preis": 4,
84    "kategorie": 2
85  },
86  {
87    "id": 14,
88    "name": "Feldschlösschen o.A.",
89    "preis": 4,
90    "kategorie": 2
91  },
92  {
93    "id": 15,
94    "name": "Saurer Most",
95    "preis": 4.5,
96    "kategorie": 2
97  },
```

```

98     {
99       "id": 16,
100      "name": "Saurer Most o.A.",
101      "preis": 4.5,
102      "kategorie": 2
103    },
104    {
105      "id": 17,
106      "name": "Eve Lichi",
107      "preis": 4,
108      "kategorie": 2
109    },
110    {
111      "id": 18,
112      "name": "Coca Cola",
113      "preis": 3,
114      "kategorie": 3
115    },
116    {
117      "id": 19,
118      "name": "Sprite",
119      "preis": 3,
120      "kategorie": 3
121    },
122    {
123      "id": 20,
124      "name": "Sinalco",
125      "preis": 3,
126      "kategorie": 3
127    },
128    {
129      "id": 21,
130      "name": "Rivella rot",
131      "preis": 3,
132      "kategorie": 3
133    },
134    {
135      "id": 22,
136      "name": "Rivella blau",
137      "preis": 3,
138      "kategorie": 3
139    },
140    {
141      "id": 23,
142      "name": "Henniez",
143      "preis": 3,
144      "kategorie": 3
145    },
146    {
147      "id": 24,
148      "name": "Ice Tea",
149      "preis": 3,
150      "kategorie": 3
151    },
152    {
153      "id": 25,
154      "name": "Kaffee Crème",
155      "preis": 3,
156      "kategorie": 3
157    },
158    {
159      "id": 26,
160      "name": "Kaffee Lutz",
161      "preis": 5,
162      "kategorie": 3
163    },
164    {
165      "id": 27,
166      "name": "Schwarztee",
167      "preis": 2,
168      "kategorie": 3
169    },
170    {
171      "id": 28,
172      "name": "Pfefferminztee",
173      "preis": 2,
174      "kategorie": 3
175    },
176    {
177      "id": 29,
178      "name": "Lindenblütentee",
179      "preis": 2,
180      "kategorie": 3
181    },
182    {
183      "id": 30,
184      "name": "Hagenbuttertee",
185      "preis": 2,
186      "kategorie": 3
187    }
188  ]

```

## 2.4. tische.json

```

1  [
2    {
3      "id": 0,
4      "name": "Inn 1"
5    },
6    {
7      "id": 1,
8      "name": "Inn 2"
9    },
10   {
11     "id": 2,
12     "name": "Inn 3"
13   },
14   {
15     "id": 3,
16     "name": "Inn 4"
17   },
18   {
19     "id": 4,
20     "name": "Inn 5"
21   },
22   {
23     "id": 5,
24     "name": "Aus 1"
25   },
26   {
27     "id": 6,
28     "name": "Aus 2"
29   },
30   {
31     "id": 7,
32     "name": "Aus 3"
33   },
34   {
35     "id": 8,
36     "name": "Aus 4"
37   },
38   {
39     "id": 9,
40     "name": "Aus 5"
41   }
42 ]

```

### 3. scripts

#### 3.1. allgemeineFunktionen.js

```

1  /*
2  * Datei: allgemeineFunktionen.js
3  * Projekt: Kellner App
4  * Autor: Michael Heider
5  * Datum: HS 2017
6  *
7  * Was:
8  * Konstruktoren, Kommunikation, Technisches
9  * wird verwendet von allen Teilen (Satellit, Basis, Sprinter)
10 * */
11
12 "use strict"; //strict mode aktivieren
13
14
15 // =====
16 // ===== KONSTRUKTOREN =====
17 // =====
18
19 // ..... BESTELLUNGEN .....
20 //Konstruktor für Bestellungen-Objekte
21 function ErstelleBestellung(tisch_, bestellt_, kellner_, id_/* = undefined*/) {
22     if (id_ === undefined) {
23         id_ = undefined;
24     }
25
26     this.id = id_; //wird in DB automatisch zugewiesen
27     this.zeit = uhrzeit(); //Zeit (hh:mm:ss) der Bestellungen-Aufnahme
28     this.tisch = tisch_; //Tisch-ID zu der die Bestellung gehört
29
30     this.bestelltVon = kellner_; //welcher Kellner die Bestellung aufgenommen hat
31
32     this.bezahlt = new Array(produkte.length); //wie viel von jedem Produkt bezahlt wurde
33     this.bezahlt.fill(0); // (kann auch mehr sein als bestellt)
34     this.storno = undefined; //wie viel von jedem Produkt storniert wurde,
35     // //sofern überhaupt etwas storniert wurde, sonst undefined
36     // // (kann auch mehr sein als bestellt)
37     //welcher Kellner die Bestellung einkassiert hat/storniert hat
38     this.bezahltVon = undefined;
39
40     //verarbeitet-, ausgeliefert-Status
41     //ACHTUNG: bei verarbeitet und ausgeliefert Reihenfolge verkehrt:
42     //rechtstes (letztes) Zeichen ist 0-te Kategorie, zweit rechtestes 1-te usw
43     //Code für i-te Kategorie: bestellung.verarbeitet.length - 1 - i
44     this.verarbeitet = "".padStart(kategorien.length, "0"); //String gefüllt mit '0'
45     this.ausgeliefert = "".padStart(kategorien.length, "0"); //String gefüllt mit '0'
46
47     //wie viel von jedem Produkt bestellt ist (0 für keins davon bestellt)
48     //Länge entspricht produkte.length
49     this.bestellt = bestellt_;
50 }
51
52 // ..... PRODUKTE, KATEGORIEN, TISCHE .....
53 //var produkte = []; //alle Produkte. Position in array stimmt mit produkt.id überein
54 //Konstruktor für Produkt-Objekte
55 function ErstelleProdukt(id_, name_, preis_, kategorie_) {
56     this.id = id_; //ID (je nach Verwendung auch Array von IDs)
57     this.name = name_;
58     this.preis = preis_; //in Franken
59     this.kategorie = kategorie_; //zu welcher Kategorie das Produkt gehört
60 }
61 //var kategorien = []; //alle Kategorien. Position in array stimmt mit kategorie.id überein
62 //Konstruktor für Kategorie-Objekte
63 function ErstelleKategorie(id_, name_, produkte_) {
64     this.id = id_; //id stimmt mit Position in kategorien array überein!
65     this.name = name_;
66     this.produkte = produkte_; //zugeteilte Produkte
67     //werden zugeteilt aufgrund von produkt.kategorie während Initialisierung
68 }
69 //var tische = []; //alle Tische. Position in array stimmt mit tisch.id überein
70 //Konstruktor für Tisch-Objekte
71 function ErstelleTisch(id_, name_) {
72     this.id = id_; //id stimmt mit Position in tische array überein
73     this.nr = name_; //String: Tisch-Name
74 }
75
76
77 // =====
78 // ===== ANZEIGE-FORMATE =====
79 // =====
80
81 //Preis mit 2 Nachkommastellen und evt. Währung (2.5 -> Fr. 2.50)
82 function preisAnzeige(x, währung/* = true*/) {
83     if (währung === undefined) {
84         währung = true;
85     }
86
87     if (währung) {
88         return "Fr. " + x.toFixed(2);
89     } else {
90         return x.toFixed(2);
91     }
92 }
93
94 //Menge zum Anzeigen (x vor-/nachstellen)
95 //x: Menge
96 //ort: true: 'x' vorgestellt, false: 'x' nachgestellt
97 //suffix: vor/nach zu stellende Zeichen (Standard: 'x')
98 function mengeAnzeige(x, ort/* = false*/, suffix/* = 'x'*/) {

```

```

99     if (ort === undefined) {
100         ort = false;
101     }
102     if (suffix === undefined) {
103         suffix = 'x';
104     }
105
106     if (ort) {
107         return suffix + x;
108     } else {
109         return x + suffix;
110     }
111 }
112
113 //gibt aktuelle Uhrzeit als "hh:mm:ss". (Datenbank-Format: "hh:mm:ss")
114 function uhrzeit() {
115     var jetzt = new Date();
116     var stunde = jetzt.getHours();
117     if (stunde < 10) {
118         stunde = "0" + stunde;
119     }
120     var minute = jetzt.getMinutes();
121     if (minute < 10) {
122         minute = "0" + minute;
123     }
124     var sekunde = jetzt.getSeconds();
125     if (sekunde < 10) {
126         sekunde = "0" + minute;
127     }
128     return stunde + ":" + minute + ":" + sekunde;
129 }
130
131 //Zeit zum Anzeigen als "hh:mm"
132 //zeit: String mit "hh:mm:ss"
133 function zeitAnzeige(zeit) {
134     return zeit.substring(0, 5); //nur "hh:mm" von "hh:mm:ss"
135 }
136
137
138 // =====
139 // ===== KOMMUNIKATION (DB, INITIALISIERUNG) =====
140 // =====
141
142 /* Funktionen zur Kommunikation mit:
143 * -der Datenbank (DB), via PHP-Dateien
144 * -um Dateien direkt auszulesen
145 * wird verwendet von allen Teilen (Satellit, Basis, Sprinter)
146 */
147
148 // .....
149 // ..... INITIALISIERUNG: EVENT-DATEN AUSLESEN .....
150 // .....
151
152 //Initialisierung: Event-Daten auslesen (Produkte, Kategorien, Tische, Name)
153 //auslesen und speichern, sowie Produkte den Kategorien zuordnen
154 //prüfen, ob DB existiert
155 function auslesen() {
156     //Event Daten auslesen und speichern (verzogerungEvent (siehe unten))
157     //Verzogerung für check bereits hier erstellen, nachher manuell resolve/reject
158     var verzogerungEventCheck = new $.Deferred();
159     var verzogerungEventAuslesen = textAuslesen("eventData/eventData.json", function(inhalt) {
160         var eventData = JSONtryParse(inhalt);
161         if (produkte === undefined) {
162             verzogerungEventAuslesen.reject("eventData/eventData.json_JSON");
163         }
164         //prüfen, ob gefragter Event existiert
165         var data = {
166             eventName: eventData.name
167         }
168         //var verzogerungEventCheck = new $.Deferred();
169
170         var verzogerungAbfrage = dateiAufrufen("database/eventExistiert.php", data, function(antwort) {
171             var antwortParse = JSONtryParse(antwort);
172
173             //Fehlererkennung
174             var fehler = false;
175             if (antwortParse === undefined) { //falls Fehler in SQL
176                 verzogerungEventCheck.reject("Datenbank nicht erreichbar");
177                 return; //HIER ABBRECHEN
178             }
179             if (!antwortParse) { //falls Table nicht existiert
180                 verzogerungEventCheck.reject("Event mit Name " + eventData.name + " existiert nicht in DB");
181                 return; //HIER ABBRECHEN
182             }
183
184             //eigentlicher Code
185             //Daten speichern
186             eventName = eventData.name;
187             //manuell resolve, weil bereits oben erstellt
188             verzogerungEventCheck.resolve();
189         });
190         //falls Fehler in Abfrage
191         verzogerungAbfrage.fail(function(fehler) {
192             verzogerungEventCheck.reject("Prüfung, ob Event mit Name " + eventData.name +
193                 " existiert in DB ist fehlgeschlagen: " + fehler);
194         });
195         verzogerungEvent = $.when(verzogerungEventAuslesen, verzogerungEventCheck)
196     });
197     //Event-Verzogerungen koppeln
198     var verzogerungEvent = $.when(verzogerungEventAuslesen, verzogerungEventCheck);
199
200     //Produkte auslesen und direkt speichern
201     var verzogerungProdukte = textAuslesen("eventData/produkte.json", function(inhalt) {
202         produkte = JSONtryParse(inhalt);
203         if (produkte === undefined) {

```

```

204     verzogerungProdukte.reject("eventData/produkte.json_JSON");
205   }
206 });
207
208 //Kategorien auslesen und direkt speichern
209 var verzogerungKategorien = textAuslesen("eventData/kategorien.json", function(inhalt) {
210   kategorien = JSONtryParse(inhalt);
211   if (kategorien === undefined) {
212     verzogerungKategorien.reject("eventData/kategorien.json_JSON");
213   }
214 });
215
216 //Tische auslesen und direkt speichern
217 var verzogerungTische = textAuslesen("eventData/tische.json", function(inhalt) {
218   tische = JSONtryParse(inhalt);
219   if (tische === undefined) {
220     verzogerungTische.reject("eventData/tische.json_JSON");
221   }
222 });
223
224 //Verzögerungen (jQuery Deferred) koppeln
225 var verzogerungAlle = $.when(verzogerungEvent, verzogerungProdukte,
226   verzogerungKategorien, verzogerungTische);
227
228 //wenn alles ausgelesen: Produkte den Kategorien zuordnen
229 verzogerungAlle.done(function() {
230   //kategorien.produkte property erstellen für alle Kategorien
231   for (var i=0; i < kategorien.length; i++) {
232     kategorien[i].produkte = [];
233   }
234   //Produkte den Kategorien zuordnen (kategorien.produkte Arrays füllen)
235   for (var i=0; i < produkte.length; i++) {
236     var zugehörigeKategorie = produkte[i].kategorie;
237     kategorien[zugehörigeKategorie].produkte.push(i);
238   }
239 });
240
241 return verzogerungAlle;
242 }
243
244
245 // .....
246 // ... DB: BESTELLUNG KONEVERTIEREN: STRING -> OBJEKT .....
247 // .....
248
249 //Bestellungen-Array-String zu einem Bestellungen-Array parsen
250 //auch alles Innere parsen (ZUKÜNFTIGES AUCH NOCH HINZUFÜGEN!!!!)
251 //sortiert Bestellungen nach aufsteigender ID
252 //optional: $.Deferred rejecten mit Fehlermeldung, falls Fehler
253 function bestellungenStringParsen(bestellungenString, verzogerung/* = undefined */) {
254   if (verzogerung === undefined) {
255     verzogerung = undefined;
256   }
257
258   var bestellungen; //Rückgabe-Variable definieren
259
260   //Bestellungen parsen
261   bestellungen = JSONtryParse(bestellungenString);
262   if (bestellungen === undefined) {
263     //verzogerung abbrechen
264     console.log("KRITISCHER FEHLER: Parsen der Bestellungen fehlgeschlagen. " + bestellungenString);
265     if (verzogerung) {
266       verzogerung.reject("KRITISCHER FEHLER: Parsen der Bestellungen fehlgeschlagen. " + bestellungenString);
267     }
268     return false;
269   }
270
271   //innere Eigenschaften von Bestellungen-Objekten auch noch parsen
272   //ACHTUNG: ZUKÜNFTIGES HIER AUCH HINZUFÜGEN!!!!!!!!!!!!
273   for (var i = 0; i < bestellungen.length; i++) {
274     //BESTELLUNGEN-ARRAY konvertieren
275     bestellungen[i].bestellt = JSONtryParse(bestellungen[i].bestellt);
276     //Fehler-Handling
277     if (bestellungen[i].bestellt === undefined) {
278       //verzogerung abbrechen
279       console.log("KRITISCHER FEHLER: in bestellungenStringParsen(): Parsen des bestellt-Array fehlgeschlagen.",
280         bestellungen[i]);
281       if (verzogerung) {
282         verzogerung.reject(
283           "KRITISCHER FEHLER: in bestellungenStringParsen(): Parsen des bestellt-Array fehlgeschlagen. " +
284           JSON.stringify(bestellungen[i]));
285       }
286       return false;
287     }
288     //Länge und Typ des Arrays überprüfen
289     if (!Array.isArray(bestellungen[i].bestellt) || bestellungen[i].bestellt.length != produkte.length) {
290       //verzogerung abbrechen
291       console.log(
292         "KRITISCHER FEHLER: in bestellungenStringParsen(): bestellt-Array hat falsche Länge (richtig ist: "
293         + produkte.length + ")", bestellungen[i]);
294       if (verzogerung) {
295         verzogerung.reject("KRITISCHER FEHLER: in bestellungenStringParsen():
296           bestellt-Array hat falsche Länge (richtig ist: " + produkte.length + ") " +
297           JSON.stringify(bestellungen[i]));
298       }
299       return false;
300     }
301
302     //BEZAHLT-ARRAY konvertieren
303     bestellungen[i].bezahlt = JSONtryParse(bestellungen[i].bezahlt);
304     //Fehler-Handling
305     if (bestellungen[i].bezahlt === undefined) {
306       //verzogerung abbrechen
307       console.log("KRITISCHER FEHLER: in bestellungenStringParsen(): Parsen des bezahlt-Array fehlgeschlagen.",
308         bestellungen[i]);

```

```

301     if (verzögerung) {
302         verzögerung.reject(
303             "KRITISCHER FEHLER: in bestellungenStringParse(): Parsen des bezahlt-Array fehlgeschlagen. " +
304             JSON.stringify(bestellungen[i]));
305     }
306     return false;
307 }
308 //Länge und Typ des Arrays überprüfen
309 if (!Array.isArray(bestellungen[i].bezahlt) || bestellungen[i].bezahlt.length !== produkte.length) {
310     //verzögerung abbrechen
311     console.log("KRITISCHER FEHLER: in bestellungenStringParse():
312         bezahlt-Array hat falsche Länge (richtig ist: " + produkte.length + ")", bestellungen[i]);
313     if (verzögerung) {
314         verzögerung.reject("KRITISCHER FEHLER: in bestellungenStringParse():
315             bezahlt-Array hat falsche Länge (richtig ist: " + produkte.length + ") " +
316             JSON.stringify(bestellungen[i]));
317     }
318     return false;
319 }
320 //STORNO-ARRAY (falls vorhanden) konvertieren
321 if (bestellungen[i].storno === undefined || bestellungen[i].storno === "") {
322     bestellungen[i].storno = undefined;
323 } else {
324     bestellungen[i].storno = JSON.parse(bestellungen[i].storno);
325     //Fehler-Handling
326     if (bestellungen[i].storno === undefined) {
327         //verzögerung abbrechen
328         console.log("KRITISCHER FEHLER: in bestellungenStringParse(): Parsen des storno-Array fehlgeschlagen.",
329             bestellungen[i]);
330         if (verzögerung) {
331             verzögerung.reject("KRITISCHER FEHLER: in bestellungenStringParse():
332                 Parsen des storno-Array fehlgeschlagen. " + JSON.stringify(bestellungen[i]));
333         }
334         return false;
335     }
336     //Länge und Typ des Arrays überprüfen
337     if (!Array.isArray(bestellungen[i].storno) || bestellungen[i].storno.length !== produkte.length) {
338         //verzögerung abbrechen
339         console.log("KRITISCHER FEHLER: in bestellungenStringParse():
340             storno-Array hat falsche Länge (richtig ist: " + produkte.length + ")", bestellungen[i]);
341         if (verzögerung) {
342             verzögerung.reject("KRITISCHER FEHLER:
343                 in bestellungenStringParse(): storno-Array hat falsche Länge (richtig ist: " +
344                 produkte.length + ") " + JSON.stringify(bestellungen[i]));
345         }
346         return false;
347     }
348 }
349 //verarbeitet konvertieren von dezimal-Zahl zu binär-String
350 // mit korrekter Anz. führender Nullen
351 bestellungen[i].verarbeitet = bestellungen[i].verarbeitet.toString(2);
352 bestellungen[i].verarbeitet.padStart(kategorien.length, "0");
353 if (bestellungen[i].verarbeitet.length > kategorien.length) {
354     //verzögerung abbrechen
355     console.log("KRITISCHER FEHLER: in bestellungenStringParse():
356         verarbeitet-Binär hat zu viele Stellen (richtig sind: " + kategorien.length + ")", bestellungen[i]);
357     if (verzögerung) {
358         verzögerung.reject("KRITISCHER FEHLER: in bestellungenStringParse():
359             verarbeitet-Binär hat zu viele Stellen (richtig sind: " + kategorien.length + ") " +
360             JSON.stringify(bestellungen[i]));
361     }
362     return false;
363 }
364 //ausgeliefert konvertieren von dezimal-Zahl zu binär-String mit
365 // korrekter Anz. führender Nullen
366 bestellungen[i].ausgeliefert = bestellungen[i].ausgeliefert.toString(2);
367 bestellungen[i].ausgeliefert.padStart(kategorien.length, "0");
368 if (bestellungen[i].ausgeliefert.length > kategorien.length) {
369     //verzögerung abbrechen
370     console.log("KRITISCHER FEHLER: in bestellungenStringParse():
371         ausgeliefert-Binär hat zu viele Stellen (richtig sind: " + kategorien.length + ")",
372             bestellungen[i]);
373     if (verzögerung) {
374         verzögerung.reject("KRITISCHER FEHLER: in bestellungenStringParse():
375             ausgeliefert-Binär hat zu viele Stellen (richtig sind: " + kategorien.length + ") " +
376             JSON.stringify(bestellungen[i]));
377     }
378     return false;
379 }
380 }
381 //nach aufsteigender ID sortieren
382 bestellungen.sort(function(a, b){return a.id - b.id});
383 return bestellungen;
384 }
385 // .....
386 // ..... DB: AUSLESEN .....
387 // .....
388 //ACHTUNG BEIM PROGRAMMIEREN:
389 //ARGUMENT DIREKT IN CALLBACK-FUNKTION, WEIL DAS URSPRÜNGLICH AUSGELESENE BEARBEITET WURDE
390 //(AUFPASSEN, DASS ARGUMENT NICHT GLEICHER NAME HAT WIE EINE BEREITS BESTEHENDE VARIABLE
391 // (Z.B. "BESTELLUNGEN"), SONST GANZ KOMISCHE FEHLER)
392 // .....
393 //Bestellungen mit bestimmten IDs auslesen aus Datenbank
394 //id: Array von IDs oder falls leer alle auslesen
395 //callback: Funktion wird aufgerufen, wenn verzögerung resolved wird,
396 // mit ausgelesenem Bestellungen-Array als Argument
397 function auslesenId(id/* = []*/, callback/* = function() {}*/) {
398     if (id === undefined) {

```

