YUM YUM Delivery

Technische Dokumentation

Von

Sophia Deckhut, Max Kaufmann, Luis Bernhardt, Justin Hubert

Inhalt

[Struktur der Webseite 3](#_Toc139958857)

[Frontend 4](#_Toc139958858)

[Aufbau der Webseite mit HTML 4](#_Toc139958859)

[Gestaltung der Webseite 6](#_Toc139958860)

[Java Script 6](#_Toc139958861)

[Backend 9](#_Toc139958862)

[Mitarbeiterseite 10](#_Toc139958863)

[Frontend 10](#_Toc139958864)

[JavaScript 11](#_Toc139958865)

[NodeJS 12](#_Toc139958866)

[Rendern der Seiten 12](#_Toc139958867)

[Auslesen JSON 13](#_Toc139958868)

[User-Handling 13](#_Toc139958869)

[Endpunkte 14](#_Toc139958870)

# **Struktur der Webseite**

Unser Projekt besteht aus zwei Hauptseiten und drei Nebenseiten. Die Hauptseiten sind die Kundebestellseite und die Mitarbeiterseite. Die Nebenseiten umfassen die Impressumsseite, die Bestelldatenseite und die Loginseite. Die einzelnen Seiten sind durch Buttons miteinander verknüpft, sodass beim Klicken auf einen Button eine Weiterleitung auf eine andere Seite erfolgt.

Die Kundebestellseite verwendet zwei CSS-Dateien, um eine bessere Übersichtlichkeit zu gewährleisten. Eine CSS-Datei (Radiobutton.css) ist für die Darstellung der Check- und Radiobuttons sowie des Warenkorbs zuständig. Die andere CSS-Datei (Bestellbearbeitung.css) enthält Stilinformationen für den Header, das Register, den Footer und die Startseite.

Darüber hinaus werden zwei JavaScript-Dateien eingebunden, die für die Warenkorb- und Button-Logik verantwortlich sind.

Die Mitarbeiterseite verwendet nur eine CSS-Datei und ist mit der Loginseite verbunden, da der Zugriff nur mit gültigen Anmeldeinformationen möglich ist.

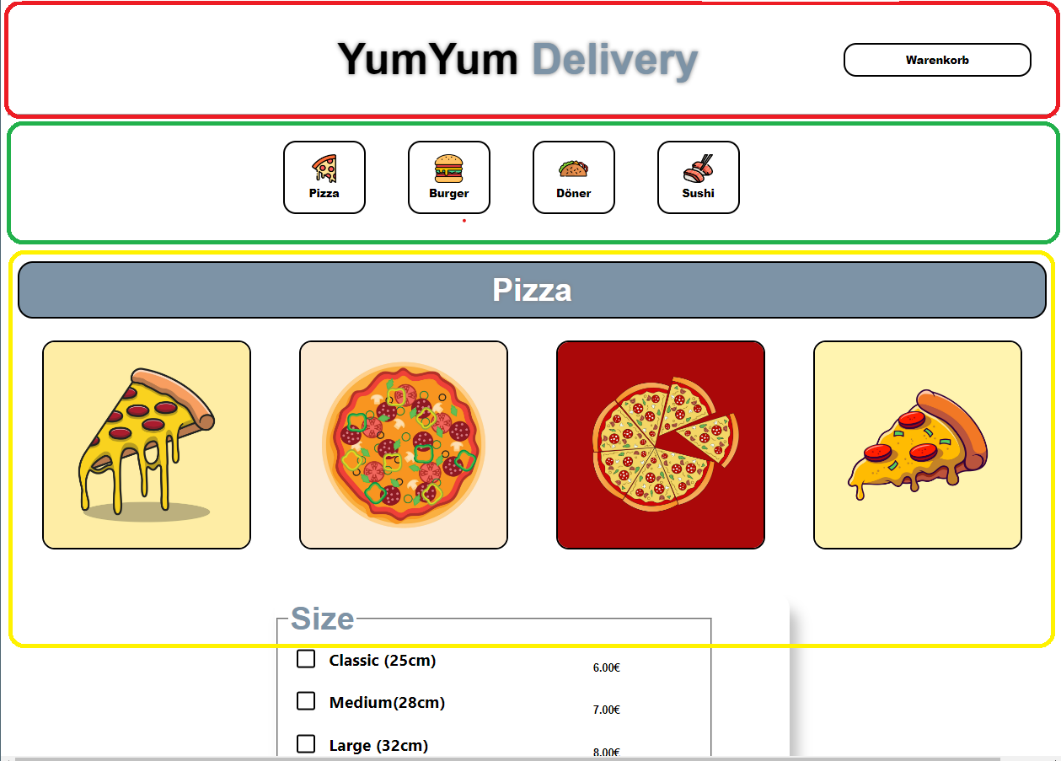
Die Impressums- und Bestelldatenseiten haben jeweils ihre eigenen CSS-Dateien, um eine klare Trennung von der Hauptseite zu gewährleisten und den Stil der Nebenseiten leicht zu ändern.

Im Gegensatz zur Impressumsseite haben die Bestelldatenseite und die Loginseite JavaScript-Einbindungen, die für die Validierung von Kundendaten und Benutzereingaben zuständig sind.

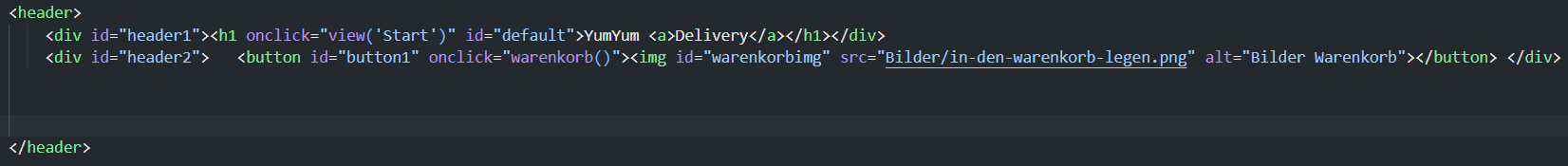
**Kundenbestellseite**

## **Frontend**

### **Aufbau der Webseite mit HTML**

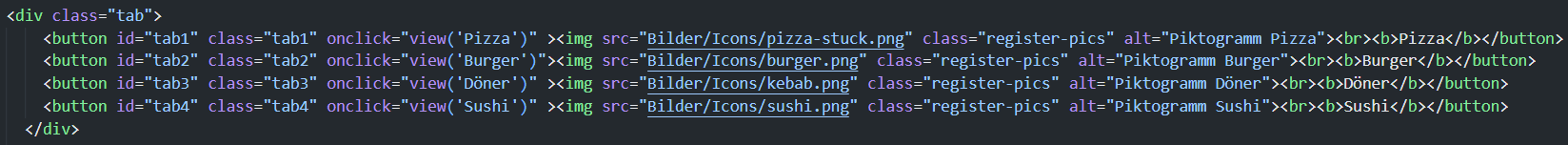


Die Kundebestellseite ist in vier Bereiche unterteilt: Header (rotes Rechteck), Register (grünes Rechteck), Hauptbereich (gelbes Rechteck) und Footer.



Der Header besteht aus zwei Div-Elementen, die jeweils eine Überschrift und den Warenkorb-Button enthalten. Jedes Div hat eine eigene ID, über die sie mit CSS gestaltet werden können.

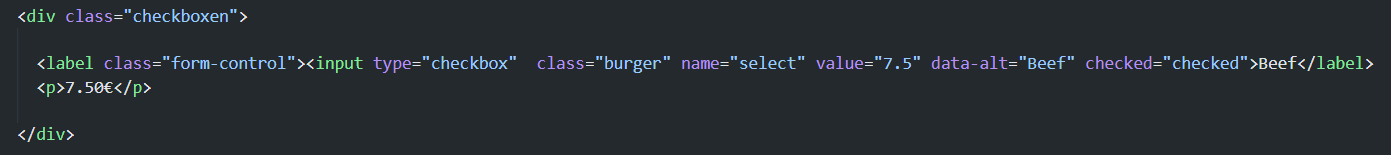
Der Registerbereich (grün) befindet sich innerhalb des Main-Tags. Er besteht aus einem eigenen Div, in dem sich die vier Buttons befinden. Jeder Button enthält ein Icon, wie im obigen Bild gezeigt.