```

389     id = [];
390   }
391   if (callback === undefined) {
392     callback = function() {};
393   }
394   //Objekt mit relevanten Informationen zum Auslesen
395   var data = {
396     eventName: eventName,
397     id: id //Array von IDs
398   }
399
400   //Bestellungen aus Datenbank auslesen
401   //verzögerung wird erst nach der Funktion resolved
402   var verzögerung = dateiAufrufen("database/auslesenId.php", data, function(antwort) {
403     //Bestellungen-String parsen zu Array von Bestellungs-Objekten
404     //Funktion tut auch Verzögerung rejecten im Fehlerfall
405     var bestellungen = bestellungenStringParsen(antwort, verzögerung);
406
407     //Callback nur ausführen, falls verzögerung nicht oben bereits rejected wurde
408     verzögerung.done(function() { //Argument direkt in Callback
409       console.log("auslesenId(): ausgelesen", bestellungen);
410
411       callback(bestellungen);
412     });
413   });
414
415   return verzögerung; //resolved sobald Datei ausgelesen, konvertiert, verarbeitet
416 }
417
418 //Bestellungen von einem Tisch auslesen aus Datenbank
419 //callback: Funktion wird aufgerufen, wenn verzögerung resolved wird,
420 // mit ausgelesenem Bestellungen-Array als Argument
421 function auslesenTisch(tisch, callback/* = function() {}/) {
422   if (callback === undefined) {
423     callback = function() {};
424   }
425
426   //Objekt mit relevanten Informationen zum Auslesen
427   var data = {
428     eventName: eventName,
429     tisch: tisch
430   }
431
432   //Bestellungen aus Datenbank auslesen
433   //verzögerung wird erst nach der Funktion resolved
434   var verzögerung = dateiAufrufen("database/auslesenTisch.php", data, function(antwort) {
435     //Bestellungen-String parsen zu Array von Bestellungs-Objekten
436     //Funktion tut auch Verzögerung rejecten im Fehlerfall
437     var bestellungen = bestellungenStringParsen(antwort, verzögerung);
438
439     //Callback nur ausführen, falls verzögerung nicht oben bereits rejected wurde
440     verzögerung.done(function() { //Argument direkt in Callback
441       console.log("auslesenTisch(): ausgelesen", bestellungen);
442
443       callback(bestellungen);
444     });
445   });
446
447   return verzögerung; //resolved sobald Datei ausgelesen, konvertiert, verarbeitet
448 }
449
450 //Bestellungen auslesen, deren verarbeitet noch min
451 // ein 0 hat bei einer anzuzeigenden Kategorie (nurKategorien)
452 //nurKategorien: Kategorien, die diese Basis/Sprinter anzeigt (Array von kategorie.id)
453 //callback: Funktion wird aufgerufen, wenn verzögerung resolved wird,
454 // mit ausgelesenem Bestellungen-Array als Argument
455 function auslesenUnverarbeitet(nurKategorien, callback/* = function() {}/) {
456   if (callback === undefined) {
457     callback = function() {};
458   }
459
460   //Objekt mit relevanten Informationen zum Auslesen
461   var data = {
462     eventName: eventName,
463     zeigeKat: nurKategorien
464   }
465
466   //Bestellungen aus Datenbank auslesen
467   //verzögerung wird erst nach der Funktion resolved
468   var verzögerung = dateiAufrufen("database/auslesenUnverarbeitet.php", data, function(antwort) {
469     //Bestellungen-String parsen zu Array von Bestellungs-Objekten
470     //Funktion tut auch verzögerung rejecten im Fehlerfall
471     var bestellungen = bestellungenStringParsen(antwort, verzögerung);
472
473     //Callback nur ausführen, falls verzögerung nicht oben bereits rejected wurde
474     verzögerung.done(function() { //Argument direkt in Callback
475       //Callback nur ausführen, falls min eine relevante Bestellung
476       if (bestellungen.length !== 0) {
477         console.log("auslesenUnverarbeitet(): Bestellungen:", bestellungen);
478
479         callback(bestellungen);
480       } else {
481         console.log("auslesenUnverarbeitet(): keine neuen relevanten Bestellungen");
482       }
483     });
484   });
485
486   return verzögerung; //resolved sobald Datei ausgelesen, konvertiert, verarbeitet
487 }
488
489 //auslieferbare Bestellungen auslesen
490 //ALLE Kategorien verarbeitet UND noch MIN 1 Kategorie nicht ausgeliefert
491 //callback: Funktion wird aufgerufen, wenn verzögerung resolved wird,
492 // mit ausgelesenem Bestellungen-Array als Argument
493 function auslesenAusliefern(nurKategorien, callback/* = function() {}/) {

```

```

494     if (callback === undefined) {
495         callback = function() {};
496     }
497
498     //Objekt mit relevanten Informationen zum Auslesen
499     var data = {
500         eventName: eventName,
501         zeigeKat: nurKategorien
502     }
503
504     //Bestellungen aus Datenbank auslesen
505     //verzögerung wird erst nach der Funktion resolved
506     var verzögerung = dateIAufrufen("database/auslesenAusliefern.php", data, function(antwort) {
507         //Bestellungen-String parsen zu Array von Bestellungen-Objekten
508         //Funktion tut auch verzögerung rejecten im Fehlerfall
509         var bestellungen = bestellungenStringParsen(antwort, verzögerung);
510
511         //Callback nur ausführen, falls verzögerung nicht oben bereits rejected wurde
512         verzögerung.done(function() { //Argument direkt in Callback
513             //Callback nur ausführen, falls min eine relevante Bestellung
514             if (bestellungen.length !== 0) {
515                 console.log("auslesenAusliefern(): Bestellungen:", bestellungen);
516
517                 callback(bestellungen);
518             } else {
519                 console.log("auslesenAusliefern(): keine neuen relevanten Bestellungen");
520             }
521         });
522     });
523
524     return verzögerung; //resolved sobald Datei ausgelesen, konvertiert, verarbeitet
525 }
526
527 // .....
528 // ..... DB: EINTRAG ERSTELLEN / UPDATEN .....
529 // .....
530 // .....
531
532 //erzeugt eine neue Bestellung in der Datenbank
533 function insertBestellung(bestellung) {
534     //Objekt mit relevanten Informationen von bestellung
535     var data = {
536         eventName: eventName,
537         zeit: bestellung.zeit,
538         tisch: bestellung.tisch,
539         bestellt: JSON.stringify(bestellung.bestellt), //stringify
540         bestelltVon: bestellung.bestelltVon,
541         bezahlt: JSON.stringify(bestellung.bezahlt), //stringify
542         storno: JSON.stringify(bestellung.storno) //stringify
543     }
544
545     //Bestellung in Datenbank schreiben
546     var verzögerung = dateIAufrufen("database/insertBestellung.php", data, function(antwort) {
547         console.log("insertBestellung(): " + antwort);
548     });
549
550     return verzögerung;
551 }
552
553 //Bestellungen updaten in Datenbank
554 //updateDaten: Objekt mit Properties: id, bezahlt, bezahltVon,
555 //                verarbeitet, tisch, bestellt (genaues siehe unten)
556 //                id: Array von IDs oder eine ID als Zahl -> wo updaten
557 //                (muss zwingend vorhanden sein)
558 //                andere: auf was updaten
559 //                um Eigenschaften nicht upzudaten weglassen
560 //                oder undefined (nicht null) setzen
561 //callback(): erhält Anz. betroffener Zeilen
562 function bestellungenUpdaten(updateDaten, callback/* = function() {}/) {
563     if (callback === undefined) {
564         callback = function() {};
565     }
566
567     //falls ID nicht als Array, sondern als einzelne ID: Array erstellen mit 1 Element
568     if (!Array.isArray(updateDaten.id)) { //falls kein Array
569         //falls auch keine einzelne Zahl oder nicht definiert
570         if (typeof(updateDaten.id) === "number") {
571             updateDaten.id = [updateDaten.id];
572         } else { //falls auch keine einzelne Zahl oder nicht definiert
573             console.log("Fehler in bestellungenUpdaten(): updateDaten.id " +
574                 "weder Array noch Zahl (oder nicht definiert):", updateDaten.id);
575             //Verzögerung nicht rejecten, weil noch nicht erstellt und
576             //                das eigentlich nicht vorkommen sollte (nur Programmierfehler)
577             return;
578         }
579     }
580
581     //Objekt mit relevanten Informationen zum Auslesen
582     var data = {
583         eventName: eventName,
584         //Where
585         id: updateDaten.id, //Array von IDs. Achtung: nur Referenz zu updateDaten.id
586         //Werte
587         bezahlt: JSON.stringify(updateDaten.bezahlt), //stringify
588         storno: JSON.stringify(updateDaten.storno), //stringify
589         bezahltVon: updateDaten.bezahltVon,
590         verarbeitet: updateDaten.verarbeitet,
591         ausgeliefert: updateDaten.ausgeliefert,
592         tisch: updateDaten.tisch,
593         bestellt: JSON.stringify(updateDaten.bestellt) //stringify
594         //Anmerkungen:
595         //id: Array von IDs (wird automatisch zu php-Array konvertiert)
596         //falls id nicht mehr existiert oder null: nichts passiert, auch keinen Fehler
597         //verarbeitet ist Array von zu setzenden Kategorien
598         //ausgeliefert: analog zu verarbeitet

```

```

599     //tisch und bestellt sollten eigentlich nicht mehr geändert werden
600     //uhrzeit und bestelltVon können nicht geändert werden
601 }
602
603 //Bestellungen in Datenbank aktualisieren
604 //verzögerung wird erst nach der Funktion resolved
605 var verzögerung = dateiAufrufen("database/updates.php", data, function(anzZeilen) {
606     //Ausnamen & Fehler Handling
607     var fehler = false;
608     var pseudoFehler = false;
609     //pseudoFehler:nicht upgedatet, aber Eintrag schon richtig
610     //ACHTUNG: eigentlich kleine Verzögerung
611
612     //falls zu wenig aktualisiert in DB:
613     if (anzZeilen < data.id.length) {
614         pseudoFehler = true;
615         //evt. schon richtig oder bereits von anderem Gerät geändert
616         //prüfen, ob in Datenbank sowieso schon steht,
617         //was hineingeschrieben werden sollte. Dann kein eigentlicher Fehler
618         //data und extra Ausgelesenes vergleichen
619         var verzögerungAuslesen = auslesenId(data.id, function(bestellungen) {
620             //für alle ausgelesenen Bestellungen
621             for (var i=0; i < bestellungen.length; i++) {
622                 //ACHTUNG: evt. weniger Bestellungen ausgelesen, als upgedatet.
623                 //Die fehlenden wurden vermutlich schon gelöscht
624                 // und werden hier ignoriert (bzw. als richtig angesehen).
625                 var bestellung = bestellungen[i];
626                 //alle Eigenschaften prüfen
627                 for (var prop in data) {
628                     //eventName und id auslassen
629                     if (prop === "eventName" || prop === "id") {
630                         continue;
631                     }
632                     if (data[prop]) { //wurde Eigenschaft gesetzt?
633                         //Eigenschaften mit Spezialbehandlung wegen Binärlogik:
634                         // verarbeitet und ausgeliefert
635                         if (prop === "verarbeitet" || prop === "ausgeliefert") {
636                             var vonData = binArray(data[prop]);
637                             var vonBestellung = parseInt(bestellung[prop], 2);
638                             if ((vonData & vonBestellung) === vonData) {
639                                 continue; //richtig: weiter
640                             } else {
641                                 fehler = true;
642                                 break; //falsch: abbrechen
643                             }
644                         }
645
646                         //normale Eigenschaften: prüfen ob richtig
647                         if (data[prop] === bestellung[prop]) {
648                             } else {
649                                 fehler = true;
650                                 break;
651                             }
652                         }
653                     }
654                 }
655                 if (fehler) { //äusseren loop abbrechen
656                     break;
657                 }
658             }
659         });
660
661         //falls auslesen fehlgeschlagen: sicherheitshalber von Fehler ausgehen
662         verzögerungAuslesen.fail(function() {
663             fehler = true;
664             //ausgelesen = "Fehler"; //was ist das????
665         });
666
667         //immer ausführen, egal ob resolved oder rejected
668         verzögerungAuslesen.always(function(ausgBestellungen) {
669             if (fehler) { //pseudoFehler sowieso true
670                 //falls tatsächlich Fehler:
671                 //verzögerung abbrechen
672                 console.log("FEHLER: in bestellungenUpdates(): weniger Zeilen aktualisiert in der Datenbank,
673                     wie Bestellungen aktualisiert.: Anz. bearbeiteter Zeilen: " + anzZeilen + " (" +
674                     ausgBestellungen, ")", Anz. Bestellungen: " + data.id.length + " (" + data.id, ")");
675                 verzögerung.reject("FEHLER: in bestellungenUpdates(): weniger Zeilen aktualisiert in der Datenbank,
676                     wie Bestellungen aktualisiert.: Anz. bearbeiteter Zeilen: " + anzZeilen + " (" +
677                     JSON.stringify(ausgBestellungen) + ")", Anz. Bestellungen: " + data.id.length + " (" +
678                     JSON.stringify(data.id) + ")");
679             } else {
680                 //falls kein richtiger Fehler sonder nur pseudoFehler:
681                 console.log("ANMERKUNG: bestellungenUpdates(): nicht gleich viele Zeilen geändert in DB (" +
682                     anzZeilen + ")", wie Bestellungen zum Update gegeben (" + data.id.length +
683                     "). Aber jetzt alle Einträge richtig (ein Teil war bereits richtig). Zum Update gegeben:",
684                     data.id);
685
686                 //anzZeilen MANIPULIEREN zu Wert, den sie eigentlich haben sollte
687                 anzZeilen = data.id.length;
688                 callback(anzZeilen); //CALLBACK (Möglichkeit 1)
689             }
690         });
691     } else if (anzZeilen > data.id.length) { //zu viel aktualisiert in DB ?!
692         fehler = true;
693         //verzögerung abbrechen
694         console.log("KRITISCHER FEHLER: in bestellungenUpdates(): mehr Zeilen aktualisiert in der Datenbank,
695             wie Bestellungen aktualisiert.: Anz. bearbeiteter Zeilen: " + anzZeilen + " , Anz. Bestellungen: " +
696             data.id.length + " , Bestellungen:", data.id);
697         verzögerung.reject("KRITISCHER FEHLER: in bestellungenUpdates(): mehr Zeilen aktualisiert in der Datenbank,
698             wie Bestellungen aktualisiert.: Anz. bearbeiteter Zeilen: " + anzZeilen + " , Anz. Bestellungen: " +
699             data.id.length + " , Bestellungen:" + JSON.stringify(data.id));
700     }
701 }
702 //hier fertig: Fehler & Ausnahmen Handling

```

```

692     //falls Fehler, wurde verzögerung bereits oben rejected
693     //Callback kann also direkt aufgerufen werden (ohne verzögerung.done)
694     if (!pseudoFehler) {
695         console.log("Bestellungen upgedatet auf (undefined = nicht upgedatet):", data);
696     }
697     callback(anzZeilen); //CALLBACK (Möglichkeit 2)
698 }
699
700 }); //hier fertig: Code von Callback von dateiAufrufen()
701
702 return verzögerung;
703 }
704
705 // .....
706 // ..... XML_HTTP_REQUEST / AJAX .....
707 // .....
708 // .....
709
710 //Datei direkt auslesen (via XMLHttpRequest)
711 //url: Datei-Pfad als string (z.B.: ordner/beispiel.json)
712 //callback: Funktion aufgerufen bei Erfolg; mit ausgelesenen Text als Argument
713 //return: jQuery deferred, resolved / rejected mit Fehler Nachricht
714 function textAuslesen(url, callback) {
715     if (callback === undefined) {
716         callback = function() {};
717     }
718
719     var verzögerung = new $.Deferred();
720
721     //Anfrage senden
722     var anfrage = new XMLHttpRequest();
723     anfrage.open("POST", url /*+ "?t=" + Math.random()*/ , true);
724     // (unkommentieren falls GET, um laden aus Cache zu verhindern)
725
726     anfrage.timeout = 10000; //wie lange es dauern darf, bevor abgebrochen wird
727
728     anfrage.send(); //Anfrage abschicken
729
730     //wenn es zu lange dauert: abbrechen und deferred.reject()
731     anfrage.ontimeout = function() {
732         verzögerung.reject(url + " timeout");
733         //console.log("Laden von " + url + " fehlgeschlagen. timeout");
734     };
735
736     //wenn Anfrage empfangen
737     anfrage.onreadystatechange = function() {
738         if (this.readyState == 4) { //Anfrage fertig bearbeitet
739             if (this.status == 200) { //Anfrage erfolgreich
740                 callback(this.responseText);
741                 verzögerung.resolve();
742                 //console.log("Laden von " + url + " erfolgreich.");
743             } else { //Anfrage fehlgeschlagen
744                 verzögerung.reject(url + "_" + this.status);
745                 //this.status zeigt aus unerfindlichem Grund falsche falsche Zahl an!
746                 //console.log("Laden von" + url + " fehlgeschlagen. Fehler: " + this.status);
747             }
748         }
749     };
750
751     return verzögerung; //deferred zurückgeben
752 }
753
754 //Datei aufrufen (z.B. PHP)
755 //data: Objekt: Nur Zahlen (int oder float/double) und Strings. Arrays stringifyen
756 //callback: aufgerufen im Erfolgsfall mit antwort (bevor verzögerung.resolve())
757 //gibt ein Deferred zurück:
758 //resolve mit antwort, falls ajax funktioniert hat UND php,
759 //reject mit Fehler, falls ajax ODER php fehlgeschlagen
760 function dateiAufrufen(url, data_ /* = null*/, callback /* = function(){} */) {
761     if (data_ === undefined) {
762         data_ = null;
763     }
764     if (callback === undefined) {
765         callback = function() {};
766     }
767
768     var verzögerung = new $.Deferred();
769
770     $.ajax({
771         url: url, //Datei Url
772         type: "POST",
773         dataType: "text", //keine Konvertierungen vornehmen durch jQuery
774         cache: false,
775
776         data: (data_),
777
778         success: function(antwort) { //ajax hat funktioniert
779             //Fehler in php Datei, Fehler-Token: ERROR_
780             //kann ein ' ' oder ein '' (nur bei JSON) vorgestellt haben (wieso?)
781             if (antwort.startsWith("ERROR.") ||
782                 antwort.startsWith(" ERROR.") ||
783                 antwort.startsWith("\ ERROR.") ||
784                 antwort.startsWith("\ ERROR.") ) {
785                 verzögerung.reject(antwort);
786             } else { //kein Fehler: OK
787                 callback(antwort);
788                 verzögerung.resolve(antwort);
789             }
790         },
791
792         //Fehlerhandhabung
793         timeout: 10000,
794
795         //ajax fehlgeschlagen:
796         error: function(XMLHttpRequest, textStatus, exceptionObject) {

```