Der Hauptbereich ist der größte Bereich der Kundebestellseite. Dort befinden sich die Registerbuttons und für jedes Produkt (Pizza, Burger, Döner, Sushi) ein eigener Container. Beim Laden der Seite werden automatisch alle Divs für die Bestellung ausgeblendet, außer der Startseite.

Die Produkt-Divs haben eine einheitliche Struktur. Zuerst wird eine Überschrift eingefügt, gefolgt von einem Container für Bilder und einem weiteren Container für Checkboxen und Radiobuttons. Es gibt auch Auswahlmöglichkeiten, die nur einmal ausgewählt werden dürfen.

Die Input-Tags haben verschiedene Attribute, darunter das Style-Attribut, das angibt, ob es sich um eine Checkbox oder einen Radiobutton handelt. Die Inputs mit dem Style-Wert "Radio" haben auch denselben Namen, sodass die Radiobuttons miteinander verbunden sind. Die Attribute Value und Data-alt werden für die Logik des Warenkorbs verwendet.



Der Warenkorb befindet sich ebenfalls im Main-Tag und wird genauso wie die Produkt-Divs standardmäßig ausgeblendet. Er wird nur sichtbar, wenn der Warenkorb-Button betätigt wird. Der Warenkorb ist ein eigener Container mit weiteren Tags.



Im Footer befinden sich die Adresse des Unternehmens und zwei weitere Buttons. Einer leitet zur Login-Seite weiter, der andere zur Impressumsseite.

### **Gestaltung der Webseite**

Jeder Tag wurde mittels CSS gestaltet, um eine ansprechende und elegante Webseite zu erhalten. Es wurde darauf geachtet, die Webseite möglichst responsiv zu gestalten. Das Layout wurde ebenfalls mit CSS angepasst.

Der Header wurde mit "display: flex" zu einer Flexbox gemacht, um die Kinderelemente nebeneinander zu positionieren. Diese Methode wurde bei vielen anderen Divs verwendet. Um die Webseite möglichst responsiv zu gestalten, wurden weitgehend prozentuale Längenangaben anstelle von Pixelwerten verwendet. Für die Schriftgröße wurde die Einheit "vw" genutzt, damit sich die Schrift in Abhängigkeit von der Fenstergröße automatisch anpasst. Um zu verhindern, dass Tags beliebig klein zusammengezogen werden, wurden die Eigenschaften "min-width" und "min-height" eingefügt, um die Mindestgröße der Tags festzulegen.

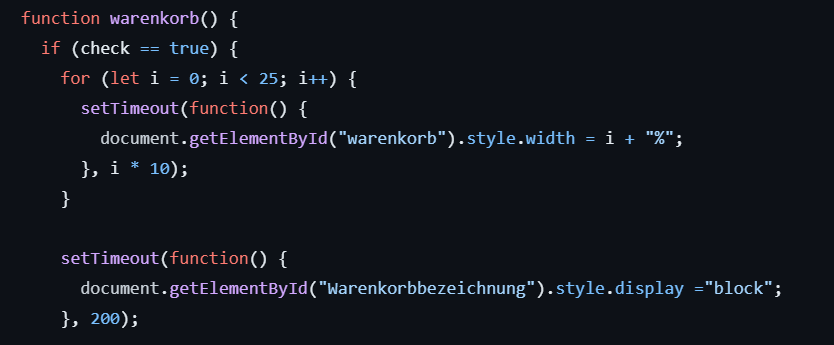
Bei der Gestaltung wurden hauptsächlich die Farben Schwarz und Weiß verwendet, mit Ausnahme der Überschriften und des Warenkorbs, für die ein helleres Blau und Gelb verwendet wurden.