```

797     var arg1 = XMLHttpRequest;
798     var arg2 = textStatus;
799     var arg3 = exceptionObject;
800     //Reihenfolge der Argumente ändern:
801     verzogerung.reject(textStatus, exceptionObject, XMLHttpRequest);
802   }
803 });
804
805   return verzogerung;
806 }
807
808 //schreiben/lesen via php: siehe Infos
809
810
811 // =====
812 // ===== TECHNISCH =====
813 // =====
814
815 // .....
816 // ..... JSON .....
817 // .....
818
819 //überprüfen, ob ein JSON-String OK ist.
820 //return: Resultat oder undefined für Fehler
821 //      (undefined kann kein Resultat sein von JSON.parse())
822 //ACHTUNG: unbedingt "=== undefined" anstatt "== false" oder "!" verwenden
823 //      zum prüfen, sonst komische Resultate mit (leeren) Arrays.
824 function JSONtryParse(jsonString){
825   try {
826     var jsonParsed = JSON.parse(jsonString);
827     return jsonParsed;
828   }
829   catch (e) {
830     return undefined; //falls fehlgeschlagen
831   }
832 }
833
834
835 // .....
836 // ..... BINÄR .....
837 // .....
838
839 //errechnet eine binäre Zahl aus den Array-Elementen (Ausgabe als Dezimalzahl)
840 //setzt alle x-ten Bits auf 1, für alle x im Array
841 //Summe von 2^x für alle x im Array
842 function binArray(arr){
843   var resultat = 0;
844   for (var i=0, len = arr.length; i < len; i++) {
845     resultat += Math.pow(2, arr[i]);
846   }
847   return resultat;
848 }
849
850
851 // .....
852 // ..... ARRAY VERGLEICHE .....
853 // .....
854 //prüfen, ob alle Elemente eines ersten Arrays grösser/gleich sind
855 //als die zugehörigen Elemente eines zweiten Arrays
856 //falls arrGross-length kleiner als arrKlein-length: false
857 //arrGross: Array, der grösser/gleich sein sollte
858 //arrKlein: Array, der kleiner/gleich sein sollte
859 function arrGrösserAls(arrGross, arrKlein) {
860   if (arrGross.length < arrKlein.length) {
861     return false;
862   }
863
864   for (var i = 0; i < arrKlein.length; i++) {
865     if (arrGross[i] < arrKlein[i]) { //arrGross kleiner
866       return false;
867     }
868   }
869   //falls oben nie false: arrGross ist grösser/gleich arrKlein
870   return true;
871 }
872
873
874 // .....
875 // ..... ERWEITERTE ANIMATIONEN .....
876 // .....
877
878 //PSEUDO ALTERNATIVEN ZU JQUERY slideUp(), slideDown(), slideToggle()
879 //akzeptieren auch mehrere Boxen (normales jQuery-Objekt)
880 //SLIDEDOWN: WICHTIG: SICHERSTELLEN, DASS PLATZHALTER IMMER NOCH DIREKT NACH BOX IST
881 //falls toggle mehrmals direkt nacheinander aufgerufen: Lag, aber wird wieder gehen
882 //Funktionsprinzip: Erstellt ein Platzhalter-div und verwendet slide auf das
883 //      -> funktioniert für alles (auch für Tabellen)
884 //boxen: jquery object der zu animierenden Elemente
885 function slideUp(boxen, dauer/* = 200*/, callback/* = function() {}*/) {
886   if (dauer === undefined) {
887     dauer = 200;
888   }
889   if (callback === undefined) {
890     callback = function() {};
891   }
892
893   for(var i = 0; i < boxen.length; i++) {
894     var box = boxen.eq(i);
895     //platzhalter erstellen (muss Text enthalten,
896     //      funktioniert sonst manchmal nicht (div wird nicht angezeigt))
897     var platzhalter = $("

8. Januar 2018



49/85


```

```

902     $(box).after(platzhalter); //platzhalter unmittelbar nach box einfügen
903     //jQuery slideUp-Animation für platzhalter
904     $(platzhalter).slideUp(dauer, function() {
905         this.remove(); //platzhalter wieder löschen
906         callback();
907     });
908     }
909 }
910 function slideDown(boxen, dauer/* = 200*/, callback/* = function() {}*/) {
911     if (dauer === undefined) {
912         dauer = 200;
913     }
914     if (callback === undefined) {
915         callback = function() {};
916     }
917     for(var i = 0; i < boxen.length; i++) {
918         var box = boxen.eq(i);
919         //platzhalter erstellen (muss Text enthalten,
920         // funktioniert sonst manchmal nicht (div wird nicht angezeigt))
921         var platzhalter = $("

### 3.2. polyfill.js



```

1  /*
2  * Datei: polyfill.js
3  * Projekt: Kellner App
4  * Autor: Michael Heider (Quellen jeweils bei den einzelnen Polyfills)
5  * Datum: HS 2017
6  *
7  * Was:
8  * Polyfills für sehr neue Funktionen (ECMA 2017)
9  * Quelle jeweils bei den einzelnen Polyfills
10 */
11
12
13 // Polyfill für String.prototype.padStart
14 // Quelle: URL (28.11.2017):
15 // https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/padStart
16 if (!String.prototype.padStart) {
17     String.prototype.padStart = function padStart(targetLength,padString) {
18         targetLength = targetLength>>0; //floor if number or convert non-number to 0;
19         padString = String(padString || ' ');
20         if (this.length > targetLength) {
21             return String(this);
22         }
23         else {
24             targetLength = targetLength-this.length;
25             if (targetLength > padString.length) {
26                 //append to original to ensure we are longer than needed
27                 padString += padString.repeat(targetLength/padString.length);
28             }
29             return padString.slice(0,targetLength) + String(this);
30         }
31     };
32 }
33
34 // Polyfill für String.prototype.padEnd
35 // Quelle: URL (29.11.2017):
36 // https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/padEnd
37 if (!String.prototype.padEnd) {
38     String.prototype.padEnd = function padEnd(targetLength,padString) {
39         targetLength = targetLength>>0; //floor if number or convert non-number to 0;
40         padString = String(padString || ' ');
41         if (this.length > targetLength) {
42             return String(this);
43         }
44         else {
45             targetLength = targetLength-this.length;
46             if (targetLength > padString.length) {
47                 //append to original to ensure we are longer than needed
48                 padString += padString.repeat(targetLength/padString.length);
49             }

```



8. Januar 2018



50/85


```

```

50         return String(this) + padString.slice(0,targetLength);
51     }
52 };
53 }

```

## 4. stylesheets

### 4.1. gemeinsam.css

```

1  /*
2  * Datei: gemeinsam.css
3  * Projekt: Kellner App
4  * Autor: Michael Heider
5  * Datum: HS 2017
6  *
7  * Was:
8  * CSS für alle Stationen (Satellit, Basis, Sprinter)
9  */
10
11
12 /* =====
13 / ===== ALLGEMEINES =====
14 / =====*/
15 p, .schrift, .grosseschrift { /*mit Schriftgröße 16px: Gesamthöhe 36px*/
16     padding: 8px;
17 }
18
19 .grosseschrift {
20     font-size: 22px;
21 }
22
23 .rahmen {
24     border-width: 1px;
25     border-style: solid;
26     border-color: black;
27
28     box-sizing: border-box;
29 }
30
31 .rahmenUnten { /*Klasse rahmen und unten: Rahmen nur unten*/
32     border-width: 0px 0px 1px 0px; /*oben, rechts, unten, links*/
33 }
34
35 .pseudoVersteckt {
36     height: 0px;
37     margin: 0px;
38     padding: 0px;
39     border-width: 0px;
40 }
41
42 #laden {
43     position: absolute;
44     top: 0px;
45     left: 0px;
46     height: 100%;
47     width: 100%;
48     z-index: 9998;
49
50     background-color: #C0C0C0;
51 }
52
53 #laden > * {
54     position: absolute;
55     top: 50%;
56     left: 50%;
57     transform: translate(-50%, -50%);
58
59     font-size: 30px;
60     font-weight: bold;
61
62     background-color: silver;
63 }
64
65 #fehler {
66     position: absolute;
67     top: 0px;
68     left: 0px;
69     height: 100%;
70     width: 100%;
71     z-index: 9999;
72
73     background-color: #C0C0C0;
74 }
75
76
77 /* .....
78 / ..... CSS FÜR CHECKBOX (FORM) .....
79 / .....*/
80 /* CSS für checkbox (form)
81 * ANPASSEN: alles als Breite/Höhe gekennzeichnete jeweils gleich.
82 *           Höhe und Breite min. 13px.
83 * weitere Pseudoklassen einfügen folgendermassen:
84 * input[type=checkbox]:checked + label:before
85 * HTML: <input type="checkbox"> <label></label> (label zwingend nötig)
86 * Funktionsweise: input[type=checkbox] im Wesentlichen ersetzt durch label:before
87 */

```

```

88 input[type=checkbox] {
89     opacity: 0;
90
91     cursor: pointer;
92
93     width: 40px; /*APASSEN: Breite*/
94     height: 40px; /*APASSEN: Höhe*/
95 }
96 label {
97     display: inline-block;
98
99     position: relative;
100
101     pointer-events: none;
102 }
103 label:before {
104     content: "";
105     display: inline-block;
106     text-align: center;
107
108     position: absolute;
109     bottom: 0px; /*vertikaler Offset: ACHTUNG: auf PC und Handy allenfalls verschieden*/
110     right: 0px; /*eventuell ein Textknoten eingefügt vom Browser (sonst 0px)*/
111
112     width: 40px; /*APASSEN: Breite*/
113     height: 40px; /*APASSEN: Höhe*/
114     line-height: 40px; /*APASSEN: Höhe (für vertikales zentrieren)*/
115
116     /*APASSEN: Stil unchecked*/
117     color: #f3f3f3;
118     font-size: 30px;
119     background-color: #aaa;
120     box-shadow: inset 0px 2px 3px 0px rgba(0, 0, 0, .3),
121                 0px 1px 0px 0px rgba(255, 255, 255, .8);
122     border-radius: 3px;
123 }
124 input[type=checkbox]:checked + label:before {
125     content: "✓"; /*ANPASSEN: Ausgewählt-Zeichen*/
126
127     /*ANPASSEN: Stil checked (übernommen von unchecked (label:before))*/
128 }
129
130
131 /* =====
132 / ===== TABELLEN & CO =====
133 / =====*/
134 table {
135     width: 100%;
136 }
137
138 tr {
139     height: 36px;
140
141     border-bottom: 1px solid black;
142 }
143
144 td, th {
145     padding: 4px;
146
147     white-space: nowrap; /*Text wir niemals auf die nächste Linie gehen*/
148     overflow: hidden; /*zu langer Text abschneiden (keine Scrollbar)*/
149 }
150
151 caption {
152     text-align: left;
153
154     font-weight: bold;
155 }
156
157
158 /* =====
159 / ===== FARBEN =====
160 / =====*/
161 .kategorie {
162     background-color: #D5FFFF;
163 }
164
165 .produkt {
166     background-color: #FFFFD5;
167 }
    
```

## 4.2. reset.css

```

1 /*
2  * Datei: reset.css
3  * Projekt: Kellner App
4  * Autor: Michael Heider
5  * Datum: HS 2017
6  *
7  * Was:
8  * setzt wichtigste Eigenschaften auf vernünftige Werte
9  * stellt sicher, dass Browser Werte überschrieben werden und überall gleich sind
10 */
11
12
13 /* ALLGEMEINES */
14 * {
15     margin: 0;
16     padding: 0;
    
```

```

17
18     /*height property beinhaltet innere Höhe, padding und border
19     (anstatt nur innere Höhe)*/
20     box-sizing: border-box;
21     border-width: 0px;
22     border-style: solid;
23     border-color: black;
24
25     font-family: "Roboto", "Helvetica", "San Francisco", "sans-serif";
26     font-size: 16px; /*line-height ist "normal" (~ 120% der font-size)*/
27 }
28
29 html, body {
30     height: 100%;
31     width: 100%;
32 }
33
34 /* TABELLEN & CO */
35 table {
36     table-layout: fixed;
37
38     border-collapse: collapse;
39     border: 0px solid black;
40 }
41
42 table, caption, tr, td, th {
43     font-size: inherit;
44     font-weight: inherit;
45     font-style: inherit;
46     font-variant: inherit;
47 }
    
```

---

## 5. Web-App

### 5.1. manifestBasis.json

```

1 {
2     "lang": "de",
3     "dir": "ltr",
4     "name": "Kellner App: Basis",
5     "short_name": "KellnerApp Basis",
6     "description": "Festwirtschaft leicht gemacht",
7     "display": "standalone",
8     "orientation": "portrait",
9     "background_color": "#FFFD5",
10    "theme_color": "#000000",
11    "icons": [
12        {
13            "src": "icon180x180.png",
14            "sizes": "180x180",
15            "type": "image/png"
16        }
17    ],
18    "scope": "/"
19 }
    
```

---

### 5.2. manifestSatellit.json

```

1 {
2     "lang": "de",
3     "dir": "ltr",
4     "name": "Kellner App: Satellit",
5     "short_name": "KellnerApp Satellit",
6     "description": "Festwirtschaft leicht gemacht",
7     "display": "standalone",
8     "orientation": "portrait",
9     "background_color": "#FFFD5",
10    "theme_color": "#000000",
11    "icons": [
12        {
13            "src": "icon180x180.png",
14            "sizes": "180x180",
15            "type": "image/png"
16        }
17    ],
18    "scope": "/"
19 }
    
```

---

### 5.3. manifestSprinter.json

```

1 {
2     "lang": "de",
3     "dir": "ltr",
4     "name": "Kellner App: Sprinter",
5     "short_name": "KellnerApp Sprinter",
6     "description": "Festwirtschaft leicht gemacht",
7     "display": "standalone",
    
```

```
8   "orientation": "portrait",
9   "background_color": "#FEFFD5",
10  "theme_color": "#000000",
11  "icons": [
12    {
13      "src": "icon180x180.png",
14      "sizes": "180x180",
15      "type": "image/png"
16    }
17  ],
18  "scope": "/"
19 }
```

---

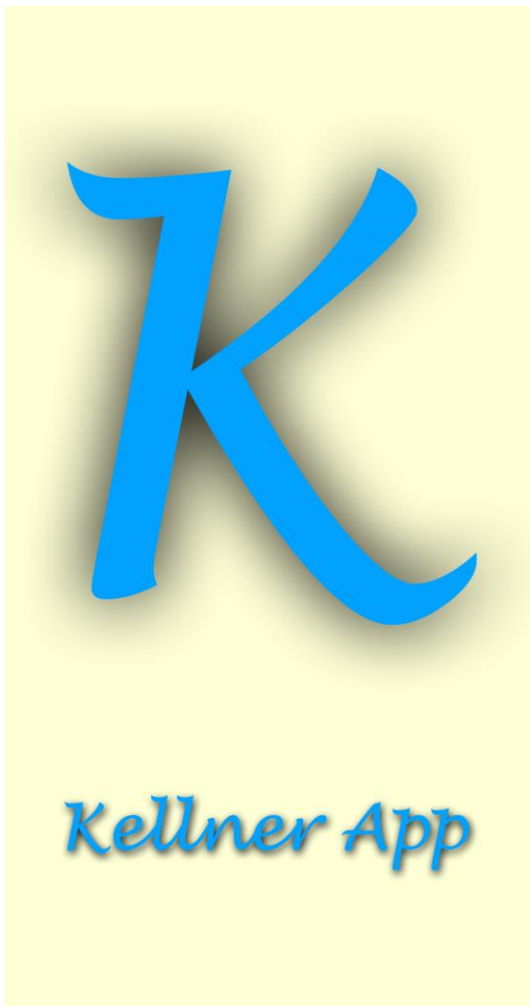
#### 5.4. icon180x180.png



---

#### 5.5. startup1900x1000.png

Die Abbildung wurde verkleinert.



## 6. basis.css

```
1 /*
2  * Datei: sprinter.js
3  * Projekt: Kellner App
4  * Autor: Michael Heider
5  * Datum: HS 2017
6  *
7  * Was:
8  * CSS zu Basis
9  */
10
11
12 /* =====
13 / ===== ALLGEMEINE KLASSEN =====
14 / =====*/
15
16 /* ALLGEMEINES */
17 /*siehe auch in gemeinsam.css*/
18
19 /* SEITENSTRUKTUR */
20 #inhalt {
21     height: 100%;
22     width: 100%;
23 }
24
25 .halfte {
26     height: 100%;
27     width: 45%;
28
29     float: left;
30 }
31
32 #mitte {
33     height: 100%;
34     width: 10%;
35
36     float: left;
37 }
38
39 .header {
40     position: fixed;
41     top: 0%;
42
43     width: inherit;
44     height: 55px;
45
46     text-align: center;
47
48     background-color: green;
49 }
50
51 .bestellungen {
52     position: absolute;
53     width: inherit;
54     top: 55px;
55     bottom: 0%;
56
57     overflow-x: hidden; /*horizontale Scroll-bar verstecken*/
58     overflow-y: scroll; /*vertikale Scroll-bar anzeigen*/
59 }
60
61
62 /* =====
63 / ===== TABELLEN =====
64 / =====*/
65 caption {
66     margin-top: 16px;
67     margin-bottom: 8px;
68 }
69
70 tr {
71     height: 30px;
72 }
73
74 tr:nth-child(2) { /*oberer Rahmen (zweites Kind, nach caption (beginnt mit 1))*/
75     border-top: 1px solid black;
76 }
77
78 td, th {
79     padding-left: 8px;
80     padding-right: 8px;
81 }
82
83 .verarbeitet {
84     margin-top: 10px;
85
86     float: right;
87
88     background-color: red;
89 }
90
91 .feldName {
92 }
93
94 .feldMenge {
95     width: 35px;
96
97     text-align: right;
98 }
99
100
```

```

101 /* =====
102 /  ===== FARBLICHE HERVORHEBUNG =====
103 /  =====*/
104 .ausgeliefert {
105     color: #aaaaaa;
106 }

```

## 7. basis.html

```

1  <!--
2  < Datei: basis.html
3  < Projekt: Kellner App
4  < Autor: Michael Heider
5  < Datum: HS 2017
6  <
7  < Was:
8  < HTML zu Basis.
9  < Enthält Grund-HTML für Layout.
10 < Alles, was EventDaten (zum Beispiel Produktnamen) verwendet,
11 < wird dynamisch erzeugt von basis.js.
12 -->
13
14
15 <!DOCTYPE html>
16 <html>
17 <head>
18 <meta charset="utf-8"/>
19
20 <title>Kellner App Basis</title>
21
22
23 <!--=====
24 < ===== HEAD BEREICH =====
25 < =====>
26
27 <!-- Korrekte Darstellung/Einstellung auf Smartphones -->
28 <meta name="viewport" content="width=device-width, initial-scale=1.0, user-scalable=no">
29
30
31 <!--.....
32 < ..... ZEUG FÜR WEB-APP .....
33 < .....>
34
35 <!--..... ANDROID .....>
36 <!-- (ACHTUNG: MUSS OBEN AN APPLE TAGS SEIN!!! (wieso??))-->
37 <link rel="manifest" href="webapp/manifestBasis.json">
38
39 <!--..... IOS .....>
40 <!-- ist eine Web App -->
41 <meta name="apple-mobile-web-app-capable" content="yes"/>
42 <!-- Statusbar Farbe und Stil -->
43 <meta name="apple-mobile-web-app-status-bar-style" content="black"/>
44 <!-- Icon auf Homescreen -->
45 <link rel="apple-touch-icon" sizes="180x180" href="webapp/icon180x180.png">
46 <!-- Icon während Ladevorgang (ca. 1900x1000 Pixel) -->
47 <link rel="apple-touch-startup-image" href="webapp/startup1900x1000.png">
48 <!-- Text auf Homescreen (anstatt Text aus title Tag) -->
49 <meta name="apple-mobile-web-app-title" content="KellnerApp Basis"/>
50
51 <!--..... SERVICE-WORKER .....>
52 <!-- ServiceWorker (braucht es einfach für Web Apps) -->
53 <!-- Code leicht bearbeitet von URL: (11.8.2017)
54 < https://developers.google.com/web/fundamentals/getting-started/primers/service-workers
55 -->
56 <script type="text/javascript">
57     if ("serviceWorker" in navigator) {
58         window.addEventListener("load", function() {
59             navigator.serviceWorker.register("sw.js").then(function(registration) {
60                 //Registrierung war erfolgreich
61                 console.log("ServiceWorker-Registrierung erfolgreich. Scope: ",
62                     registration.scope);
63             }, function(fehler) {
64                 //Registrierung fehlgeschlagen
65                 console.log("ServiceWorker-Registrierung fehlgeschlagen: ", fehler);
66             });
67         });
68     }
69 </script>
70
71
72 <!--.....
73 < ..... CSS .....
74 < .....>
75 <!-- Stylesheets einbinden -->
76 <link rel="stylesheet" type="text/css" href="stylesheets/reset.css">
77 <link rel="stylesheet" type="text/css" href="stylesheets/gemeinsam.css">
78 <link rel="stylesheet" type="text/css" href="basis.css">
79
80
81 <!--.....
82 < ..... JAVASCRIPT .....
83 < .....>
84 <!-- Scripts einbinden -->
85 <script type="text/javascript" src="scripts/polyfill.js"></script>
86 <script type="text/javascript" src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
87 <script type="text/javascript" src="scripts/allgemeineFunktionen.js"></script>
88 <!-- Hauptsript (auszuführendes Script) -->
89 <script type="text/javascript" src="basis.js"></script>
90

```

```

91 </head>
92
93
94 <!--=====
95 < ===== BODY BEREICH =====
96 < =====>
97 <body>
98
99 <!--..... LADEN .....-->
100 <div id="laden">
101   <span>LADEN</span>
102 </div>
103
104 <!--..... FEHLER .....-->
105 <div id="fehler" style="display: none;" style="display:none;">
106   <p><b>KRITISCHE FEHLER:</b></p><br>
107 </div>
108
109 <!--..... INHALT .....-->
110 <div id="inhalt" style="display:none;">
111   <!-- LINKS -->
112   <section class="halfte">
113     <section class="header rahmen schrift grosseschrift">in Produktion (alt)</section>
114     <!--Bestellungen-->
115     <section id="alteBestellungen" class="bestellungen rahmen"></section>
116   </section>
117
118   <!-- Mitte -->
119   <section id="mitte"></section>
120
121   <!-- RECHTS -->
122   <section class="halfte">
123     <section class="header rahmen schrift grosseschrift">neu</section>
124     <!--Bestellungen-->
125     <section id="neueBestellungen" class="bestellungen rahmen"><div></div></section>
126   </section>
127 </div>
128
129 </body>
130 </html>

```

## 8. basis.js

```

1  /*
2  * Datei: sprinter.js
3  * Projekt: Kellner App
4  * Autor: Michael Heider
5  * Datum: HS 2017
6  *
7  * Was:
8  * Javascript-Logik Basis
9  */
10
11 "use strict"; //strict mode aktivieren
12
13
14 //Warnung einblenden bevor schliessen oder versehentlichem Neuladen
15 // (speziell auf Handy durch runterziehen)
16 //auf einigen Browser muss zuerst mit der Webseite interagiert worden sein,
17 // bevor die Meldung auftritt
18 window.addEventListener("beforeunload", function(e) {
19   //Nachricht wird nicht auf allen Browsern angezeigt, sondern irgendein Default
20   var nachricht = "Wenn Sie die App neu laden, müssen Sie sich erneut einloggen.";
21   e.returnValue = nachricht; //zwei return Zeilen für umfassenderen Browser-Support
22   return nachricht;
23 });
24
25
26 // =====
27 // ===== VARIABLEN, OBJEKTE, OBJECT CONSTRUCTORS, ETC. =====
28 // =====
29 // ..... IDENTIFIKATION .....
30 var basisId = "Basis 1";
31 var eventName; //wird ausgelesen in auslesen()
32 var zeigeKategorien = [0,1,2,3];
33 //Produkte aus welchen Kategorien (Ids) angezeigt werden sollen (einstellbar).
34 //Array von Kategorie-IDs (muss nicht der Reihe nach sein).
35 //Sicherstellen, dass es KEINE NICHT EXISTIERENDEN kategorien gibt., keine führenden 0,
36 // können mehrmals vorkommen
37 //KEINE ÜBERLAGERUNGEN IN zeigeKategorien zwischen Basen (siehe bestellungenUpdates.php)
38 //muss übereinstimmen mit Array von zugehörigem Sprinter
39
40 // ..... BESTELLUNGEN .....
41 var angezeigteBestellungen = []; //IDs von Bestellungen, die im Moment angezeigt werden
42 // und also noch unverarbeitet sind
43 //siehe auch ErstelleBestellung-Constructor (allgemeineFunktionen.js)
44
45 // ..... EVENT DATEN .....
46 var produkte = []; //alle Produkte. Position in Array stimmt mit produkt.id überein
47 //siehe auch ErstelleProdukt-Constructor (allgemeineFunktionen.js)
48 var kategorien = []; //alle Kategorien. Position in Array stimmt mit kategorie.id überein
49 //siehe auch ErstelleKategorie-Constructor (allgemeineFunktionen.js)
50 var tische = [];
51 //siehe auch ErstelleTisch-Constructor (allgemeineFunktionen.js)
52
53 // ..... AUSLESEN .....
54 var auslesenInterval; //Interval zum neueAuslesen()
55 var auslesenErfolg = 5; //Erfolgszähler, stellt sicher, dass neueAuslesen()
56 // nicht zu viele Male nacheinander fehlschlägt. Siehe auslesenFehler()
57

```

```

58
59 // =====
60 // ===== DOCUMENT READY FUNCTION: INITIALISIERUNG =====
61 // =====
62 $(document).ready(function() {
63     // .....
64     // ..... EVENT-DATEN AUSLESEN UND VORBEREITEN .....
65     // .....
66
67     //Event-Daten auslesen: Liste Produkte auslesen, Liste Kategorien auslesen,
68     //     Liste Tische, EventData
69     //Produkte den Kategorien zuordnen (siehe allgemeineFunktionen.js)
70     var verzogerung = auslesen();
71
72     //Fehler Meldung, falls eines oder mehrere fehlschlagen
73     verzogerung.fail(function(fehler) {
74         console.log("KRITISCHER FEHLER: Laden der Event-Daten fehlgeschlagen: " + fehler);
75
76         var fehlerAnzeige = $("

<p><br>").text("KRITISCHER FEHLER: Laden der Event-Daten fehlgeschlagen: " + fehler);
77         $("#fehler").append(fehlerAnzeige);
78
79         ansicht("fehler");
80     });
81
82     // .....
83     // ..... INITIALISIERUNG .....
84     // .....
85     //beginnen, nachdem Event-Daten geladen und vorbereitet sind
86     verzogerung.done(function() {
87         console.log("Laden des Events erfolgreich: Eventname: " + eventName +
88             ", Anz. Produkte: " + produkte.length +
89             ", Anz. Kategorien: " + kategorien.length +
90             ", Anz. Tische: " + tische.length);
91
92         //AUSLESEN:
93         //alle 7s einmal auslesen (10s ist der Timeout der dateiAufrufen-Funktion)
94         //auslesenFehler() überwacht Fehler und
95         //     bricht Interval ab nach zu vielen aufeinanderfolgenden Fehlern
96         auslesenInterval = setInterval(function() {
97             neueAuslesen();
98
99             //stellt sicher, dass neueAuslesen() nicht zu viele Male in Folge fehlschlägt
100             auslesenErfolg++;
101             if (auslesenErfolg > 5) {
102                 auslesenErfolg = 5;
103             }
104             }, 7000);
105         //schon einmal auslesen
106         neueAuslesen();
107
108         //alles fertig geladen: Ansicht setzen und so auch Ladeanimation verstecken
109         ansicht("inhalt");
110
111     }); //verzogerungAlle.done hier fertig
112 }); //document ready function hier fertig
113
114
115 // =====
116 // ===== FUNKTIONEN =====
117 // =====
118
119 // .....
120 // ..... ALLGEMEIN .....
121 // .....
122 //siehe auch: allgemeineFunktionen.js
123
124 //Ansicht wechseln
125 function ansicht(ansichtNeu) {
126     //Entwicklungs-Fehler-Überprüfung
127     if (typeof(ansichtNeu) !== "string") {
128         cconsole.log("Fehler in ansicht(ansichtNeu)! Angabe kein String:", ansichtNeu);
129         return;
130     }
131
132     $("body").children().hide(); //altes verstecken
133
134     //Ansicht öffnen und nötige Aktionen ausführen
135     switch(ansichtNeu) {
136         case "inhalt":
137             $("#inhalt").show();
138             break;
139
140         case "fehler":
141             $("#fehler").show();
142             break;
143
144         case "laden":
145             $("#laden").show();
146             break;
147
148         default:
149             console.log("Ansicht \"" + ansichtNeu + "\" nicht abgedeckt durch switch.");
150             ansicht("inhalt");
151             break;
152     }
153 }
154
155 //stellt sicher, dass neueAuslesen() nicht zu viele Male nacheinander fehlschlägt
156 //auslesenInterval wurde in Initialisierung aktiviert
157 function auslesenFehler(fehler/* = ""*/, zwingen/* = false*/) {
158     if (fehler === undefined) {
159         fehler = "";
160     }
161     if (zwingen === undefined) {
162         zwingen = false;


```

```

163     }
164
165     auslesenErfolg -= 2; //2 abziehen, weil 1 hinzugefügt wird im interval
166
167     if (auslesenErfolg < 1 || zwingen) { //also =0
168         clearInterval(auslesenInterval);
169
170         ansicht("fehler");
171
172         console.log("KRITISCHER FEHLER: in neueAuslesen(): auslesen zu viele Male fehlgeschlagen: " + fehler);
173
174         var fehlerAnzeige = $("

</p><br>").text(
175             "KRITISCHER FEHLER: in neueAuslesen(): auslesen zu viele Male fehlgeschlagen: " + fehler);
176         $("#fehler").append(fehlerAnzeige);
177     }
178
179     console.log("auslesenInterval fehlgeschlagen. Zähler (kritischer Fehler bei 0): " + auslesenErfolg);
180
181     return auslesenErfolg;
182 }
183
184 // .....
185 // ..... BESTELLUNGEN AUSLESEN .....
186 // .....
187 //neue Bestellungen auslesen und anzeigen, falls sie nicht schon sind
188 //falls nichts zum Anzeigen, Bestellung direkt als verarbeitet markieren
189 function neueAuslesen() {
190     var verzogerung = auslesenUnverarbeitet(zeigeKategorien, function(bestellungen) {
191         //Bestellungen, die nicht angezeigt werden müssen, unten als verarbeitet markieren
192         var unwichtig = [];
193
194         var neuId = []; //neu angezeigte IDs, nur für console.log()
195
196         //bestellungen: Bestellungen, deren verarbeitet-Array
197         // nicht nur '1' enthält (min noch ein '0')
198         for(var i = 0; i < bestellungen.length; i++) {
199             var bestellung = bestellungen[i];
200
201             //nur falls nicht schon angezeigt
202             if (!angezeigteBestellungen.includes(bestellung.id)) {
203                 var wichtig = bestellungAnzeigenObj(bestellung); //Bestellung anzeigen
204                 if (wichtig) {
205                     //Bestellungs-Id zu angezeigten hinzufügen
206                     angezeigteBestellungen.push(bestellung.id);
207                     neuId.push(bestellung.id);
208                 } else {
209                     //falls keine Relevanten Produkte in der Bestellung:
210                     // sofort als verarbeitet markieren
211                     unwichtig.push(bestellung.id);
212                 }
213             }
214         }
215
216         console.log("neueAuslesen(): neu angezeigte IDs:", neuId);
217
218         //unwichtige Bestellungen direkt als verarbeitet markieren
219         if (unwichtig.length > 0) {
220             //Daten zum Updaten erstellen
221             var data = {
222                 id: unwichtig,
223                 verarbeitet: zeigeKategorien
224             };
225             var verzogerungUpdaten = bestellungenUpdaten(data, function(anzZeilen) {
226                 if (anzZeilen < 1) { //sollte nie vorkommen
227                     verzogerungUpdaten.reject("FEHLER:
228                         0 Zeilen aktualisiert in DB. Bestellungen existierten vermutlich nicht (mehr) in DB.");
229                     return; //FEHLER, HIER ABBRECHEN
230                 }
231
232                 console.log("neueAuslesen(): DIREKT als verarbeitet markiert:", unwichtig);
233             }); //hier fertig: Callback bestellungenUpdaten
234             //falls Fehler:
235             verzogerungUpdaten.fail(function(fehler) {
236                 ansicht("fehler");
237
238                 console.log("KRITISCHER FEHLER: Direktes Updaten der Bestellungen (" + unwichtig + ") fehlgeschlagen: " +
239                     fehler);
240
241                 var fehlerAnzeige = $("

</p><br>").text("KRITISCHER FEHLER: Updaten der Bestellungen (" +
242                     JSON.stringify(unwichtig) + ") fehlgeschlagen: " + fehler);
243                 $("#fehler").append(fehlerAnzeige);
244             });
245         }
246     });
247
248     //falls Fehler:
249     verzogerung.fail(function(fehler) {
250         //evt. Interval stoppen (siehe auslesenFehler-Funktion)
251         auslesenFehler(fehler);
252     });
253 }
254
255 //alle Bestellungen aus Ansichten löschen und nochmal auslesen mit neueAuslesen()
256 //ACHTUNG: es wird nicht gespeichert, ob gesehen oder nicht
257 // -> alle erscheinen nochmal in NEU
258 function erneutAuslesen() {
259     //angezeigte Bestellungen Array zurück setzen
260     angezeigteBestellungen = [];
261
262     //alle alten Zeilen löschen (in beiden Tabellen)
263     var tabellen = $("#alteBestellungen, #neueBestellungen").children();
264     tabellen.remove(); //Tabellen löschen
265
266     //auslesen und anzeigen


```

```

264     neueAuslesen();
265 }
266
267
268 // .....
269 // ..... BESTELLUNGEN ANZEIGEN .....
270 // .....
271
272 //eine Bestellung anzeigen (unter neue- und alte-Bestellungen) (keine Animation)
273 //bestellung: Bestellungs-Objekt
274 function bestellungAnzeigenObj(bestellung) {
275     //id zur einfacheren Erreichbarkeit
276     var id = bestellung.id;
277
278     //Tabelle erstellen
279     var tabelle = $("<table id='bestellung_' + id + "'></table>");
280
281     //Caption erstellen (white-space:pre-wrap; ermöglicht mehrere Leerzeichen
282     //     nacheinander. Aber Zeilenumbrüche Spinnen unter Umständen.)
283     var caption = $("<caption class=''><span class='schrift' style='float:left; white-space:pre-wrap;'>Id: " + id +
284     " , Zeit: " + zeitAnzeige(bestellung.zeit) + "<br>(Tisch: " + tische[bestellung.tisch].name + ", Von: " +
285     bestellung.bestelltVon + ")</span></caption>");
286     $(tabelle).append(caption);
287
288     //verarbeitet-Knopf erstellen
289     var verarbeitet = $("<span class='verarbeitet schrift rahmen'>verarbeitet</span>");
290     verarbeitet.attr("onclick", "verarbeitet(" + id + ")");
291     $(caption).append(verarbeitet);
292
293     //evt. ganze Bestellung nicht anzeigen, falls Bestellung leer
294     //     (d.h. kein Produkt soll angezeigt werden) (siehe unten)
295     var bestellungLeer = true;
296
297     //Zeilen erstellen für Produkte und Kategorien
298     for(var i = 0; i < kategorien.length; i++) {
299         //Kategorie (und Produkte darin) überspringen,
300         //     falls sie nicht angezeigt werden soll
301         if (zeigeKategorien.includes(i) == false) {
302             continue;
303         }
304
305         //falls eine Kategorie aus irgendeinem Grund schon verarbeitet
306         //     Das sollte eigentlich nie vorkommen,
307         //     solange keine Überlagerungen in zeigeKategorien der Basen
308         //trotzdem ganze Bestellung anzeigen
309         if (bestellung.verarbeitet[bestellung.verarbeitet.length - 1 - i] == true) {
310             console.log("ANMERKUNG: Kategorie schon verarbeitet (Bestellung: " + id + ", betreffende Kategorie: " + i +
311             "). Trotzdem ganze Bestellung angezeigt.");
312         }
313
314         //Zeile für Kategorie (enthält trotzdem auch Menge-Feld, wegen Layout)
315         var zeileKategorie = $("<tr class='kategorie'><td class='feldName'>" + kategorien[i].name +
316         "</td><td class='feldMenge'></td></tr>");
317         $(tabelle).append(zeileKategorie); //Zeile an Tabelle pappen
318
319         //evt. Zeile für Kategorie wieder wegnehmen (siehe unten)
320         var kategorieLeer = true;
321
322         for(var k = 0; k < kategorien[i].produkte.length; k++) {
323             var produkt = kategorien[i].produkte[k]; //ID des Produkts
324
325             var menge = bestellung.bestellt[produkt];
326             if (menge == 0) { //Überspringen, falls kein solches Produkt bestellt ist
327                 continue;
328             }
329
330             //mindestens einmal bis hier gekommen: Kategorie & Bestellung nicht leer
331             kategorieLeer = false;
332             bestellungLeer = false;
333
334             //Zeile für Produkt erstellen
335             var zeileProdukt = $("<tr id='_' + id + "_p" + produkt +
336             "' class='produkt' onclick='abhaken($ (this))'><td class='feldName'>" + produkte[produkt].name +
337             "</td><td class='feldMenge'>" + menge + "</td></tr>"); //Name, Menge
338             $(tabelle).append(zeileProdukt); //Zeile an Tabelle pappen
339         }
340
341         //Kategorie-Zeile wieder wegnehmen, falls kein Produkt dieser Kategorien
342         if (kategorieLeer) {
343             zeileKategorie.remove();
344         }
345     }
346
347     //abbrechen und nicht auf Seite pappen falls Bestellung leer
348     // (d.h. kein Produkt soll angezeigt werden, nicht dass kein Produkt bestellt wurde!)
349     if (bestellungLeer) {
350         return false;
351     }
352
353     //ALT-Seite
354     //Kopie von Tabelle anhängen an ALTE Bestellungen
355     $("#alteBestellungen").append(tabelle.clone());
356
357     //NEU-Seite
358     //anspassen für neu
359     verarbeitet.html("gesehen");
360     verarbeitet.attr("onclick", "gesehen(" + id + ")");
361     tabelle.attr("id", "bestellungNeu_" + id)
362
363     //Kopie von Tabelle (mit Änderungen) anhängen an NEUE Bestellungen
364     $("#neueBestellungen").append(tabelle.clone());
365     return true;
366 }
367
368 //Bestellungen als verarbeitet markieren und aus Ansichten löschen

```

```

363 function verarbeitet(id) { //id: ID der Bestellung
364     //Tabellen aus Ansichten löschen
365     var tabellenVerz = new $.Deferred();
366     //beide Tabellen finden
367     var tabellen = $("#bestellung " + id + ", " + "#bestellungNeu " + id);
368     slideUp(tabellen, 200, function() { //Tabellen slideUp-Animation
369         //tabellen.remove(); //nach Animation Tabellen löschen
370         //Tabellen erst löschen, wenn sicher, dass sie nicht mehr gebraucht werden
371         tabellenVerz.done(function() {
372             tabellen.remove(); //Tabellen löschen
373         });
374         tabellenVerz.fail(function() {
375             tabellen.show(); //Tabellen wieder anzeigen (ohne Animation)
376         });
377     });
378
379     //Daten zum Updaten erstellen
380     var data = {
381         id: id,
382         verarbeitet: zeigeKategorien
383     }
384
385     var verzogerung = bestellungenUpdaten(data, function(anzZeilen) {
386         if (anzZeilen < 1) { //sollte nie vorkommen
387             verzogerung.reject(
388                 "FEHLER: 0 Zeilen aktualisiert in DB. Bestellung existiert vermutlich nicht (mehr) in DB.");
389         }
390
391         //ID aus angezeigteBestellungen-Array löschen
392         var index = angezeigteBestellungen.indexOf(id);
393         if (index > -1) {
394             angezeigteBestellungen.splice(index, 1);
395         } else { //sollte nie vorkommen
396             console.log(
397                 "ANMERKUNG: in verarbeitet(id): als verarbeitet zu markierende id (" + id +
398                 ") existierte nicht im angezeigteBestellungen-Array", angezeigteBestellungen);
399
400             tabellenVerz.resolve(); //Tabellen endgültig löschen
401
402             console.log("verarbeitet(): Bestellung " + id + " als verarbeitet markiert");
403         }); //hier fertig: Callback bestellungenUpdaten
404
405         //falls Fehler:
406         verzogerung.fail(function(fehler) {
407             ansicht("fehler");
408
409             console.log("KRITISCHER FEHLER: Updaten der Bestellung (" + id + ") fehlgeschlagen: " + fehler);
410
411             var fehlerAnzeige = $("

<p><br>").text(
412                 "KRITISCHER FEHLER: Updaten der Bestellung (" + id + ") fehlgeschlagen: " + fehler);
413             $("#fehler").append(fehlerAnzeige);
414
415             //Tabellen wieder anzeigen (nur theoretisch, da ja ansicht("fehler"))
416             tabellenVerz.reject();
417         });
418     });
419
420     //gesehen-Knopf: Bestellung aus #neueBestellungen löschen
421     //id: ID der Bestellung
422     function gesehen(id) {
423         //Tabelle finden, zu welcher der Knopf gehört
424         var tabelle = $("#bestellungNeu " + id);
425         slideUp(tabelle, 200, function() { //Tabelle slideUp-Animation
426             tabelle.remove(); //nach Animation Tabelle löschen
427         });
428     }
429
430     // .....
431     // ..... EINZELNE PRODUKTE ABHACKEN .....
432     // .....
433     //einzelne Produkte-Zeilen abhacken
434     function abhaken(zeile) {
435         zeile.addClass("ausgeliefert"); //Style ändern (graue Schrift)
436     }
437 }


```

## 9. satellit.css

```

1 /*
2  * Datei: satellit.css
3  * Projekt: Kellner App
4  * Autor: Michael Heider
5  * Datum: HS 2017
6  *
7  * Was:
8  * CSS zu Satellit
9  */
10
11
12 /* =====
13 / ===== ALLGEMEINE KLASSEN =====
14 / =====*/
15
16 /* ALLGEMEINES */
17 /*siehe auch in gemeinsam.css*/
18
19 /* SEITENSTRUKTUR */
20 .header {

```