### **Java Script**

Webseite abbilden. Es gibt die Datei "Registerkarte.js", die beim Klicken auf einen Tab-Button die anderen Tabs ausblendet und nur das ausgewählte Produkt anzeigt. Sie enthält auch die Aktion für den Impressumsbutton.

Die eigentliche Logik befindet sich in der Datei "checkInJson.js". Diese Datei enthält verschiedene Funktionen.

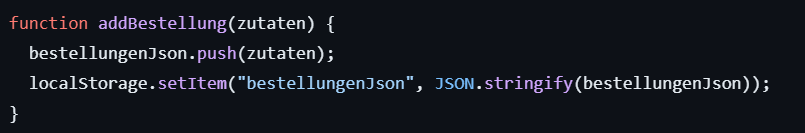
**"warenkorb()":** Diese Funktion wird beim Klicken auf den Warenkorb-Button aufgerufen und sorgt für die Animation des Warenkorbs, indem die Breite des Divs schrittweise erhöht wird. Die "setTimeout"-Funktion sorgt für Pausen zwischen den Schritten.



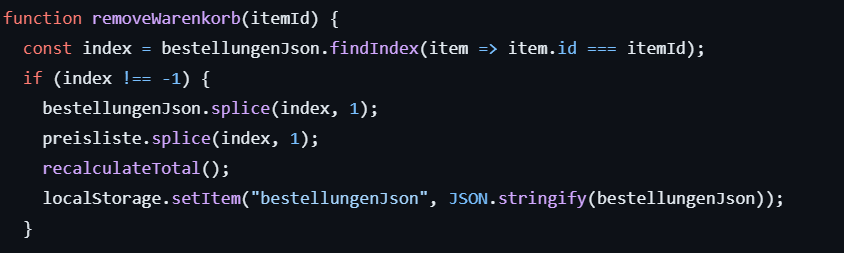
**"closeWarenkorb()"**: Diese Funktion verringert die zuvor erhöhte Breite des Warenkorbs wieder, um ihn zu schließen.



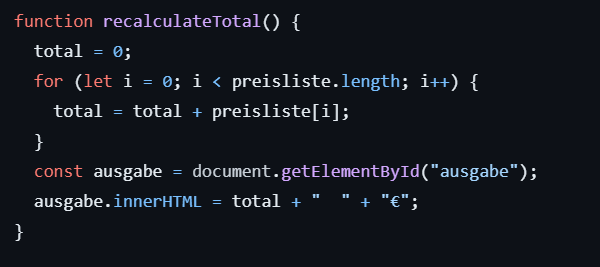
**"addBestellung()":** Hier werden die Zutaten dem "bestellungenJson"-Array mit der "push"-Methode hinzugefügt. Das Array wird auch im "localStorage" gespeichert.



**"removeWarenkorb()":** Diese Funktion entfernt die Divs mit der angegebenen ID aus dem Warenkorb und ruft die Methode "recalculateTotal()" auf.



**"recalculateTotal()":** Diese Funktion berechnet den neuen Gesamtwert des Warenkorbs, nachdem eine Bestellung aus dem Warenkorb entfernt wurde.

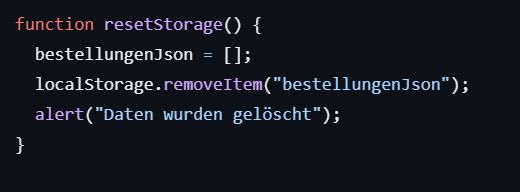


**"calculateBurger(), calculatePizza(), calculateDoner(), calculateSushi()":** In diesen Methoden wird einerseits der Preis für die Bestellungen berechnet, die mit dem "In den Warenkorb" Button hinzugefügt werden. Dazu werden alle ausgewählten Checkboxen abgerufen und die Value-Werte der einzelnen Elemente addiert. Die Preise der individuellen Bestellungen werden separat in einem Preislisten-Array gespeichert. Anschließend wird für jede Bestellung ein Div-Element mit einer eindeutigen ID erstellt, die aus "div" plus dem aktuellen Datum besteht, um sicherzustellen, dass jede ID eindeutig ist. Zum Schluss werden die einzelnen Zutaten mithilfe von DOM-Manipulation in das HTML-Dokument eingefügt und mit der Methode "addBestellung()" im Array gespeichert. Der Preis wird ausgegeben und alle Häkchen werden zurückgesetzt.