```

21     position: fixed;
22     width: 100%;
23     height: 58px;
24     top: 0%;
25 }
26
27 .footer {
28     position: fixed;
29     width: 100%;
30     height: 58px;
31     bottom: 0%;
32 }
33
34 .mitte {
35     position: absolute;
36     width: 100%;
37     top: 58px; /*alles zwischen 55px von oben und 55px von unten ist #allekacheln*/
38     bottom: 58px;
39
40     overflow: auto;
41 }
42
43 .rechteck { /*für footer*/
44     width: 20%;
45     height: 100%;
46     float: right;
47
48     text-align: center;
49 }
50
51 .kachel {
52     width: 33.33333333%;
53     height: 93px; /*entspricht vier Linien Text
54     (4 Linien * 16px Schriftgröße * ca. 120% line-height
55     + 2 * 8Px padding + rahmen (falls box-sizing: border-box;))*/
56
57     float: left;
58
59     box-sizing: border-box;
60     border-style: solid;
61     border-width: 0px 1px 1px 0px; /*top right bottom left*/
62     border-color: black;
63
64     word-wrap: break-word; /*Wörter Teilen auch im Wort*/
65     overflow: auto; /*scrollbar anzeigen, falls Text zu lang*/
66 }
67
68
69 /* =====
70 /  ===== SPEZIFISCH FÜR ANSICHTEN =====
71 /  =====*/
72 /* .....
73 / ..... TISCHE .....
74 / .....*/
75 .tisch {
76     background-color: yellow;
77 }
78
79
80 /* .....
81 / ..... EINZELNER TISCH: BESTELLUNGEN .....
82 / .....*/
83 /*48px Höhe: entspricht zwei Linien Text
84 2 Linien * 16px Schriftgröße * ca. 120% line-height +
85 2 * 4Px padding + rahmen (falls box-sizing: border-box;))*/
86
87 .bestellungCaption {
88 }
89
90 .ausgeliefertStatusString {
91     font-weight: normal;
92     color: green;
93 }
94
95 .bestellungInfo {
96     float: left;
97
98     line-height: 200%;
99 }
100
101 .bestellungAuswählen {
102     float: right;
103
104     margin-top: 20px; /*pseudo vertikales Zentrieren*/
105
106     text-align: center;
107 }
108
109 label:before {
110     bottom: -14px; /*vertikaler Offset: ACHTUNG: auf PC und Handy allenfalls verschieden*/
111 }
112
113 .hacken { /*gleiche Höhe und font-size wie Checkbox*/
114     width: 40px;
115     height: 40px;
116     line-height: 40px;
117
118     font-size: 30px;
119 }
120
121 /*oberer Rahmen (zweites Kind, nach caption (beginnt mit 1))*/
122 tr:nth-child(2).kategorie {
123     border-top: 1px solid black;
124 }

```

```

125
126 .fieldName {
127 }
128
129 .feldMenge {
130     width: 60px;
131
132     text-align: right;
133
134     /*text-overflow, so dass klar, dass es nnoch weiter ginge*/
135     text-overflow: "..."; /*wie ellipsis, aber eigener String*/
136 }
137
138 #zuBezahlen {
139     background-color: green;
140 }
141
142
143 /* .....
144 / ..... BEZAHLEN .....
145 / .....*/
146
147 /* ..... BEZAHLEN AUSWÄHLEN .....*/
148 #bezahlenHeader {
149     position: fixed; /*bleibt oben an Tabelle, bzw scrollt nicht weg*/
150     z-index: 5; /*vor Tabelle*/
151     background-color: white; /*Tabelle nicht sichtbar darunter*/
152
153     width: 100%;
154 }
155
156 #bezahlenHeader > p {
157     width: 50%;
158     float: left;
159
160     text-align: center;
161     font-weight: bold;
162 }
163
164 #allesSchiebenNein {
165     width: 30px;
166     float: right;
167 }
168 #allesSchiebenJa {
169     width: 30px;
170     float: left;
171 }
172
173 #bezahlen {
174     background-color: green;
175 }
176
177 #storno {
178     background-color: red;
179 }
180
181 .bezahlenHalfte {
182     position: relative;
183     top: 36px; /*36px Platzlassen für #bezahlenHeader*/
184
185     width: 50%;
186     float: left;
187 }
188
189 /* Tabellen Felder */
190 .bezahlenMenge {
191     width: 35px;
192
193     text-align: right;
194 }
195
196 .bezahlenName {
197     text-align: left;
198 }
199
200
201 /* ..... QUITTUNG .....*/
202 .quittungZeile {
203     border-width: 0px;
204 }
205
206 .quittungTotal {
207     height: 100%;
208
209     float: right;
210 }
211
212
213 /* .....
214 / ..... BESTELLEN .....
215 / .....*/
216
217 /* ..... KACHELN .....*/
218
219 /* ELEMENTE IN STRUKTUR */
220 .kategorieNavigation {
221     background-color: orange;
222
223     width: 25%;
224     height: 100%;
225
226     float: left;
227     text-align: center;
228 }

```

```
229
230 .jetzigeKategorie {
231     background-color: #FCF67D;
232
233     width: 50%;
234     height: 100%;
235
236     float: left;
237     text-align: center;
238 }
239
240 .fertig {
241     background-color: #93D07F;
242
243     width: 25%;
244     height: 100%;
245
246     float: left;
247     text-align: center;
248 }
249
250 #allekacheln { /*funktional: alles darin verstecken*/
251     width: 100%;
252     height: 100%;
253
254     box-sizing: border-box;
255     border-style: solid;
256     border-width: 0px 1px 0px 2px; /*top right bottom left*/
257     border-color: black;
258 }
259
260 #subAngaben {
261     width: 80%;
262     height: 100%;
263
264     float: left;
265 }
266
267 #abbestellen {
268     float: left;
269
270     background-color: red;
271 }
272 .modusBestellen {
273 }
274 .modusAbbestellen {
275     border-width: 3px;
276     border-color: #ffdb00;
277 }
278
279
280 /* CACHELN */
281 /*.kachel siehe unter: allgemeine Klassen --> Strukturelemente*/
282 /*siehe auch in gemeinsam.css*/
283
284 .produktMenge {
285     float: right;
286 }
287
288
289 /* ..... LISTE .....*/
290
291 /* ELEMENTE IN STRUKTUR */
292 #liste {
293     width: 100%;
294     height: 100%;
295 }
296
297 #listeTotalPreis {
298     height: 100%;
299
300     float: right;
301 }
302
303 /* TABELLEN FELDER */
304 .menge {
305     width: 35px;
306
307     text-align: right;
308 }
309
310 .name {
311     text-align: left;
312 }
313
314 .afeld {
315     width: 17px;
316
317     text-align: right;
318 }
319
320 .preis {
321     width: 57px;
322     padding-right: 8px;
323
324     text-align: right;
325 }
```

## 10. satellit.html

```

1  <!--
2  < Datei: satellit.html
3  < Projekt: Kellner App
4  < Autor: Michael Heider
5  < Datum: HS 2017
6  <
7  < Was:
8  < HTML zu Satellit.
9  < Enthält Grund-HTML für alle Ansichten (jeweils nur eines sichtbar).
10 < Alles, was EventDaten (zum Beispiel Produktnamen) verwendet,
11 < wird dynamisch erzeugt von satellit.js.
12 -->
13
14
15 <!DOCTYPE html>
16 <html>
17 <head>
18 <meta charset="utf-8"/>
19
20 <title>Kellner App Satellit</title>
21
22
23 <!--=====
24 < ===== HEAD BEREICH =====
25 < =====>
26
27 <!-- Korrekte Darstellung/Einstellung auf Smartphones -->
28 <meta name="viewport" content="width=device-width, initial-scale=1.0, user-scalable=no">
29
30
31 <!--.....
32 < ..... ZEUG FÜR WEB-APP .....
33 < .....>
34
35 <!--..... ANDROID .....-->
36 <!-- (ACHTUNG: MUSS OBEN AN APPLE TAGS SEIN!!! (wieso??))-->
37 <link rel="manifest" href="webapp/manifestSatellit.json">
38
39 <!--..... IOS .....-->
40 <!-- ist eine Web App -->
41 <meta name="apple-mobile-web-app-capable" content="yes"/>
42 <!-- Statusbar Farbe und Stil -->
43 <meta name="apple-mobile-web-app-status-bar-style" content="black"/>
44 <!-- Icon auf Homescreen -->
45 <link rel="apple-touch-icon" sizes="180x180" href="webapp/icon180x180.png">
46 <!-- Icon während Ladevorgang (ca. 1900x1000 Pixel) -->
47 <link rel="apple-touch-startup-image" href="webapp/startup1900x1000.png">
48 <!-- Text auf Homescreen (anstatt Text aus title Tag) -->
49 <meta name="apple-mobile-web-app-title" content="KellnerApp Satellit"/>
50
51 <!--..... SERVICE-WORKER .....-->
52 <!-- ServiceWorker (braucht es einfach für Web Apps) -->
53 <!-- Code leicht bearbeitet von URL: (11.8.2017)
54 < https://developers.google.com/web/fundamentals/getting-started/primers/service-workers
55 -->
56 <script type="text/javascript">
57   if ("serviceWorker" in navigator) {
58     window.addEventListener("load", function() {
59       navigator.serviceWorker.register("sw.js").then(function(registration) {
60         //Registrierung war erfolgreich
61         console.log("ServiceWorker-Registrierung erfolgreich. Scope: ",
62           registration.scope);
63       }, function(fehler) {
64         //Registrierung fehlgeschlagen
65         console.log("ServiceWorker-Registrierung fehlgeschlagen: ", fehler);
66       });
67     });
68   }
69 </script>
70
71
72 <!--.....
73 < ..... CSS .....
74 < .....>
75 <!-- Stylesheets einbinden -->
76 <link rel="stylesheet" type="text/css" href="stylesheets/reset.css">
77 <link rel="stylesheet" type="text/css" href="stylesheets/gemeinsam.css">
78 <link rel="stylesheet" type="text/css" href="satellit.css">
79
80
81 <!--.....
82 < ..... JAVASCRIPT .....
83 < .....>
84 <!-- Scripts einbinden -->
85 <script type="text/javascript" src="scripts/polyfill.js"></script>
86 <script type="text/javascript" src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
87 <script type="text/javascript" src="scripts/allgemeineFunktionen.js"></script>
88 <!-- Hauptsript (auszuführendes Script) -->
89 <script type="text/javascript" src="satellit.js"></script>
90
91
92 </head>
93
94
95 <!--=====
96 < ===== BODY BEREICH =====
97 < =====>
98 <body>
99
100 <!--..... LADEN .....-->
101 <div id="laden">

```

```

102     <span>LADEN</span>
103 </div>
104
105 <!--..... FEHLER .....-->
106 <div id="fehler" style="display:none;">
107     <p><b>KRITISCHE FEHLER:</b></p><br>
108 </div>
109
110
111 <!--..... TISCHE ÜBERSICHT .....-->
112 <!-- ..... TISCHE ÜBERSICHT ..... -->
113 <div id="tischeÜbersicht" style="display:none;">
114     <!-- HEADER -->
115     <section class="header rahmen">
116         <div class="kategorieNavigation rahmen schrift" onclick=""></div>
117         <div class="jetzigeKategorie rahmen grosseschrift">Tische</div>
118         <div class="fertig rahmen schrift" onclick=""></div>
119     </section>
120
121     <!-- MITTE -->
122     <section class="mitte">
123         <div id="alleTische"></div> <!--Tische mit JavaScript-->
124     </section>
125
126     <!-- FOOTER -->
127     <section class="footer rahmen">
128
129     </section>
130 </div>
131
132
133 <!--..... EINZELNER TISCH .....-->
134 <!-- ..... (EINZELNER) TISCH ..... -->
135 <div id="tisch" style="display:none;">
136     <!-- HEADER -->
137     <section class="header rahmen">
138         <div class="kategorieNavigation rahmen schrift" onclick="ansicht('tischeÜbersicht')">zurück</div>
139         <div id="jetzigerTisch" class="jetzigeKategorie rahmen grosseschrift">INITIAL_jetzigerTisch</div>
140         <div class="fertig rahmen schrift" onclick="neueBestellung()">neue Bestellung</div>
141     </section>
142
143     <!-- MITTE -->
144     <section class="mitte">
145         <!--Bestellungen mit JavaScript, jedes Mal neu erzeugt-->
146         <div id="alleBestellungen"></div>
147     </section>
148
149     <!-- FOOTER -->
150     <section class="footer rahmen">
151         <div id="zuBezahlen" class="rechteck rahmen schrift" onclick="zuBezahlen()">bezahl<br>storno</div>
152     </section>
153 </div>
154
155 <!-- ..... BEZAHLEN ..... -->
156 <div id="bezahlen" style="display:none;">
157     <!-- HEADER -->
158     <section class="header rahmen">
159         <div class="kategorieNavigation rahmen schrift" onclick="ansicht('tisch')">zurück</div>
160         <div class="jetzigeKategorie rahmen grosseschrift">bezahlen</div>
161         <div class="fertig rahmen schrift" onclick="quittungAnzeigen(bezahlenJa)">Quittung</div>
162     </section>
163
164     <!-- MITTE -->
165     <section class="mitte">
166         <div id="bezahlenHeader" class="rahmenUnten">
167             <!--bezahlenNein/Ja-Seite Titel und Knopf zum alles schieben-->
168             <!--bezahlenNein-Seite-->
169             <p>nicht bezahlen<span id="allesSchiebenNein" class="rahmen"
170                 onclick="bezahlenSchiebenAlle(true)"></span></p>
171             <!--bezahlenJa-Seite-->
172             <p><span id="allesSchiebenJa" class="rahmen" onclick="bezahlenSchiebenAlle(false)"></span>zu bezahlen</p>
173         </div>
174         <!--die beiden Hälften sind direkt Tables, Produkte-Rows mit JavaScript-->
175         <table id="bezahlenNein" class="bezahlenHalfte rahmen"></table>
176         <table id="bezahlenJa" class="bezahlenHalfte rahmen"></table>
177     </section>
178
179     <!-- FOOTER -->
180     <section class="footer rahmen">
181         <!--on double click!-->
182         <div id="storno" class="rechteck rahmen schrift" onclick="bezahlen(true)">storno<br>(doppel)</div>
183     </section>
184 </div>
185
186 <!-- ..... QUITTUNG ..... -->
187 <div id="quittung" style="display:none;">
188     <!-- HEADER -->
189     <section class="header rahmen">
190         <div class="kategorieNavigation rahmen schrift" onclick="ansicht('bezahlen')">zurück</div>
191         <div class="jetzigeKategorie rahmen grosseschrift">Quittung</div>
192         <!--on double click!-->
193         <div class="fertig rahmen schrift" onclick="bezahlen()">bezahl<br>(doppel)</div>
194     </section>
195
196     <!-- MITTE -->
197     <section class="mitte">
198         <table id="quittungAlle"></table> <!--Produkte-Rows mit JavaScript-->
199     </section>
200
201     <!-- FOOTER -->
202     <section class="footer rahmen">
203         <var id="quittungTotalPreis" class="quittungTotal schrift"></var>
204         <span class="quittungTotal schrift">Total (inkl. Mwst.):</span>
205     </section>
206 </div>

```

```

206
207
208 <!--..... BESTELLEN .....-->
209 <!-- ..... KACHELN ..... -->
210 <div id="kacheln" style="display:none;">
211 <!-- HEADER -->
212 <section class="header rahmen">
213 <!--nur eine der beiden Navigationen gleichzeitig sichtbar-->
214 <div id="navigationKategorie" class="kategorieNavigation rahmen schrift"
      onclick="kategorieOffnen(-1)">Kategorien</div>
215 <div id="navigationAbbrechen" class="kategorieNavigation rahmen schrift"
      onclick="bestellungAbbrechen()">abbrechen</div>
216 <div id="jetzigeKategorie" class="jetzigeKategorie rahmen grosseschrift">INITIAL_jetztigeKategorie</div>
217 <div class="fertig rahmen schrift" onclick="ansicht('liste')">Weiter</div>
218 </section>
219
220 <!-- MITTE -->
221 <section class="mitte">
222 <!--Produkt- und Kategorie-Kacheln mit JavaScript-->
223 <div id="allekacheln"></div>
224 </section>
225
226 <!-- FOOTER -->
227 <section class="footer rahmen">
228 <p id="subAngaben" class="rahmen">
229 <var id="subTotal">INITIAL_Preis</var>
230 </p>
231 <!--Anfangszustand: class modusBestellen (abbestellen = true/false)-->
232 <div id="abbestellen" class="rechteck rahmen schrift modusBestellen" onclick="modusAbbestellen()">löschen</div>
233 </section>
234 </div>
235
236
237 <!-- ..... LISTE ..... -->
238 <div id="liste" style="display:none;">
239 <!-- ..... HEADER ..... -->
240 <section class="header rahmen">
241 <div class="kategorieNavigation rahmen schrift" onclick="ansicht('kacheln')">zurück</div>
242 <div class="jetzigeKategorie rahmen grosseschrift">Liste</div>
243 <div class="fertig rahmen schrift" onclick="bestellungBestatigen(jetzigerTisch)">Bestätigen</div>
244 </section>
245
246 <!-- MITTE -->
247 <section class="mitte">
248 <!--Produkt- und Kategorie-Einträge mit JavaScript-->
249 <div id="allelisten"></div>
250 </section>
251
252 <!-- FOOTER -->
253 <section class="footer rahmen">
254 <div id="totalPreis" class="schrift"></div>
255 </section>
256 </div>
257
258 </body>
259 </html>

```

## 11. satellit.js

```

1  /*
2  * Datei: satellit.js
3  * Projekt: Kellner App
4  * Autor: Michael Heider
5  * Datum: HS 2017
6  *
7  * Was:
8  * Javascript-Logik Satellit
9  */
10
11 "use strict"; //strict mode aktivieren
12
13
14 //Warnung einblenden bevor schliessen oder versehentlichem Neuladen
15 // (speziell auf Handy durch runterziehen)
16 //auf einigen Browser muss zuerst mit der Webseite interagiert worden sein,
17 // bevor die Meldung auftritt
18 window.addEventListener("beforeunload", function(e) {
19 //Nachricht wird nicht auf allen Browsern angezeigt, sondern irgendein Default
20 var nachricht = "Wenn Sie die App neu laden, müssen Sie sich erneut einloggen.";
21 e.returnValue = nachricht; //zwei return Zeilen für umfassenderen Browser-Support
22 return nachricht;
23 });
24
25
26 // =====
27 // ===== VARIABLEN, OBJEKTE, OBJECT CONSTRUCTORS, ETC. =====
28 // =====
29 // ..... IDENTIFIKATION .....
30 var kellnerId = "Satellit 1";
31 var eventName; //wird ausgelesen in auslesen()
32
33 // ..... NEUE BESTELLUNG .....
34 //Object-Constructors siehe allgemeineFunktionen.js
35
36 var tische = []; //alle Tische. Position in array stimmt mit tisch.id überein
37 //siehe auch ErstelleTisch-constructor (allgemeineFunktionen.js)
38
39 var tischeDiv = []; //alle Tische als Divs
40
41 var jetzigerTisch = -1; //Tisch, der zurzeit angezeigt wird (-1 für Tische Übersicht)

```

```

42
43 // ..... TISCH .....
44 var hackenFelder = []; //alle Hackenfelder als jQuery
45 var bestellungen = []; //gerade angezeigte Bestellungen als Objekte
46
47 //siehe auch ErstelleBestellung-Constructor (allgemeineFunktionen.js)
48
49 // ..... BEZAHLEN .....
50 //Format wie bestellt-Array
51 var bezahlenNein = []; //wie viel von welchem Produkt auf bezahlenNein-Seite ist
52 var bezahlenJa = []; //wie viel von welchem Produkt auf bezahlenJa-Seite ist
53
54 var bestellungenBezahlen = []; //Referenzen zu Bestellungen-Objekten von bestellungen
55 //nur während bezahlen- und Quittung-Ansicht gefüllt, sonst leer
56
57 //siehe auch bestellungen-Array unter Tisch
58
59 // ..... BESTELLEN .....
60 var produkte = []; //alle Produkte. Position in Array stimmt mit produkt.id überein
61 //siehe auch ErstelleProdukt-Constructor (allgemeineFunktionen.js)
62 var kategorien = []; //alle Kategorien. Position in Array stimmt mit kategorie.id überein
63 //siehe auch ErstelleKategorie-Constructor (allgemeineFunktionen.js)
64
65 var produkteDiv = []; //alle Produkte als divs
66 var kategorienDiv = []; //alle Kategorien als divs
67 var mengeDiv = []; //div: in Produkt-Kacheln wird Menge angezeigt
68
69 var abbestellen = false; //normaler/abbestellen Modus
70 //normales Modus: false, abbestellen Modus: true
71
72 var jetzigeKategorie = -1; //id der Kategorie, in der man zurzeit ist.
73 //-1 für Kategorien-Übersicht (name wird oben in #jetzigeKategorie angezeigt)
74
75 var bestellt = []; //wie viel von jedem Produkt bestellt ist (0 für keins davon bestellt).
76 //Länge entspricht produkte.length
77
78 var totalPreis = 0; //in Franken
79
80
81 // =====
82 // ===== DOCUMENT READY FUNCTION: INITIALISIERUNG =====
83 // =====
84 $(document).ready(function() {
85 // .....
86 // ..... EVENT-DATEN AUSLESEN UND VORBEREITEN .....
87 // .....
88
89 //Event-Daten auslesen: Liste Produkte auslesen, Liste Kategorien auslesen,
90 // Liste Tische, eventData
91 //Produkte den Kategorien zuordnen (siehe allgemeineFunktionen.js)
92 var verzogerung = auslesen();
93
94 //Fehler Meldung, falls eines oder mehrere fehlschlagen
95 verzogerung.fail(function(fehler) {
96 console.log("KRITISCHER FEHLER: Laden der Event-Daten fehlgeschlagen: " + fehler);
97
98 var fehlerAnzeige = $("