**"storage()":** In dieser Funktion werden die im "localStorage" gespeicherten Daten in das "bestellungenJson"-Array geladen, um sicherzustellen, dass die Daten auch beim Wechseln der Webseite erhalten bleiben.



**"resetStorage()":** Diese Funktion löscht den Inhalt des "bestellungenJson"-Arrays und des "localStorage", um alle Inhalte nach dem Bestellen zurückzusetzen.



**"sendToServer()":** Diese Funktion sendet die Daten über einen FETCH-Aufruf an den Server, um sie in einer JSON-Datei zu speichern.



**"validate()":** Diese Funktion stellt sicher, dass bei der Eingabe von Adresse und Zahlungsart alle Pflichtfelder ausgefüllt sind.

## **Backend**

Das Backend ist für das Laden der HTML-Seiten und das Empfangen der per FETCH gesendeten Daten und das Schreiben dieser Daten in eine JSON-Datei zuständig.

Um die HTML-Seiten zu laden, wurden drei verschiedene Request-Handler implementiert. Jeder Handler lädt eine andere HTML-Seite für unterschiedliche Routen mithilfe von "app.render". Dafür wurde zunächst ein Template namens "eta" installiert und verwendet.



Die per FETCH empfangenen Daten werden von einem "app.post" Handler entgegengenommen. Zunächst werden die bereits in der JSON-Datei enthaltenen Daten ausgelesen, und dann werden die neuen Daten angehängt. Anschließend werden die Daten mithilfe von "fs.writeFile" wieder in die Datei geschrieben.



# **Mitarbeiterseite**

## **Frontend**

Das Frontend beinhaltet zunächst die Login Seite, bei der man mithilfe von „fetch“ eine POST-Anfrage mit den Logindaten an das Backend senden kann. Falls diese Erfolgreich validiert werden, bekommt man als Antwort ein Token, mit dem man Zugriff auf die Mitarbeiterseite erhält und wird auf die entsprechende Seite weitergeleitet.

Die Mitarbeiterseite an sich dient zur Darstellung der offenen Bestellungen, so wie den bereits abgearbeiteten Bestellungen – der „Bestellhistorie“.

Die offenen Bestellungen werden in einer Tabelle gelistet und haben jeweils auf der rechten Seite eine Checkbox welche als Value die „ID“ der Bestellung hält. Es gibt einen „processed“-Button, welcher alle über die Checkboxen angehakten Bestellungen aus den Bestellungen raus löscht und der Bestellhistorie anhängt. Die Bestellhistorie wird rechts neben den Bestellungen ebenfalls in einer Tabelle dargestellt. Falls die Listen zu lang werden, gibt es unter der Tabelle einen Button um wieder nach oben zu scrollen. Zusätzlich gibt es einen „refresh“-Button und einen „logout“-button welcher das Token für die Validierung raus löscht und die Seite refresht.

Die Gestaltung der beiden Seiten wird über jeweils eine eigene CSS-Datei übernommen.

## **Ein Bild, das Text, Elektronik, Screenshot, Display enthält. Automatisch generierte BeschreibungJavaScript**

"login()": Schickt die Login Anfrage der Login Seite ans Backend und handelt die Antwort.

**fetchProtectedContent():** Verifiziert das Token.

**getOrders():** Fordert alle offenen Bestellungen vom Backend an.

**getHistory():** Fordert die Bestellhistorie vom Backend an.

**processed():** Schickt die bearbeiteten Daten ans Backend und fordert die aktualisierten Listen an.

**logout():** Löscht das Verify-Token und refresht die Seite.

**clear(div):** Entfernt alle Elemente der beiden Listen (Bestellliste und Bestellhistorie). Wird benötigt, bevor z.B. bei einer Aktualisierung alle Elemente neu generiert und hinzugrfügt werden.