</p><br>").text(
99 "KRITISCHER FEHLER: Laden der Event-Daten fehlgeschlagen: " + fehler);
100 $("#fehler").append(fehlerAnzeige);
101
102 ansicht("fehler");
103 });
104
105 // .....
106 // ..... INITIALISIERUNG .....
107 // .....
108 //beginnen, nachdem Event-Daten geladen und vorbereitet sind
109 verzogerung.done(function() {
110 console.log("Laden des Events erfolgreich: Eventname: " + eventName +
111 " , Anz. Produkte: " + produkte.length +
112 " , Anz. Kategorien: " + kategorien.length +
113 " , Anz. Tische: " + tische.length);
114
115
116 // ..... NEUE BESTELLUNGEN .....
117 //divs für Tische produzieren
118 for (var i=0; i < tische.length; i++) {
119 var div = $("<div id='tisch' + tische[i].id +
120 " " class='kachel tisch schrift' onclick='tischOffnen(" +
121 tische[i].id + ")'>" + tische[i].name + "</div>");
122 tischeDiv.push(div);
123 $("#alleTische").append(div); //div auf Webseite pappen
124 }
125
126
127 // ..... BEZAHLEN .....
128 //Arrays auf richtige Länge setzen
129 bezahlenJa = new Array(produkte.length);
130 bezahlenJa.fill(0);
131 bezahlenNein = new Array(produkte.length);
132 bezahlenNein.fill(0);
133
134 //zwei Tabellen erstellen: für ja- und nein-Seite
135 for (var m=0; m < 2; m++) {
136 var jaNein;
137 if (m === 0) {
138 jaNein = "Ja";
139 } else {
140 jaNein = "Nein";
141 }
142
143 //Funktionsargument für onclick-Funktionen:
144 // "true"/"false" aufgrund von jaNein festlegen
145 var funktionArgument = (jaNein.toUpperCase() === "JA" ? "false" : "true");


```

```

146
147     var tabelle = $("#bezahlen" + jaNein); //Tabelle finden
148
149     //erste Zeile für Breite der Spalten
150     //(kann NICHT display:hidden haben, also pseudoVersteckt (inkl. Felder))
151     var fixZeile = $("|
|  |

```

```

245 //divs und tables für Liste erstellen. Name und Preis schon einfüllen.
246 //Menge nicht, sie wird aktualisiert in listeAktualisieren()
247
248 //für jede Kategorie
249 for (var i=0; i < kategorien.length; i++) {
250 //Div für Kategorie und zugehörige Tabelle.
251 var kategorieDiv = $("<div id='listeKategorie' + kategorien[i].id +
    " class='listeKategorie rahmenUnten kategorie'></div>");
252
253 var tabelle = $("<table></table>");
254
255 //caption für Tabelle (Kategorienname) erzeugen (Kategorie Namen fix)
256 var caption = $("<caption class='schrift rahmenUnten'>" + kategorien[i].name + "</caption>");
257 $(tabelle).append(caption); //caption an Tabelle pappen
258
259 //Zeilen für Tabelle erstellen: für jedes Produkt k, das zur Kategorie i gehört
260 for (var k=0; k < kategorien[i].produkte.length; k++) {
261 //Zeile erstellen
262 var zeile = $("<tr class='produkt'></tr>");
263 $(tabelle).append(zeile);
264
265 //Sachen von Produkt k auslesen
266 var id = kategorien[i].produkte[k];
267 var name = produkte[id].name;
268 var preis = produkte[id].preis;
269
270 //Felder für Zeile erstellen
271 //Menge wird in listeAktualisieren() aufgrund der CSS-ID aktualisiert
272 var mengeFeld = $("<td class='menge' id='listeMenge' + id + "'></td>");
273 var nameFeld = $("<td class='name'>" + name + "</td>");
274 var aFeld = $("<td class='afeld'>&lt;/td>");
275 var preisFeld = $("<td class='preis'>" + preisAnzeige(preis, false) + "</td>");
276
277 //alle Felder in Zeile einfüllen
278 $(zeile).append(mengeFeld, nameFeld, aFeld, preisFeld);
279 }
280
281 $(kategorieDiv).append(tabelle); //Tabelle an kategorieDiv pappen
282 $("#allelisten").append(kategorieDiv); //kategorieDiv auf Webseite pappen
283 }
284
285 //Liste schon einmal aktualisieren (versteckt leere Zeilen)
286 listeAktualisieren();
287
288 // ..... WEITERE INITIALISIERUNG .....
289 //alles fertig geladen: Ansicht setzen und so auch Ladeanimation verstecken
290 ansicht("tischeUbersicht");
291
292 }); //verzogerungAlle.done hier fertig
293 }); //document ready function hier fertig
294
295
296 // =====
297 // ===== FUNKTIONEN =====
298 // =====
299
300 // .....
301 // ..... ALLGEMEIN .....
302 // .....
303 //siehe auch: allgemeineFunktionen.js
304
305 //Ansicht wechseln
306 //bei "tischeUbersicht" einige Sachen zurücksetzen
307 //sonst allerdings möglichst nichts, weil mehrere Wege auf eine Ansicht
308 function ansicht(ansichtNeu) {
309 //Entwicklungs-Fehler-Überprüfung
310 if (typeof(ansichtNeu) != "string") {
311 console.log("Fehler in ansicht(ansichtNeu)! Angabe kein String:", ansichtNeu);
312 return;
313 }
314
315 $("body").children().not("#fehler").hide(); //Altes verstecken, aber nicht Fehler
316 laden(false); //Ladeanimation sicher ausblenden
317
318 //Ansicht öffnen und nötige Aktionen ausführen (möglichst wenige)
319 switch(ansichtNeu) {
320 case "tischeUbersicht":
321 //Sachen zurücksetzen
322 jetzigerTisch = -1;
323 hackenFelder = [];
324 bestellungen = [];
325 bestellungenBezahlen = [];
326
327 $("#tischeUbersicht").show();
328 break;
329
330 case "tisch":
331 bestellungenBezahlen = [];
332
333 $("#tisch").show();
334 break;
335
336 case "bezahlen":
337 $("#bezahlen").show();
338 break;
339
340 case "quittung":
341 $("#quittung").show();
342 break;
343
344 case "kacheln":
345 kategorieOffnen(-1); //Ober-Kategorie öffnen
346 $("#subTotal").text(preisAnzeige(totalPreis, true));
347 modusAbbestellen(false); //abbestellen Modus ausschalten
348

```

```

349         $("#kacheln").show();
350         break;
351
352     case "liste":
353         listeAktualisieren();
354
355         $("#liste").show();
356         break;
357
358     case "fehler":
359         $("#fehler").show();
360         break;
361
362     case "laden":
363         $("#laden").show();
364         break;
365
366     default:
367         console.log("Ansicht \" + ansichtNeu + "\" nicht abgedeckt durch switch.");
368         ansicht("tischeÜbersicht");
369         break;
370     }
371 }
372
373 //Ladebildschirm ein-/ausblenden: nur überlagern, ansicht bleibt
374 //stat (bool): Ladebildschirm ein-/ausblenden
375 function laden(stat/* = false*/) {
376     if (stat === undefined) {
377         stat = false;
378     }
379
380     if (stat) {
381         $("#laden").show()
382     } else {
383         $("#laden").hide()
384     }
385 }
386
387
388 // .....
389 // ..... EINZELNER TISCH .....
390 // .....
391
392 //alle Bestellungen eines Tisches (tisch.id) in Divs packen
393 //tisch: ID des Tisches
394 function tischOffnen(tisch) {
395     //Fehlerüberprüfung (normalerweise jetzigerTisch = -1 falls kein Tisch angezeigt)
396     if (tisch < 0) {
397         console.log("FEHLER: in tischOffnen(tisch): tisch kann nicht < 0 sein: " + tisch +
398             "\nKehre zu Hauptansicht zurück");
399         ansicht("tischeÜbersicht");
400         return;
401     }
402
403     laden(true); //Ladebildschirm
404
405     bestellungen = []; //alte Bestellungen sicher löschen
406     hackenFelder = []; //alte Hackenfelder sicher löschen
407
408     //jetziger Tisch Anzeige updaten
409     jetzigerTisch = tisch;
410     $("#jetzigerTisch").text("Tisch " + tische[tisch].name); //Tisch Text aktualisieren
411
412     //alte divs löschen
413     $("#alleBestellungen").children().remove(); //alle alten Bestellungen-Anzeigen löschen
414
415     //aktuelle Bestellungen auslesen. Callback-Funktion ausführen, sobald auslesen fertig
416     //bestellungenHier: Bestellungen auf Tisch tisch
417     var verzögerung = auslesenTisch(tisch, function(bestellungenHier) {
418         //Bestellungs-Objekte speichern
419         bestellungen = bestellungenHier;
420
421         //Bestellungen nacheinander anzeigen
422         for (var m = 0; m < bestellungenHier.length; m++) {
423             //Bestellung dieses Schleifendurchlaufsfestlegen
424             var bestellung = bestellungenHier[m];
425
426             //falls vollständig ausgeliefert und bezahlt Bestellung nicht anzeigen
427             if (parseInt(bestellung.ausgeliefert, 2) === Math.pow(2, kategorien.length) - 1
428                 && arrGrosserAls(bestellung.bezahlt, bestellung.bestellt)) {
429                 continue; //HIER ÜBERSPRINGEN
430             }
431
432             //Tabelle erstellen
433             var tabelle = $("<table id='b" + bestellung.id + "'></table>");
434
435             //caption
436             var caption = $("<caption class='bestellungCaption schrift'></caption>");
437             caption.attr("onclick", "verstecken($(this))");
438             $(tabelle).append(caption);
439
440             //Infos für in Caption
441             var anzProdukte = 0;
442             var preis = 0;
443             for (var i = 0; i < bestellung.bestellt.length; i++) {
444                 anzProdukte += bestellung.bestellt[i];
445                 preis += bestellung.bestellt[i] * produkte[i].preis;
446             }
447             var ausgeliefertStatusString; //verwendet ausgeliefert, nicht verarbeitet
448             if (parseInt(bestellung.ausgeliefert, 2) === Math.pow(2, kategorien.length)-1) {
449                 ausgeliefertStatusString = "ausgel";
450             } else {
451                 ausgeliefertStatusString = "";
452             }
453

```

```

454     //p-div für allen Anzeige-Text
455     var anzeigeDiv = $("<p class='bestellungInfo'></p>");
456     $(caption).append(anzeigeDiv);
457
458     //Texte für die Anzeige zusammenbasteln
459     var anzeigel = $("<span></span>").text("Id: " + bestellung.id +
460         ", Zeit: " + zeitAnzeige(bestellung.zeit) + ", ");
461     var anzeige2 = $("<span class='ausgeliefertStatusString'></span>").
462         text(ausgeliefertStatusString);
463     var anzeige3 = $("<span></span>").
464         text("Menge: " + anzProdukte + ", Preis: " + preisAnzeige(preis));
465     $(anzeigeDiv).append(anzeigel, anzeige2, $("<br>"), anzeige3);
466
467     //Bezahlen-Feld hinzufügen
468     //event.stopPropagation(); um ausfahren der Bestellung zu verhindern
469     var bezahlenFeld = $("<div class='bestellungAuswahlen' onclick='event.stopPropagation();'></div>");
470     $(caption).append(bezahlenFeld);
471     //entweder Häkchen oder Checkbox in bezahlenFeld
472     if (arrGrosserAls(bestellung.bezahlt, bestellung.bestellt)) {
473         var hackenFeld = $("<div class='hacken'></div>");
474         $(bezahlenFeld).append(hackenFeld);
475     } else {
476         var hackenFeld = $("<input type='checkbox' data-bestellung-id='" + bestellung.id +
477             "'><label></label>");
478         $(bezahlenFeld).append(hackenFeld);
479         hackenFelder.push(hackenFeld.first()); //nur Checkbox pushen, label nicht
480     }
481
482     //Zeilen erstellen für Produkte und Kategorien
483     for (var i = 0; i < kategorien.length; i++) {
484         //Zeile für Kategorie. In Menge-Feld steht ausgeliefert Status
485         //enthält trotzdem Menge-Feld, unter anderem wegen Layout)
486
487         //Info ob Kategorie ausgeliefert
488         var ausgeliefertStat = bestellung.ausgeliefert.charAt(
489             bestellung.ausgeliefert.length - 1 - i);
490         var ausgeliefertStr;
491         if (ausgeliefertStat === '1') {
492             ausgeliefertStr = "ausgel";
493         } else {
494             ausgeliefertStr = "";
495         }
496         //Zeile erstellen
497         var zeileKategorie = $("<tr class='kategorie'><td class='feldName'>" + kategorien[i].name +
498             "</td><td class='feldMenge ausgeliefertStatusString'>" + ausgeliefertStr + "</td></tr>");
499         $(tabelle).append(zeileKategorie); //Zeile an Tabelle pappen
500         zeileKategorie.hide(); //Zeile verstecken
501
502         //evt. muss Zeile für Kategorie wieder weggenommen werden (siehe unten)
503         var kategorieLeer = true;
504
505         for (var k = 0; k < kategorien[i].produkte.length; k++) {
506             var produkt = kategorien[i].produkte[k]; //id des Produkts
507
508             var menge = bestellung.bestellt[produkt];
509             var bezahlt = bestellung.bezahlt[produkt];
510             //Überspringen, falls kein solches Produkt bestellt ist und
511             //auch keines bezahlt
512             if (menge === 0 && bezahlt === 0) {
513                 continue;
514             }
515             //mindestens einmal bis hier gekommen: Kategorie nicht leer
516             kategorieLeer = false;
517
518             //Zeile für Produkt erstellen: Name, Menge bezahlt, Menge bestellt
519             var zeileProdukt = $("<tr class='produkt'><td class='feldName'>" + produkte[produkt].name +
520                 "</td><td id='b'" + bestellung.id + " p" + produkt + "' class='feldMenge'>" +
521                 bezahlt + " / " + menge + "</td></tr>");
522             $(tabelle).append(zeileProdukt); //Zeile an Tabelle pappen
523             zeileProdukt.hide(); //Zeile verstecken
524         }
525
526         //Kategorie-Zeile wieder wegnehmen, falls kein Produkt dieser Kategorie
527         if (kategorieLeer) {
528             zeileKategorie.remove();
529         }
530     }
531
532     //Tabelle an #neueBestellungen pappen
533     $("#neueBestellungen").append(tabelle);
534
535     //Ansicht wechseln
536     ansicht("tisch");
537     laden(false);
538 }); //Funktion für nach dem Auslesen fertig
539
540 //FEHLERMELDUNG falls Auslesen fehlschlägt
541 verzogerung.fail(function(fehler) {
542     ansicht("fehler");
543
544     var fehlerAnzeige = $("<p></p><br>").text("Laden der Bestellungen fehlgeschlagen: " + fehler);
545     $("#fehler").append(fehlerAnzeige);
546
547     console.log("KRITISCHER FEHLER: Laden der Bestellungen fehlgeschlagen: " + fehler);
548 });
549
550 function verstecken(caption) {
551     var onclick = caption.attr("onclick"); //onclick-Funktion speichern
552     caption.attr("onclick", ""); //onclick-Funktion entfernen
553     slideToggle(caption.siblings(), 200, function() {
554         //nach Animation: onclick-Funktion wieder hinzufügen
555         caption.attr("onclick", onclick);
556     });
557 }

```

```

554     });
555 }
556
557 //beginnt neue Bestellung
558 function neueBestellung() {
559     bestellt.fill(0);
560     totalPreis = 0;
561
562     ansicht("kacheln");
563 }
564
565 //erstellt neues Bestellung-Objekt
566 function bestellungBestatigen(tisch) {
567     //prüfen, ob bestellt-Array leer: abbrechen
568     var istLeer = false;
569     istLeer = bestellt.every(function(eintrag, index, array) {
570         return eintrag == 0;
571     });
572     if (istLeer) {
573         ansicht("tischeÜbersicht");
574         return;
575     }
576
577     //neues Bestellung-Objekt für diesen Tisch erstellen
578     var bestellung = new ErstelleBestellung(tisch, bestellt.slice(), kellnerId);
579
580     //Bestellung in Datenbank eintragen
581     var verzogerung = insertBestellung(bestellung);
582     verzogerung.fail(function(fehler) {
583         ansicht("fehler");
584
585         var fehlerAnzeige = $("

</p><br>").text("KRITISCHER FEHLER: Schreiben der Bestellung (" +
586             JSON.stringify(bestellung) + ") fehlgeschlagen: " + fehler);
587         $("#fehler").append(fehlerAnzeige);
588
589         console.log("KRITISCHER FEHLER: Schreiben der Bestellung (" + bestellung + ") fehlgeschlagen: " + fehler);
590     });
591
592     //zur neueBestellung Ansicht zurückkehren
593     ansicht("tischeÜbersicht");
594 }
595
596 //Bestellung abbrechen
597 function bestellungAbbrechen() {
598     ansicht("tischeÜbersicht");
599 }
600
601 //bezahlen-Knopf in Tisch-Ansicht
602 function zuBezahlen() {
603     var zuBezahlen = []; //IDs von Bestellungen (nicht zwingend sortiert)
604
605     //für jede Box: überprüfen ob ausgewählt und
606     //falls ja zugehörige Bestellung-ID zum zuBezahlen-Array hinzufügen
607     for (var i = 0; i < hackenFelder.length; i++) {
608         var box = hackenFelder[i]; //aktuelle Box
609         if (box.prop("checked")) { //falls sie angewählt ist
610             zuBezahlen.push(parseInt(box.data("bestellung-id")));
611         }
612     }
613
614     //abbrechen, falls gar nichts angewählt (Ansicht nicht wechseln)
615     if (zuBezahlen.length == 0) {
616         return;
617     }
618
619     console.log("zuBezahlen(): zuBezahlen-Array (IDs):", zuBezahlen);
620
621     //Bestellungs-Objekte in Bestellungen-Array suchen aufgrund von zuBezahlen
622     //bestellt- und bezahlt-Arrays der Bestellungen zusammenzählen
623     bestellungenBezahlen = [];
624     var mengen = new Array(produkte.length);
625     mengen.fill(0);
626     for (var i=0; i < bestellungen.length; i++) {
627         for (var k=0; k < zuBezahlen.length; k++) {
628             if (bestellungen[i].id == zuBezahlen[k]) {
629                 bestellungenBezahlen.push(bestellungen[i]); //nur Referenz
630
631                 //bestellt-Arrays zusammenzählen
632                 var bestellungBestellt = bestellungen[i].bestellt;
633                 var bestellungBezahlt = bestellungen[i].bezahlt;
634
635                 for (var m=0; m < mengen.length; m++) {
636                     mengen[m] += bestellungBestellt[m];
637                     mengen[m] -= bestellungBezahlt[m];
638
639                     //würde Negative anzeigen in bezahlen-Ansicht
640                     if (mengen[m] <= 0) {
641                         mengen[m] = 0;
642                     }
643                 }
644                 break; //nächstes Objekt prüfen
645             }
646         }
647     }
648
649     console.log("zu bezahlende Produkte (bestellt-Array):", mengen);
650
651     //Bezahlen-Ansicht vorbereiten und anzeigen
652     bezahlenInitialisieren(mengen);
653     ansicht("bezahlen");
654 }
655
656
657 // .....


```

```

658 // ..... BEZAHLEN .....
659 // .....
660
661 //Initialisierung der bezahlen-Ansicht
662 //bezahlen-Arrays initialisieren
663 //nein-Seite Produkte & Kategorien anzeigen, Mengen einfüllen (auch 0)
664 //ja-Seite alle Produkte verstecken (aber nicht 0 setzen),
665 // aber Kategorien der nein-Seite anzeigen
666 //arrNein, arrJa: Format von bestellt-Array, ACHTUNG: nur als Referenz
667 //arrNein: bezahlenNein-Array, alle Produkte, die bezahlt werden könnten
668 //arrJa: bezahlenJa-Array. Meistens (immer?) weggelassen (dann einfach alles 0)
669 function bezahlenInitialisieren(arrNein, arrJa/* = false*/) {
670     if (arrJa === undefined) {
671         arrJa = false;
672     }
673
674     //bezahlen-Nein/-Ja-Arrays aufsetzen
675     bezahlenNein = arrNein.slice();
676
677     if (arrJa === false) { //falls nicht gesetzt
678         bezahlenJa.fill(0);
679     } else {
680         bezahlenJa = arrJa.slice();
681     }
682
683     //Kategorien, welche in nein-Seite angezeigt werden,
684     //diese werden dann auch auf ja-Seite angezeigt
685     var kategorienAnzeigen = []
686
687     //NEIN-SEITE
688     //Alle Kategorie- und Produkt-Zeilen verstecken (aber nicht caption)
689     $("#bezahlenNein").children().not(".pseudoVersteckt").hide();
690
691     //für jede Kategorie
692     for (var i=0; i < kategorien.length; i++) {
693         var kategorie = kategorien[i];
694
695         //ob Kategorie-Zeile angezeigt werden soll
696         var kategorieLeer = true;
697
698         //für jedes Produkt der Kategorie
699         for (var k=0; k < kategorie.produkte.length; k++) {
700             var id = kategorie.produkte[k];
701
702             //Mengenanzeige aktualisieren (auch wenn 0)
703             $("#bezahlenMengeNein" + id).text(bezahlenNein[id]);
704
705             //anzeigen, falls Menge nicht 0
706             if (bezahlenNein[id] !== 0) {
707                 $("#bezahlenProduktNein" + id).show();
708
709                 //min ein Mal bis hier gekommen: Kategorie nicht leer
710                 kategorieLeer = false;
711             }
712         }
713
714         //Kategorie-Zeile anzeigen/Verstecken
715         if (!kategorieLeer) {
716             $("#bezahlenKategorieNein" + kategorie.id).show();
717             kategorienAnzeigen.push(i);
718         }
719     }
720
721     //JA-SEITE
722     //Alle Kategorie- und Produkt-Zeilen verstecken (aber nicht caption)
723     $("#bezahlenJa").children().not(".pseudoVersteckt").hide();
724     //$("#bezahlenJa").children().not("caption").hide();
725     for (var i=0; i < kategorienAnzeigen.length; i++) {
726         $("#bezahlenKategorieJa" + kategorienAnzeigen[i]).show();
727     }
728 }
729
730 //ein Produkt von der bezahlenja zu der bezahlenNein Seite schieben oder umgekehrt
731 //bezahlen-Nein/Ja-Arrays updaten, Menge Anzeige beider Seiten des Produkts updaten
732 //Produkt-Zeile verstecken, falls 0
733 //id: produkt.id des Produkts
734 //richtung: true: von Nein nach Ja, false: von ja nach nein
735 function bezahlenSchieben(id, richtung) {
736     //bezahlen-Arrays updaten
737     if (richtung) { //nein -> ja
738         if (bezahlenNein[id] === 0) { //nicht ausführen, falls schon 0
739             return;
740         }
741         bezahlenNein[id]--;
742         bezahlenJa[id]++;
743     } else { //ja -> nein
744         if (bezahlenJa[id] === 0) { //nicht ausführen, falls schon 0
745             return;
746         }
747         bezahlenNein[id]++;
748         bezahlenJa[id]--;
749     }
750
751     //Anzeige updaten
752     $("#bezahlenMengeNein" + id).text(bezahlenNein[id]);
753     if (bezahlenNein[id] === 0) {
754         $("#bezahlenProduktNein" + id).hide();
755     } else {
756         $("#bezahlenProduktNein" + id).show();
757     }
758     $("#bezahlenMengeJa" + id).text(bezahlenJa[id]);
759     if (bezahlenJa[id] === 0) {
760         $("#bezahlenProduktJa" + id).hide();
761     } else {
762         $("#bezahlenProduktJa" + id).show();

```

```

763     }
764 }
765
766 //alle Produkte einer Kategorie, die noch auf der betreffenden Seite sind,
767 // von der bezahlenJa zu der bezahlenNein Seite schieben oder umgekehrt
768 //bezahlen-Nein/Ja-Arrays updaten, Menge Anzeige beider Seiten des Produkts updaten
769 //Produkt-Zeilen verstecken / anzeigen
770 //id: kategorie.id der Kategorie
771 //richtung: true: von Nein nach Ja, false: von ja nach nein
772 function bezahlenSchiebenKat(kat, richtung) {
773     var betreffendeProdukte = kategorien[kat].produkte;
774
775     //für jedes Produkt der Kategorie
776     for (var i=0; i < betreffendeProdukte.length; i++) {
777         var id = betreffendeProdukte[i]; //id des Produkts
778
779         if (richtung) { //nein -> ja
780             bezahlenJa[id] += bezahlenNein[id];
781             bezahlenNein[id] = 0;
782
783             //Anzeigen updaten
784             $("#bezahlenMengeNein" + id).text(0);
785             $("#bezahlenProduktNein" + id).hide();
786
787             $("#bezahlenMengeJa" + id).text(bezahlenJa[id]);
788             if (bezahlenJa[id] === 0) {
789                 $("#bezahlenProduktJa" + id).hide();
790             } else {
791                 $("#bezahlenProduktJa" + id).show();
792             }
793         } else { //ja -> nein
794             bezahlenNein[id] += bezahlenJa[id];
795             bezahlenJa[id] = 0;
796
797             //Anzeigen updaten
798             $("#bezahlenMengeJa" + id).text(0);
799             $("#bezahlenProduktJa" + id).hide();
800
801             $("#bezahlenMengeNein" + id).text(bezahlenNein[id]);
802             if (bezahlenNein[id] === 0) {
803                 $("#bezahlenProduktNein" + id).hide();
804             } else {
805                 $("#bezahlenProduktNein" + id).show();
806             }
807         }
808     }
809 }
810
811 //alle Produkte die noch auf der betreffenden Seite sind,
812 // von der bezahlenJa zu der bezahlenNein Seite schieben oder umgekehrt
813 //bezahlen-Nein/Ja-Arrays updaten, Menge Anzeige beider Seiten des Produkts updaten
814 //Produkt-Zeilen verstecken / anzeigen
815 //richtung: true: von Nein nach Ja, false: von ja nach nein
816 function bezahlenSchiebenAlle(richtung) {
817     //für alle Produkte bzw. alle Stellen in bezahlen-Arrays
818     for (var i=0; i < bezahlenJa.length; i++) {
819         var id = i; //id des Produkts bzw. Index in bezahlen-Arrays
820
821         if (richtung) { //nein -> ja
822             bezahlenJa[id] += bezahlenNein[id];
823             bezahlenNein[id] = 0;
824
825             //Anzeigen updaten
826             $("#bezahlenMengeNein" + id).text(0);
827             $("#bezahlenProduktNein" + id).hide();
828
829             $("#bezahlenMengeJa" + id).text(bezahlenJa[id]);
830             if (bezahlenJa[id] === 0) {
831                 $("#bezahlenProduktJa" + id).hide();
832             } else {
833                 $("#bezahlenProduktJa" + id).show();
834             }
835         } else { //ja -> nein
836             bezahlenNein[id] += bezahlenJa[id];
837             bezahlenJa[id] = 0;
838
839             //Anzeigen updaten
840             $("#bezahlenMengeJa" + id).text(0);
841             $("#bezahlenProduktJa" + id).hide();
842
843             $("#bezahlenMengeNein" + id).text(bezahlenNein[id]);
844             if (bezahlenNein[id] === 0) {
845                 $("#bezahlenProduktNein" + id).hide();
846             } else {
847                 $("#bezahlenProduktNein" + id).show();
848             }
849         }
850     }
851 }
852
853 // .....
854 // ..... QUITTUNG .....
855 // .....
856
857 //Quittungsansicht vorbereiten und anzeigen
858 //mengen: Format von bestellt-Array: was anzuzeigen ist
859 function quittungAnzeigen(mengen) {
860     //nur anzeigen, falls überhaupt min 1 Produkt ausgewählt
861     var anzProdukte = mengen.reduce(function(acc, e) { return acc + e; }, 0);
862     if (anzProdukte < 1) {
863         return; //ABBRECHEN, falls keine Produkte
864     }
865
866     ansicht("quittung");
867

```

```

868
869     var totalPreis = 0;
870
871     for (var id=0; id < produkte.length; id++) { //id als Counter
872         var menge = mengen[id];
873
874         //Preis für die bestimmte Menge des Produkts
875         var produktPreis = produkte[id].preis * menge;
876
877         totalPreis += produktPreis;
878
879         //Menge Feld aktualisieren
880         $("#quittungMenge" + id).text(mengeAnzeige(menge));
881         //ProdukteFeld aktualisieren
882         $("#quittungPreis" + id).text(preisAnzeige(produktPreis, false));
883
884         //Zeile anzeigen/verstecken
885         if (menge > 0) {
886             $("#quittungMenge" + id).parent().show();
887         } else {
888             $("#quittungMenge" + id).parent().hide();
889         }
890     }
891
892     //Total Preis anzeigen
893     $("#quittungTotalPreis").text(preisAnzeige(totalPreis, true));
894 }
895
896
897
898 // ACHTUNG = ACHTUNG = ACHTUNG = ACHTUNG = ACHTUNG = ACHTUNG = ACHTUNG = ACHTUNG
899 // falls zwei Kellner an gleicher Bestellung bezahlen/stornieren,
900 // überschneiden sich bezahlen- und storno-Arrays.
901 // Allenfalls umgehen mit 1-Bit Spalte für "wird gerade bearbeitet",
902 // nicht zugreifbar von anderen Satelliten (kein Problem für Basen/Sprinter)
903 // ACHTUNG = ACHTUNG = ACHTUNG = ACHTUNG = ACHTUNG = ACHTUNG = ACHTUNG = ACHTUNG
904
905
906
907 //markiert Produkte von Bestellungen als bezahlt
908 //falls von mehreren Bestellungen zusammen bezahlt wurde:
909 // bezahlt-Array der chronologisch ersten verwendet, falls dort
910 // genügend Produkte bestellt, sonst chronologisch zweite etc.
911 // Allfällige bezahl-Überschüsse (zu viel bezahlt), gehen an die chronologisch letzte
912 // Aber alles kein problem, falls nur von einer Bestellung bezahlt
913 //verwendet und updated bezahlenNein & bezahlenJa Arrays und
914 // von bestellungenBezahlen jeweils .bezahlt
915 //storno: true: Produkte zusätzlich dem storno-Array hinzufügen, false: normal bezahlen
916 function bezahlen(storno/* = false */) {
917     if (storno === undefined) {
918         storno = false;
919     }
920
921     //nur machen, falls überhaupt min 1 Produkt ausgewählt
922     var anzProdukte = bezahlenJa.reduce(function(acc, e) { return acc + e; }, 0);
923     if (anzProdukte < 1) {
924         return false; //ABBRECHEN, falls keine Produkte
925     }
926
927     var geandert = []; //alle geänderten Bestellungen
928
929     //bezahlte Produkte (bezahlenJa-Array) auf .bezahlt
930     // der bestellungenBezahlen verteilen
931     for (var i = 0; i < bezahlenJa.length; i++) {
932         var mengeZuBezahlen = bezahlenJa[i]; //wie viel vom Produkt bezahlt werden soll
933         if (mengeZuBezahlen < 1) { //also falls 0
934             continue;
935         }
936
937         for (var k = 0; k < bestellungenBezahlen.length; k++) {
938             if (storno) {
939                 if (bestellungenBezahlen[k].storno === undefined) {
940                     bestellungenBezahlen[k].storno = new Array(produkte.length);
941                     bestellungenBezahlen[k].storno.fill(0);
942                 }
943             }
944
945             //wie viel vom Produkt bestellt wurde in dieser Bestellung
946             var menge = bestellungenBezahlen[k].bestellt[i];
947             //wie viel vom Produkt bereits bezahlt wurde
948             var mengeBezahlt = bestellungenBezahlen[k].bezahlt[i];
949
950             if (mengeBezahlt >= menge) { //schon alles/zu viel bezahlt
951                 continue;
952             }
953
954             //Bestellung.bezahlt wird geändert
955             //schon hier oben pushen, damit vor break unten
956             //aber nur anfügen, falls nicht schon bei letztem Durchgang
957             if (geandert.length > 0) {
958                 if (geandert[geandert.length - 1].id !== bestellungenBezahlen[k].id) {
959                     geandert.push(bestellungenBezahlen[k]);
960                 }
961             } else {
962                 geandert.push(bestellungenBezahlen[k]);
963             }
964
965             var kapazitat = menge - mengeBezahlt; //wie viele unbezahlte es noch gibt
966             // // bzw. wie viel Platz noch frei ist
967
968             //kann alles auf diese Bestellung getan werden?
969             if (mengeZuBezahlen <= kapazitat) {
970                 bestellungenBezahlen[k].bezahlt[i] += mengeZuBezahlen;
971                 if (storno) {
972                     bestellungenBezahlen[k].storno[i] += mengeZuBezahlen;

```

```

973     }
974     mengeZuBezahlen = 0;
975     break; //falls komplette Anz zugeordnet: abbrechen
976 } else { //also mehr bezahlt werden soll, als noch Platz in dieser Bestellung
977     bestellungenBezahlen[k].bezahlt[i] += kapazitat;
978     if (storno) {
979         bestellungenBezahlen[k].storno[i] += kapazitat;
980     }
981 }
982     mengeZuBezahlen -= kapazitat;
983 }
984 }
985
986 //falls mehr bezahlt werden soll, als eigentlich bestellt wurde
987 if (mengeZuBezahlen > 0) {
988     //Überschuss in mengeZuBezahlen einfach der letzten Bestellung dazuzählen
989     var index = bestellungenBezahlen.length - 1;
990     bestellungenBezahlen[index].bezahlt[i] += mengeZuBezahlen;
991     if (storno) {
992         bestellungenBezahlen[index].storno[i] += mengeZuBezahlen;
993     }
994     mengeZuBezahlen = 0;
995
996     //nur anfügen, falls nicht schon bei letztem Durchgang
997     if (geandert.length > 0) {
998         if (geandert[geandert.length - 1].id !== bestellungenBezahlen[index].id) {
999             geandert.push(bestellungenBezahlen[index]);
1000         }
1001     } else {
1002         geandert.push(bestellungenBezahlen[index]);
1003     }
1004 }
1005 }
1006
1007 //DB: alle geänderten Bestellungen updaten in DB
1008 var alleVerzogerungen = [];
1009 for (var i=0; i < geandert.length; i++) {
1010     var updateDaten = {
1011         id: geandert[i].id,
1012         bezahlt: geandert[i].bezahlt,
1013         bezahltVon: kellnerId
1014     }
1015     //allenfalls auch updateDaten.storno setzen
1016     if (storno) { //storno Array existiert sicher
1017         //true: alles 0, false: nicht alles 0
1018         var stornoArrLeer = true;
1019         for (var k = 0; k < geandert[i].storno.length; k++) {
1020             if (geandert[i].storno[k] !== 0) {
1021                 stornoArrLeer = false;
1022                 break;
1023             }
1024         }
1025         if (!stornoArrLeer) {
1026             updateDaten.storno = geandert[i].storno;
1027         }
1028     }
1029
1030     var verzogerung = bestellungenUpdaten(updateDaten);
1031     alleVerzogerungen.push(verzogerung);
1032 }
1033 //Verzögerungen zusammenfassen
1034 var masterVerzogerung = $.when.apply($, alleVerzogerungen);
1035 //Fehlerhandling
1036 masterVerzogerung.fail(function(fehler) {
1037     console.log("KRITISCHER FEHLER: in bezahlen():
1038         bezahlt-Arrays, bezahltVon-Felder in DB nicht oder nicht alle geupdatet:", fehler);
1039
1040     var fehlerAnzeige = $("

<p><br>").text("KRITISCHER FEHLER:
1041         in bezahlen(): bezahlt-Arrays, bezahltVon-Felder in DB nicht oder nicht alle geupdatet: " + fehler);
1042     $("#fehler").append(fehlerAnzeige);
1043 });
1044 masterVerzogerung.done(function(nachricht) {
1045     console.log("bezahlen():
1046         OK (alle bezahltVon-Felder und bezahlen-Arrays (und allenfalls storno-Arrays) geupdatet)");
1047 });
1048 //bezahlt-Mengen in einzelner-Tisch-Ansicht der geänderten Bestellungen updaten
1049 //ID der ganzen Zeile: id='b' + bestellung.id + "_p" + produkt + ""
1050 for (var i=0; i < geandert.length; i++) {
1051     var bestellung = geandert[i];
1052     var id = bestellung.id;
1053
1054     for (var k=0; k < bestellung.bezahlt.length; k++) {
1055         $("#b" + id + "_p" + k).text(bestellung.bezahlt[k] + " / " + bestellung.bestellt[k]);
1056     }
1057 }
1058
1059 //Ansicht zurück zu bezahlen
1060 //bezahlenJa Arrays wieder mit 0 füllen (bezahlenNein-Array hat noch Unbezahltes)
1061 bezahlenJa.fill(0);
1062
1063 bezahlenInitialisieren(bezahlenNein);
1064 ansicht("bezahlen");
1065 }
1066
1067 // .....
1068 // ..... BESTELLEN .....
1069 // .....
1070
1071 // ..... ALLGEMEIN .....
1072 //Produkt (ab)bestellen wenn auf Kachel geklickt
1073 //Position id des bestellt-Arrays plus/minus 1
1074 //id: ID des zu bestellenden Produkts


```

```

1075 function bestellen(id) {
1076     if (abbestellen) {
1077         //eins abzählen
1078         if (bestellt[id] > 0) { //keines abzählen, falls bereits 0
1079             bestellt[id]--;
1080             totalPreis -= produkte[id].preis;
1081         }
1082     } else {
1083         //eins hinzufügen
1084         bestellt[id]++;
1085         totalPreis += produkte[id].preis;
1086     }
1087
1088     //Menge in Produkte Kachel aktualisieren
1089     mengeDiv[id].text(mengeAnzeige(bestellt[id], true));
1090     //Angaben unten aktualisieren
1091     $("#subTotal").text(preisAnzeige(totalPreis, true));
1092 }
1093
1094 //Modus auf abbestellen/normal wechseln
1095 //abbestellenNeu: zu setzender Modus, falls nicht angegeben: umschalten
1096 function modusAbbestellen(abbestellenNeu/* = ! abbestellen */) {
1097     if (abbestellenNeu === undefined) {
1098         abbestellenNeu = !abbestellen;
1099     }
1100
1101     abbestellen = abbestellenNeu;
1102
1103     if (abbestellen) { //umdesignen auf abbestellen
1104         $("#abbestellen").removeClass("modusBestellen").addClass("modusAbbestellen");
1105     } else { //umdesignen auf bestellen
1106         $("#abbestellen").removeClass("modusAbbestellen").addClass("modusBestellen");
1107     }
1108 }
1109
1110 // ..... KACHELN .....
1111 //Kategorie öffnen
1112 //x: kategorie.id
1113 function kategorieOffnen(x) {
1114     //überprüfen, ob x eine existierende Kategorie ist
1115     if (x >= kategorien.length || x < -1) {
1116         console.log("Fehler in kategorieOffnen(x). x ist " + x +
1117             ". Es gibt nur " + (kategorien.length - 1) +
1118             " Kategorien und -1 für alle Kategorien (ober Kategorie).");
1119         return;
1120     }
1121 }
1122
1123 modusAbbestellen(false); //abbestellen ausschalten
1124
1125 if (x == -1) { //alle Kategorien (ober Kategorie) öffnen
1126     //Kategorie Knopf austauschen
1127     $("#navigationAbbrechen").show();
1128     $("#navigationKategorie").hide();
1129
1130     jetzigeKategorie = -1;
1131     $("#jetzigeKategorie").text("Kategorien");
1132     $("#allekacheln").children().hide(); //alles alte verstecken
1133     for (var i=0; i < kategorienDiv.length; i++) {
1134         kategorienDiv[i].show();
1135     }
1136 } else { //eine Kategorie öffnen
1137     //Kategorie Knopf austauschen
1138     $("#navigationKategorie").show();
1139     $("#navigationAbbrechen").hide();
1140
1141     $("#allekacheln").children().hide(); //alles alte verstecken
1142     jetzigeKategorie = x;
1143     $("#jetzigeKategorie").text(kategorien[x].name); //Kategorie Text aktualisieren
1144     var anzuzeigen = kategorien[x].produkte;
1145     for (var i=0; i < anzuzeigen.length; i++) {
1146         var produktId = anzuzeigen[i];
1147         //Menge des Produktes aktualisieren (wird auch in bestellen(id) gemacht)
1148         mengeDiv[produktId].text(mengeAnzeige(bestellt[produktId], true));
1149         produkteDiv[produktId].show();
1150     }
1151 }
1152 }
1153
1154 // ..... LISTE .....
1155 //Liste aktualisieren. Mengen aktualisieren
1156 //Produkte mit Menge null ausblenden und Kategorien ohne anzuzeigende Produkte ausblenden
1157 function listeAktualisieren() {
1158     for (var i=0; i < kategorien.length; i++) {
1159         //ob Kategorie überhaupt angezeigt werden soll
1160         var kategorieLeer = true;
1161
1162         for (var k=0; k < kategorien[i].produkte.length; k++) {
1163             var id = kategorien[i].produkte[k];
1164             var menge = bestellt[id]; //Menge auslesen
1165             //Menge in entsprechendes Feld (td) schreiben
1166             $("#listeMenge" + id).text(mengeAnzeige(menge));
1167
1168             //ganze Zeile (--> .parent() ) anzeigen oder verbergen
1169             if (menge > 0) {
1170                 $("#listeMenge" + id).parent().show();
1171                 //min 1 Mal bis hier gekommen: Kategorie nicht leer
1172                 kategorieLeer = false;
1173             } else {
1174                 $("#listeMenge" + id).parent().hide();
1175             }
1176         }
1177     }
1178
1179     //Kategorie anzeigen/verstecken (versteckt ganzes Tabelle umschliessendes Div)

```

```

1180         if (kategorieLeer) {
1181             $("#listeKategorie" + kategorien[i].id).hide();
1182         } else {
1183             $("#listeKategorie" + kategorien[i].id).show();
1184         }
1185     }
1186
1187     //Preisanzeige unten aktualisieren
1188     $("#listeTotalPreis").text("Total: " + preisAnzeige(totalPreis, true));
1189 }
    