**scrollUp():** Scrollt von ganz unten der Seite nach ganz oben.

## **NodeJS**

Constant-Part

Ein Bild, das Text, Screenshot enthält.

Automatisch generierte BeschreibungEnthält alle nötigen Imports und Konstanten für das Backend. Dazu gehören bspw. „jsonwebtoken“ sowie der „secretKey“, welche für die Verifizierung des Mitarbeiter-Logins benötigt werden.

### **Ein Bild, das Text, Screenshot, Schrift enthält. Automatisch generierte BeschreibungRendern der Seiten**

Die Zwei Methoden zum Rendern der beiden Seiten:

### **Auslesen JSON**

Ein Bild, das Text, Screenshot, Software enthält.

Automatisch generierte BeschreibungEs gibt drei Methoden zum Auslesen, von JSON-Dateien, da wir der Einfachheit halber auf eine Datenbank verzichtet haben und alle Daten lokal in JSON-Files speichern. Die erste Methode ist „getUsers()“ und liefert alle eingetragenen User, welche später für die Validierung des Logins benötigt werden. Falls kein entsprechendes JSON-File existiert, wird ein solches erstellt und zwei „Beispiel“-User eingetragen. Die anderen beiden Funktionen sind „getOrders()“ und „getHistory()“ welche fast gleich aufgebaut sind. Allerdings wird nur eine leere Datei erzeugt, falls keine gefunden wird ohne Beispieldaten.

### **User-Handling**

Ein Bild, das Text, Screenshot, Display, Software enthält.

Automatisch generierte BeschreibungÜber die Funktion „findUser(username, password)“ wird überprüft ob ein User mit den als Parameterübergebenen Login-Daten innerhalb einer JSON mit allen Usern existiert. Mithilfe der Funktion „getUserById(id)“ werden anhand einer User-ID die restlichen Daten des Users herausgesucht und zurückgegeben (Username, Password).

### **Ein Bild, das Text, Screenshot enthält. Automatisch generierte BeschreibungEndpunkte**

„/login“:

empfängt Username und Passwort, validiert diese und schickt, falls gültig, einen JsonWebToken an den Client, welcher für eine Stunde gültig ist. Nach dieser Stunde muss der Client sich erneut einloggen um Zugriff auf die Mitarbeiterseite zu erhalten

„/getorders“: sendet alle Bestellungen, welche in dem entsprechenden JSON-File gespeichert sind zurück.

„/gethistory“: sendet die Bestellhistorie, welche in dem entsprechenden JSON-File gespeichert ist, zurück.

„/protected“: kontrolliert mithilfe einer Middleware, ob das mitgesendete JsonWebToken gültig ist. Falls das der Fall ist, sendet es die Rückgabe „valid=true“ und den Username des Users an den Client.

„/processed“: empfängt eine Liste mit fertig bearbeiteten Bestellungen und verschiebt diese in die Bestellhistorie

**Ein Bild, das Text, Screenshot, Display, Software enthält.

Automatisch generierte BeschreibungMiddlware**

Die Middleware „authenticateToken“ verifiziert mithilfe der in dem Modul „jsonwebtoken“ Funktion „verify()“ das mitgesendete Token. Falls dabei ein Fehler Auftritt wird ein Fehler „returned“ bevor „next()“ aufgerufen werden kann und somit wird der Endpunkt „/protected“ nicht erreicht.

Handling von fertigen Bestellungen

Ein Bild, das Text, Screenshot, Display, Software enthält.

Automatisch generierte Beschreibung

In der Funktion „handleOrders(ordersProcessed)“ werden die in einer Liste übergebenen Bestellungen in den aktuell eingetragenen Bestellungen gesucht. Falls sie gefunden werden, wird die Bestellung der Bestellhistorie hinzugefügt und aus der aktuellen Liste entfernt. Zum Schluss werden die aktualisierten Daten in die entsprechende Speicherdatei geschrieben.