```

## 12. sprinter.css

```

1 /*
2  * Datei: sprinter.css
3  * Projekt: Kellner App
4  * Autor: Michael Heider
5  * Datum: HS 2017
6  *
7  * Was:
8  * CSS zu Sprinter
9  */
10
11
12 /* =====
13 / ===== GENERELLES =====
14 / =====*/
15 /*allgemein grössere Schriftgrösse, damit von Weitem und im vorbeigehen besser leserlich*/
16 * {
17     font-size: 22px;
18 }
19
20
21 /* =====
22 / ===== ALLGEMEINE KLASSEN =====
23 / =====*/
24
25 /* ALLGEMEINES */
26 /*siehe auch in gemeinsam.css*/
27
28 /* SEITENSTRUKTUR */
29 .header {
30     position: fixed;
31     width: 100%;
32     height: 55px;
33     top: 0%;
34 }
35
36 .footer {
37     position: fixed;
38     width: 100%;
39     height: 55px;
40     bottom: 0%;
41 }
42
43 #hauptbereich {
44     position: absolute;
45     width: 100%;
46     left: 0px;
47     top: 55px;
48     bottom: 0px;
49
50     overflow: auto;
51 }
52
53 /*ELEMENTE IN STRUKTUR*/
54 #headerMitte {
55     background-color: #FCF67D;
56
57     width: 100%;
58     height: 100%;
59
60     float: left;
61     text-align: center;
62 }
63
64
65 /* =====
66 / ===== ANZEIGE VON BESTELLUNGEN =====
67 / =====*/
68
69 .bestellungCaption {
70     padding-top: 16px;
71     padding-bottom: 16px;
72 }
73
74 .bestellungInfo {
75     float: left;
76
77     line-height: 200%;
78 }
79
80 .bestellungAuswählen {
81     float: right;
82
83     width: 40px; /*APASSEN: Breite*/
84     height: 40px; /*APASSEN: Höhe*/
85
86     /*margin-top: 0px; /*pseudo vertikales Zentrieren*/
87
    
```

```

88     cursor: pointer;
89     border-radius: 3px;
90     background-color: #aaa;
91 }
92
93 label:before {
94     bottom: 0px; /*vertikaler Offset anpassen*/
95 }
96
97 .feldName {
98 }
99
100 .feldMenge {
101     width: 35px;
102
103     text-align: right;
104 }
105
106 tr:nth-child(2) { /* oberer Rahmen (zweites Kind, nach caption (beginnt mit 1)) */
107     border-top: 1px solid black;
108 }

```

### 13. sprinter.html

```

1  <!--
2  < Datei: sprinter.html
3  < Projekt: Kellner App
4  < Autor: Michael Heider
5  < Datum: HS 2017
6  <
7  < Was:
8  < HTML zu Sprinter.
9  < Enthält Grund-HTML für Layout.
10 < Alles, was EventDaten (zum Beispiel Produktnamen) verwendet,
11 < wird dynamisch erzeugt von sprinter.js.
12 -->
13
14
15 <!DOCTYPE html>
16 <html>
17 <head>
18 <meta charset="utf-8"/>
19
20 <title>Kellner App Sprinter</title>
21
22
23 <!--=====
24 < ===== HEAD BEREICH =====
25 < =====>
26
27 <!-- Korrekte Darstellung/Einstellung auf Smartphones -->
28 <meta name="viewport" content="width=device-width, initial-scale=1.0, user-scalable=no">
29
30
31 <!--.....
32 < ..... ZEUG FÜR WEB-APP .....
33 < .....>
34
35 <!--..... ANDROID .....>
36 <!-- (ACHTUNG: MUSS OBEN AN APPLE TAGS SEIN!!! (wieso??))-->
37 <link rel="manifest" href="webapp/manifestSprinter.json">
38
39 <!--..... IOS .....>
40 <!-- ist eine Web App -->
41 <meta name="apple-mobile-web-app-capable" content="yes"/>
42 <!-- Statusbar Farbe und Stil -->
43 <meta name="apple-mobile-web-app-status-bar-style" content="black"/>
44 <!-- Icon auf Homescreen -->
45 <link rel="apple-touch-icon" sizes="180x180" href="webapp/icon180x180.png">
46 <!-- Icon während Ladevorgang (ca. 1900x1000 Pixel) -->
47 <link rel="apple-touch-startup-image" href="webapp/startup1900x1000.png">
48 <!-- Text auf Homescreen (anstatt Text aus title Tag) -->
49 <meta name="apple-mobile-web-app-title" content="KellnerApp Sprinter"/>
50
51 <!--..... SERVICE-WORKER .....>
52 <!-- ServiceWorker (braucht es einfach für Web Apps) -->
53 <!-- Code leicht bearbeitet von URL: (11.8.2017)
54 < https://developers.google.com/web/fundamentals/getting-started/primers/service-workers
55 -->
56 <script type="text/javascript">
57     if ("serviceWorker" in navigator) {
58         window.addEventListener("load", function() {
59             navigator.serviceWorker.register("sw.js").then(function(registration) {
60                 //Registrierung war erfolgreich
61                 console.log("ServiceWorker-Registrierung erfolgreich. Scope: ",
62                     registration.scope);
63             }, function(fehler) {
64                 //Registrierung fehlgeschlagen
65                 console.log("ServiceWorker-Registrierung fehlgeschlagen: ", fehler);
66             });
67         });
68     }
69 </script>
70
71
72 <!--.....
73 < ..... CSS .....
74 < .....>
75 <!-- Stylesheets einbinden -->

```

```

76 <link rel="stylesheet" type="text/css" href="stylesheets/reset.css">
77 <link rel="stylesheet" type="text/css" href="stylesheets/gemeinsam.css">
78 <link rel="stylesheet" type="text/css" href="sprinter.css">
79
80
81 <!--.....
82 < ..... JAVASCRIPT .....
83 < .....
84 <!-- Scripts einbinden -->
85 <script type="text/javascript" src="scripts/polyfill.js"></script>
86 <script type="text/javascript" src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
87 <script type="text/javascript" src="scripts/allgemeineFunktionen.js"></script>
88 <!-- Hauptscript (auszuführendes Script) -->
89 <script type="text/javascript" src="sprinter.js"></script>
90
91
92 </head>
93
94
95 <!--=====
96 < ===== BODY BEREICH =====
97 < =====
98 <body>
99
100 <!--..... LADEN .....
101 <div id="laden">
102   <span>LADEN</span>
103 </div>
104
105 <!--..... FEHLER .....
106 <div id="fehler" style="display:none;">
107   <p><b>KRITISCHE FEHLER:</b></p><br>
108 </div>
109
110 <!--..... ANZEIGE .....
111 <div id="inhalt" style="display:none;">
112   <!-- HEADER -->
113   <section class="header rahmen">
114     <div id="headerMitte" class="jetzigeKategorie rahmen grosseschrift">Bestellungen</div>
115   </section>
116
117   <!-- HAUPTBEREICH -->
118   <section id="hauptbereich">
119     <div id="alleBestellungen"></div>
120   </section>
121 </div>
122
123 </body>
124 </html>

```

## 14. sprinter.js

```

1  /*
2  * Datei: sprinter.js
3  * Projekt: Kellner App
4  * Autor: Michael Heider
5  * Datum: HS 2017
6  *
7  * Was:
8  * Javascript-Logik Sprinter
9  */
10
11 "use strict"; //strict mode aktivieren
12
13
14 //Warnung einblenden bevor schliessen oder versehentlichem Neuladen
15 //   (speziell auf Handy durch runterziehen)
16 //auf einigen Browser muss zuerst mit der Webseite interagiert worden sein,
17 //   bevor die Meldung auftritt
18 window.addEventListener("beforeunload", function(e) {
19   //Nachricht wird nicht auf allen Browsern angezeigt, sondern irgendein Default
20   var nachricht = "Wenn Sie die App neu laden, müssen Sie sich erneut einloggen.";
21   e.returnValue = nachricht; //zwei return Zeilen für umfassenderen Browser-Support
22   return nachricht;
23 });
24
25
26 // =====
27 // ===== VARIABLEN, OBJEKTE, OBJECT CONSTRUCTORS, ETC. =====
28 // =====
29 // ..... IDENTIFIKATION .....
30 var sprinterId = "Sprinter_1";
31 var eventName; //wird ausgelesen in auslesen()
32 var zeigeKategorien = [0,1,2,3];
33 //Produkte aus welchen Kategorien (Ids) angezeigt werden sollen (einstellbar).
34 //Array von Kategorie-IDs (muss nicht der Reihe nach sein).
35 //Sicherstellen, dass es KEINE NICHT EXISTIERENDEN kategorien gibt., keine führenden 0,
36 //   können mehrmals vorkommen
37 //muss übereinstimmen mit Array von zugehöriger Basis
38
39 // ..... EVENT DATEN .....
40 var produkte = []; //alle Produkte. Position in Array stimmt mit produkt.id überein
41 //siehe auch ErstelleProdukt-Constructor (allgemeineFunktionen.js)
42 var kategorien = []; //alle Kategorien. Position in Array stimmt mit kategorie.id überein
43 //siehe auch ErstelleKategorie-Constructor (allgemeineFunktionen.js)
44 var tische = [];
45 //siehe auch ErstelleTisch-Constructor (allgemeineFunktionen.js)
46
47 // ..... ANSICHT INHALT .....
48 //var hackenFelder = []; //ale Hackenfelder als jQuery

```

```

49 var angezeigteBestellungen = []; //Array von angezeigten Bestellungen-IDs
50 var bestellungenJQ; //alle angezeigten Bestellungen-Tabellen als EIN jQuery
51 // (hinzufügen mit .add()). Wird am Anfang initialisiert.
52
53 // ..... AUSLESEN .....
54 var auslesenInterval; //Interval zum neueAuslesen()
55 var auslesenErfolg = 5; //Erfolgszähler, stellt sicher, dass neueAuslesen()
56 // nicht zu viele Male nacheinander fehlschlägt. Siehe auslesenFehler()
57
58
59 // =====
60 // ===== DOCUMENT READY FUNCTION: INITIALISIERUNG =====
61 // =====
62 $(document).ready(function() {
63 // .....
64 // ..... EVENT-DATEN AUSLESEN UND VORBEREITEN .....
65 // .....
66
67 //Event-Daten auslesen: Liste Produkte auslesen, Liste Kategorien auslesen,
68 // Liste Tische, eventData
69 //Produkte den Kategorien zuordnen (siehe allgemeineFunktionen.js)
70 var verzogerung = auslesen();
71
72 //Fehler Meldung, falls eines oder mehrere fehlschlagen
73 verzogerung.fail(function(fehler) {
74 console.log("KRITISCHER FEHLER: Laden der Event-Daten fehlgeschlagen: " + fehler);
75
76 var fehlerAnzeige = $("

</p><br>").text("KRITISCHER FEHLER: Laden der Event-Daten fehlgeschlagen: " + fehler);
77 $("#fehler").append(fehlerAnzeige);
78
79 ansicht("fehler");
80 });
81
82
83 // .....
84 // ..... INITIALISIERUNG .....
85 // .....
86 //beginnen, nachdem Event-Daten geladen und vorbereitet sind
87 verzogerung.done(function() {
88 console.log("Laden des Events erfolgreich: Eventname: " + eventName +
89 " , Anz. Produkte: " + produkte.length +
90 " , Anz. Kategorien: " + kategorien.length +
91 " , Anz. Tische: " + tische.length);
92
93 //AUSLESEN:
94 //alle 7s einmal auslesen (10s ist der Timeout der dateiAufrufen-Funktion)
95 //auslesenFehler() überwacht Fehler und
96 // bricht Interval ab nach zu vielen aufeinanderfolgenden Fehlern
97 auslesenInterval = setInterval(function() {
98 neueAuslesen();
99
100 //stellt sicher, dass neueAuslesen() nicht zu viele Male in Folge fehlschlägt
101 auslesenErfolg++;
102 if (auslesenErfolg > 5) {
103 auslesenErfolg = 5;
104 }
105 }, 7000);
106 //schon einmal auslesen
107 neueAuslesen();
108
109
110 // ..... WEITERE INITIALISIERUNG .....
111 //bestellungenJQ initialisieren als JQ-Objekt
112 bestellungenJQ = $();
113
114 //alles fertig geladen: Ansicht setzen und so auch Ladeanimation verstecken
115 ansicht("inhalt");
116
117 }); //verzogerungAlle.done hier fertig
118 }); //document ready function hier fertig
119
120
121 // =====
122 // ===== FUNKTIONEN =====
123 // =====
124
125 // .....
126 // ..... ALLGEMEIN .....
127 // .....
128 //siehe auch: allgemeineFunktionen.js
129
130 //Ansicht wechseln
131 function ansicht(ansichtNeu) {
132 //Entwicklungs-Fehler-Überprüfung
133 if (typeof(anspruchNeu) !== "string") {
134 console.log("Fehler in ansicht(anspruchNeu)! Angabe kein String:", anspruchNeu);
135 return;
136 }
137
138 $("body").children().hide(); //Altes verstecken
139 laden(false); //Ladeanimation sicher ausblenden
140
141 //Ansicht öffnen und nötige Aktionen ausführen
142 switch(anspruchNeu) {
143 case "inhalt":
144 $("#inhalt").show();
145 break;
146
147 case "fehler":
148 $("#fehler").show();
149 break;
150
151 case "laden":
152 $("#laden").show();
153 break;


```

```

154
155     default:
156         console.log("Ansicht \"" + ansichtNeu + "\"" nicht abgedeckt durch switch.");
157         ansicht("inhalt");
158         break;
159     }
160 }
161
162 //Ladebildschirm ein-/ausblenden: nur überlagern, ansicht bleibt
163 //stat (bool): Ladebildschirm ein-/ausblenden
164 function laden(stat/* = false*/) {
165     if (stat === undefined) {
166         stat = false;
167     }
168
169     if (stat) {
170         $("#laden").show()
171     } else {
172         $("#laden").hide()
173     }
174 }
175
176 //Stellt sicher, dass neueAuslesen() nicht zu viele mal nacheinander fehlschlägt
177 //auslesenInterval wurde in Initialisierung aktiviert
178 function auslesenFehler(fehler/* = ""*/, zwingen/* = false*/) {
179     if (fehler === undefined) {
180         fehler = "";
181     }
182     if (zwingen === undefined) {
183         zwingen = false;
184     }
185
186     auslesenErfolg -= 2; //2 abziehen, weil 1 hinzugefügt wird im interval
187
188     if (auslesenErfolg < 1 || zwingen) { //also =0
189         clearInterval(auslesenInterval);
190
191         ansicht("fehler");
192
193         console.log("KRITISCHER FEHLER: in neueAuslesen(): auslesen zu viele Male fehlgeschlagen: " + fehler);
194
195         var fehlerAnzeige = $("

<p><br>").text(
196             "KRITISCHER FEHLER: in neueAuslesen(): auslesen zu viele Male fehlgeschlagen: " + fehler);
197         $("#fehler").append(fehlerAnzeige);
198     }
199     console.log("auslesenInterval fehlgeschlagen. Zähler (kritischer Fehler bei 0): " + auslesenErfolg);
200
201     return auslesenErfolg;
202 }
203
204 // .....
205 // ..... BESTELLUNGEN AUSLESEN & ANZEIGEN .....
206 // .....
207
208 //neue Bestellungen auslesen und anzeigen, falls sie nicht schon sind
209 //falls nichts zum Anzeigen, Bestellung direkt als ausgeliefert markieren
210 function neueAuslesen() {
211     var verzogerung = auslesenAusliefern(zeigeKategorien, function(bestellungen) {
212         //Bestellungen, die nicht angezeigt werden müssen,
213         // unten als ausgeliefert markieren
214         var unwichtig = [];
215
216         var neuId = []; //neu angezeigte IDs, nur für console.log()
217
218         //bestellungen: Bestellungen, deren verarbeitet-Array
219         // nicht nur '1' enthält (min noch ein '0')
220         for(var i = 0; i < bestellungen.length; i++) {
221             var bestellung = bestellungen[i];
222
223             //nur falls nicht schon angezeigt
224             if (!angezeigteBestellungen.includes(bestellung.id)) {
225                 var wichtig = bestellungAnzeigen(bestellung); //Bestellung anzeigen
226                 if (wichtig) {
227                     //Bestellungs-Id zu angezeigten hinzufügen
228                     angezeigteBestellungen.push(bestellung.id);
229                     neuId.push(bestellung.id);
230                 } else {
231                     //falls keine Relevanten Produkte in der Bestellung:
232                     // sofort als verarbeitet markieren
233                     unwichtig.push(bestellung.id);
234                 }
235             }
236         }
237     }
238
239     console.log("neueAuslesen(): neu angezeigte IDs:", neuId);
240
241     //unwichtige Bestellungen direkt als ausgeliefert markieren
242     if (unwichtig.length > 0) {
243         //Daten zum Updaten erstellen
244         var data = {
245             id: unwichtig,
246             ausgeliefert: zeigeKategorien
247         }
248         var verzogerungUpdaten = bestellungenUpdaten(data, function(anzZeilen) {
249             if (anzZeilen < 1) { //sollte nie vorkommen
250                 verzogerungUpdaten.reject("FEHLER:
251                     0 Zeilen aktualisiert in DB. Bestellungen existierten vermutlich nicht (mehr) in DB.");
252                 return; //FEHLER, HIER ABBRECHEN
253             }
254             console.log("neueAuslesen(): Bestellungen DIREKT als ausgeliefert markiert:", unwichtig);
255         }); //hier fertig: Callback bestellungenUpdaten
256         //falls Fehler:


```

```

257         verzogerungUpdates.fail(function(fehler) {
258             ansicht("fehler");
259
260             console.log("KRITISCHER FEHLER: Direktes Updaten der Bestellungen (" + unwichtig + ") fehlgeschlagen: " +
                fehler);
261
262             var fehlerAnzeige = $("

<<p><br>

").text("KRITISCHER FEHLER: Updaten der Bestellungen (" +
                JSON.stringify(unwichtig) + ") fehlgeschlagen: " + fehler);
263             $("#fehler").append(fehlerAnzeige);
264         });
265     });
266 });
267
268 //falls Fehler:
269 verzogerung.fail(function(fehler) {
270     auslesenFehler(fehler); //evt. Interval stoppen (siehe auslesenFehler-Funktion)
271 });
272 }
273
274 //Anzeigen einer Bestellung
275 //(Slide-Animation vorhanden, aber standardmässig ausgefahren)
276 //bestellung: Bestellungs-Objekt
277 //return: ture für angezeigt, false für nichts zum anzeigen
278 function bestellungAnzeigen(bestellung) {
279     var id = bestellung.id; //zur einfacheren Erreichbarkeit
280
281     //evt. ganze Bestellung nicht anzeigen, falls Bestellung leer
282     // (d.h. kein Produkt soll angezeigt werden) (siehe unten)
283     var bestellungLeer = true;
284
285     //Tabelle erstellen
286     var tabelle = $("




```

```

355 //Tabelle auf Seite pappen
356 $("#alleBestellungen").append(tabelle);
357
358 angezeigteBestellungen.push(bestellung.id);
359 //Tabelle als jquery zu den angezeigten Bestellungen hinzufügen
360 bestellungenJQ = bestellungenJQ.add(tabelle);
361
362 return true; //true bedeutet angezeigt
363 }
364
365
366 // .....
367 // ..... AUSGELIEFERT UND LÖSCHEN .....
368 // .....
369
370 //Bestellung als ausgeliefert markieren und löschen von Bildschirm
371 //id: ID von Bestellung
372 function ausgeliefert(id) {
373 //Tabelle aus Ansicht löschen
374 var tabelleVerz = new $.Deferred();
375 var tabelle = bestellungenJQ.filter("#b" + id); //Tabelle finden
376 slideUp(tabelle, 200, function() { //Tabelle slideUp-Animation
377 //Tabelle erst löschen, wenn sicher, dass sie nicht mehr gebraucht wird
378 tabelleVerz.done(function() {
379 tabelle.remove(); //Tabelle löschen von Bildschirm
380 //Tabelle aus bestellungenJQ löschen
381 bestellungenJQ = bestellungenJQ.not(tabelle);
382 });
383 tabelleVerz.fail(function() {
384 tabelle.show(); //Tabelle wieder anzeigen (ohne Animation)
385 });
386 });
387
388 //DB updaten
389 var verzogerung = bestellungenUpdaten({
390 //Where
391 id: id,
392 //Werte
393 ausgeliefert: zeigeKategorien
394 }, function(anzZeilen) {
395 //ID aus angezeigteBestellungen-Array löschen
396 var index = angezeigteBestellungen.indexOf(id);
397 if (index > -1) {
398 angezeigteBestellungen.splice(index, 1);
399 } else { //sollte nie vorkommen
400 console.log("ANMERKUNG: in ausgeliefert(id): als ausgeliefert zu markierende id (" + id +
401 ") existierte nicht im angezeigteBestellungen-Array", angezeigteBestellungen);
402
403 tabelleVerz.resolve(); //Tabelle endgültig löschen
404
405 console.log("Bestellung als ausgeliefert markiert: " + id + ",
406 Anz. upgedateter Zeilen (sollte 1 sein): " + anzZeilen);
407 });
408 //FEHLERMELDUNG falls DB-Updates fehlschlägt
409 verzogerung.fail(function(fehler) {
410 ansicht("fehler");
411
412 var fehlerAnzeige = $("

</p><br>").text("Updates der Bestellungen fehlgeschlagen: " + fehler);
413 $("#fehler").append(fehlerAnzeige);
414
415 console.log("Updates der Bestellung fehlgeschlagen: " + fehler);
416
417 //Tabelle wieder anzeigen (nur theoretisch, da ansicht("fehler"))
418 tabelleVerz.reject();
419 });
420 }


```